src/ - nginx-1.7.10

- core/
- event/
- http/
- mail/
- misc/
- os/

src/core/ - nginx-1.7.10

- nginx.c
- nginx.h
- ngx_array.c
- ngx_array.h
- ngx_buf.c
- ngx_buf.h
- ngx_conf_file.c
- ngx_conf_file.h
- ngx_config.h
- ngx_connection.c
- ngx_connection.h
- ngx_core.h
- ngx_cpuinfo.c
- ngx_crc.h
- ngx_crc32.c
- ngx_crc32.h
- ngx_crypt.c
- ngx_crypt.h
- ngx_cycle.c
- ngx_cycle.h
- ngx_file.c
- ngx_file.h
- ngx_hash.c
- ngx_hash.h
- ngx_inet.c
- ngx_inet.h
- ngx_list.c
- ngx_list.h
- ngx_log.c
- ngx_log.h
- ngx_md5.c

- ngx_md5.h
- ngx_murmurhash.c
- ngx_murmurhash.h
- ngx_open_file_cache.c
- ngx_open_file_cache.h
- ngx_output_chain.c
- ngx_palloc.c
- ngx_palloc.h
- ngx_parse.c
- ngx_parse.h
- ngx_proxy_protocol.c
- ngx_proxy_protocol.h
- ngx_queue.c
- ngx_queue.h
- ngx_radix_tree.c
- ngx_radix_tree.h
- ngx_rbtree.c
- ngx_rbtree.h
- ngx_regex.c
- ngx_regex.h
- ngx_resolver.c
- ngx_resolver.h
- ngx_sha1.h
- ngx_shmtx.c
- ngx_shmtx.h
- ngx_slab.c
- ngx_slab.h
- ngx_spinlock.c
- ngx_string.c
- ngx_string.h
- ngx_syslog.c
- ngx_syslog.h
- ngx_times.c

- ngx_times.h

One Level Up Top Level

src/core/nginx.c - nginx-1.7.10

Global variables defined

- ngx_conf_file
- ngx_conf_params
- ngx_core_commands
- ngx_core_module
- ngx_core_module_ctx
- ngx_debug_points
- ngx_max_module
- ngx_os_environ
- ngx_prefix
- ngx_show_configure
- ngx_show_help
- ngx_show_version
- ngx_signal

Functions defined

- main
- ngx_add_inherited_sockets
- ngx_core_module_create_conf
- ngx_core_module_init_conf
- ngx_exec_new_binary
- ngx_get_cpu_affinity
- ngx_get_options
- ngx_process_options
- ngx_save_argv
- ngx_set_cpu_affinity
- ngx_set_env
- ngx_set_environment
- ngx_set_priority
- ngx_set_user
- ngx_set_worker_processes

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <nginx.h>
11
12
13  static ngx_int_t ngx_add_inherited_sockets(ngx_cycle_t *cycle);
14  static ngx_int_t ngx_get_options(int argc, char *const *argv);
15  static ngx_int_t ngx_process_options(ngx_cycle_t *cycle);
16  static ngx_int_t ngx_save_argv(ngx_cycle_t *cycle, int argc, char *const *argv);
17  static void *ngx_core_module_create_conf(ngx_cycle_t *cycle);
18  static char *ngx_core_module_init_conf(ngx_cycle_t *cycle, void *conf);
19  static char *ngx_set_user(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
20  static char *ngx_set_env(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
21  static char *ngx_set_priority(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
22  static char *ngx_set_cpu_affinity(ngx_conf_t *cf, ngx_command_t *cmd,
23      void *conf);
24  static char *ngx_set_worker_processes(ngx_conf_t *cf, ngx_command_t *cmd,
25      void *conf);
26
27
28  static ngx_conf_enum_t  ngx_debug_points[] = {
29      { ngx_string("stop"), NGX_DEBUG_POINTS_STOP },
30      { ngx_string("abort"), NGX_DEBUG_POINTS_ABORT },
31      { ngx_null_string, 0 }
32  };
33
34
35  static ngx_command_t  ngx_core_commands[] = {
36
37      { ngx_string("daemon"),
38        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_FLAG,
39        ngx_conf_set_flag_slot,
40        0,
41        offsetof(ngx_core_conf_t, daemon),
42        NULL },
43
44      { ngx_string("master_process"),
45        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_FLAG,
46        ngx_conf_set_flag_slot,
47        0,
48        offsetof(ngx_core_conf_t, master),
49        NULL },
50
51      { ngx_string("timer_resolution"),
52        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
53        ngx_conf_set_msec_slot,
54        0,
55        offsetof(ngx_core_conf_t, timer_resolution),
56        NULL },
57
58      { ngx_string("pid"),
59        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
60        ngx_conf_set_str_slot,
61        0,
62        offsetof(ngx_core_conf_t, pid),
63        NULL },
64
65      { ngx_string("lock_file"),
66        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
67        ngx_conf_set_str_slot,
68        0,
69        offsetof(ngx_core_conf_t, lock_file),
70        NULL },
71
72      { ngx_string("worker_processes"),
73        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
```

```
      ngx_set_worker_processes,
      0,
      0,
      NULL },

    { ngx_string("debug_points"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      0,
      offsetof(ngx_core_conf_t, debug_points),
      &ngx_debug_points },

    { ngx_string("user"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE12,
      ngx_set_user,
      0,
      0,
      NULL },

    { ngx_string("worker_priority"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_set_priority,
      0,
      0,
      NULL },

    { ngx_string("worker_cpu_affinity"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_1MORE,
      ngx_set_cpu_affinity,
      0,
      0,
      NULL },

    { ngx_string("worker_rlimit_nofile"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      0,
      offsetof(ngx_core_conf_t, rlimit_nofile),
      NULL },

    { ngx_string("worker_rlimit_core"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_off_slot,
      0,
      offsetof(ngx_core_conf_t, rlimit_core),
      NULL },

    { ngx_string("worker_rlimit_sigpending"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      0,
      offsetof(ngx_core_conf_t, rlimit_sigpending),
      NULL },

    { ngx_string("working_directory"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      0,
      offsetof(ngx_core_conf_t, working_directory),
      NULL },

    { ngx_string("env"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_set_env,
      0,
      0,
      NULL },

#if (NGX_THREADS)

    { ngx_string("worker_threads"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      0,
      offsetof(ngx_core_conf_t, worker_threads),
      NULL },
```

```
150
151     { ngx_string("thread_stack_size"),
152       NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
153       ngx_conf_set_size_slot,
154       0,
155       offsetof(ngx_core_conf_t, thread_stack_size),
156       NULL },
157
158 #endif
159
160       ngx_null_command
161 };
162
163
164 static ngx_core_module_t  ngx_core_module_ctx = {
165     ngx_string("core"),
166     ngx_core_module_create_conf,
167     ngx_core_module_init_conf
168 };
169
170
171 ngx_module_t  ngx_core_module = {
172     NGX_MODULE_V1,
173     &ngx_core_module_ctx,                  /* module context */
174     ngx_core_commands,                     /* module directives */
175     NGX_CORE_MODULE,                       /* module type */
176     NULL,                                  /* init master */
177     NULL,                                  /* init module */
178     NULL,                                  /* init process */
179     NULL,                                  /* init thread */
180     NULL,                                  /* exit thread */
181     NULL,                                  /* exit process */
182     NULL,                                  /* exit master */
183     NGX_MODULE_V1_PADDING
184 };
185
186
187 ngx_uint_t          ngx_max_module;
188
189 static ngx_uint_t   ngx_show_help;
190 static ngx_uint_t   ngx_show_version;
191 static ngx_uint_t   ngx_show_configure;
192 static u_char      *ngx_prefix;
193 static u_char      *ngx_conf_file;
194 static u_char      *ngx_conf_params;
195 static char        *ngx_signal;
196
197
198 static char **ngx_os_environ;
199
200
201 int ngx_cdecl
202 main(int argc, char *const *argv)
203 {
204     ngx_int_t         i;
205     ngx_log_t        *log;
206     ngx_cycle_t      *cycle, init_cycle;
207     ngx_core_conf_t  *ccf;
208
209     ngx_debug_init();
210
211     if (ngx_strerror_init() != NGX_OK) {
212         return 1;
213     }
214
215     if (ngx_get_options(argc, argv) != NGX_OK) {
216         return 1;
217     }
218
219     if (ngx_show_version) {
220         ngx_write_stderr("nginx version: " NGINX_VER_BUILD NGX_LINEFEED);
221
222         if (ngx_show_help) {
223             ngx_write_stderr(
224                 "Usage: nginx [-?hvVtq] [-s signal] [-c filename] "
225                              "[-p prefix] [-g directives]" NGX_LINEFEED
```

```
                            NGX_LINEFEED
                "Options:" NGX_LINEFEED
                "  -?,-h         : this help" NGX_LINEFEED
                "  -v            : show version and exit" NGX_LINEFEED
                "  -V            : show version and configure options then exit"
                                   NGX_LINEFEED
                "  -t            : test configuration and exit" NGX_LINEFEED
                "  -q            : suppress non-error messages "
                                   "during configuration testing" NGX_LINEFEED
                "  -s signal     : send signal to a master process: "
                                   "stop, quit, reopen, reload" NGX_LINEFEED
#ifdef NGX_PREFIX
                "  -p prefix     : set prefix path (default: "
                                   NGX_PREFIX ")" NGX_LINEFEED
#else
                "  -p prefix     : set prefix path (default: NONE)" NGX_LINEFEED
#endif
                "  -c filename   : set configuration file (default: "
                                   NGX_CONF_PATH ")" NGX_LINEFEED
                "  -g directives : set global directives out of configuration "
                                   "file" NGX_LINEFEED NGX_LINEFEED
                );
        }

        if (ngx_show_configure) {
            ngx_write_stderr(
#ifdef NGX_COMPILER
                "built by " NGX_COMPILER NGX_LINEFEED
#endif
#if (NGX_SSL)
#ifdef SSL_CTRL_SET_TLSEXT_HOSTNAME
                "TLS SNI support enabled" NGX_LINEFEED
#else
                "TLS SNI support disabled" NGX_LINEFEED
#endif
#endif
                "configure arguments:" NGX_CONFIGURE NGX_LINEFEED);
        }

        if (!ngx_test_config) {
            return 0;
        }
    }

    /* TODO */ ngx_max_sockets = -1;

    ngx_time_init();

#if (NGX_PCRE)
    ngx_regex_init();
#endif

    ngx_pid = ngx_getpid();

    log = ngx_log_init(ngx_prefix);
    if (log == NULL) {
        return 1;
    }

    /* STUB */
#if (NGX_OPENSSL)
    ngx_ssl_init(log);
#endif

    /*
     * init_cycle->log is required for signal handlers and
     * ngx_process_options()
     */

    ngx_memzero(&init_cycle, sizeof(ngx_cycle_t));
    init_cycle.log = log;
    ngx_cycle = &init_cycle;

    init_cycle.pool = ngx_create_pool(1024, log);
    if (init_cycle.pool == NULL) {
        return 1;
```

```
302     }
303
304     if (ngx_save_argv(&init_cycle, argc, argv) != NGX_OK) {
305         return 1;
306     }
307
308     if (ngx_process_options(&init_cycle) != NGX_OK) {
309         return 1;
310     }
311
312     if (ngx_os_init(log) != NGX_OK) {
313         return 1;
314     }
315
316     /*
317      * ngx_crc32_table_init() requires ngx_cacheline_size set in ngx_os_init()
318      */
319
320     if (ngx_crc32_table_init() != NGX_OK) {
321         return 1;
322     }
323
324     if (ngx_add_inherited_sockets(&init_cycle) != NGX_OK) {
325         return 1;
326     }
327
328     ngx_max_module = 0;
329     for (i = 0; ngx_modules[i]; i++) {
330         ngx_modules[i]->index = ngx_max_module++;
331     }
332
333     cycle = ngx_init_cycle(&init_cycle);
334     if (cycle == NULL) {
335         if (ngx_test_config) {
336             ngx_log_stderr(0, "configuration file %s test failed",
337                            init_cycle.conf_file.data);
338         }
339
340         return 1;
341     }
342
343     if (ngx_test_config) {
344         if (!ngx_quiet_mode) {
345             ngx_log_stderr(0, "configuration file %s test is successful",
346                            cycle->conf_file.data);
347         }
348
349         return 0;
350     }
351
352     if (ngx_signal) {
353         return ngx_signal_process(cycle, ngx_signal);
354     }
355
356     ngx_os_status(cycle->log);
357
358     ngx_cycle = cycle;
359
360     ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
361
362     if (ccf->master && ngx_process == NGX_PROCESS_SINGLE) {
363         ngx_process = NGX_PROCESS_MASTER;
364     }
365
366 #if !(NGX_WIN32)
367
368     if (ngx_init_signals(cycle->log) != NGX_OK) {
369         return 1;
370     }
371
372     if (!ngx_inherited && ccf->daemon) {
373         if (ngx_daemon(cycle->log) != NGX_OK) {
374             return 1;
375         }
376
377         ngx_daemonized = 1;
```

```
    }

    if (ngx_inherited) {
        ngx_daemonized = 1;
    }

#endif

    if (ngx_create_pidfile(&ccf->pid, cycle->log) != NGX_OK) {
        return 1;
    }

    if (ngx_log_redirect_stderr(cycle) != NGX_OK) {
        return 1;
    }

    if (log->file->fd != ngx_stderr) {
        if (ngx_close_file(log->file->fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          ngx_close_file_n " built-in log failed");
        }
    }

    ngx_use_stderr = 0;

    if (ngx_process == NGX_PROCESS_SINGLE) {
        ngx_single_process_cycle(cycle);

    } else {
        ngx_master_process_cycle(cycle);
    }

    return 0;
}


static ngx_int_t
ngx_add_inherited_sockets(ngx_cycle_t *cycle)
{
    u_char           *p, *v, *inherited;
    ngx_int_t         s;
    ngx_listening_t  *ls;

    inherited = (u_char *) getenv(NGINX_VAR);

    if (inherited == NULL) {
        return NGX_OK;
    }

    ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0,
                  "using inherited sockets from \"%s\"", inherited);

    if (ngx_array_init(&cycle->listening, cycle->pool, 10,
                       sizeof(ngx_listening_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    for (p = inherited, v = p; *p; p++) {
        if (*p == ':' || *p == ';') {
            s = ngx_atoi(v, p - v);
            if (s == NGX_ERROR) {
                ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
                              "invalid socket number \"%s\" in " NGINX_VAR
                              " environment variable, ignoring the rest"
                              " of the variable", v);
                break;
            }

            v = p + 1;

            ls = ngx_array_push(&cycle->listening);
            if (ls == NULL) {
                return NGX_ERROR;
            }
```

```
            ngx_memzero(ls, sizeof(ngx_listening_t));

            ls->fd = (ngx_socket_t) s;
        }
    }

    ngx_inherited = 1;

    return ngx_set_inherited_sockets(cycle);
}


char **
ngx_set_environment(ngx_cycle_t *cycle, ngx_uint_t *last)
{
    char             **p, **env;
    ngx_str_t         *var;
    ngx_uint_t         i, n;
    ngx_core_conf_t   *ccf;

    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);

    if (last == NULL && ccf->environment) {
        return ccf->environment;
    }

    var = ccf->env.elts;

    for (i = 0; i < ccf->env.nelts; i++) {
        if (ngx_strcmp(var[i].data, "TZ") == 0
            || ngx_strncmp(var[i].data, "TZ=", 3) == 0)
        {
            goto tz_found;
        }
    }

    var = ngx_array_push(&ccf->env);
    if (var == NULL) {
        return NULL;
    }

    var->len = 2;
    var->data = (u_char *) "TZ";

    var = ccf->env.elts;

tz_found:

    n = 0;

    for (i = 0; i < ccf->env.nelts; i++) {

        if (var[i].data[var[i].len] == '=') {
            n++;
            continue;
        }

        for (p = ngx_os_environ; *p; p++) {

            if (ngx_strncmp(*p, var[i].data, var[i].len) == 0
                && (*p)[var[i].len] == '=')
            {
                n++;
                break;
            }
        }
    }

    if (last) {
        env = ngx_alloc((*last + n + 1) * sizeof(char *), cycle->log);
        *last = n;

    } else {
        env = ngx_palloc(cycle->pool, (n + 1) * sizeof(char *));
    }
```

```
530
531     if (env == NULL) {
532         return NULL;
533     }
534
535     n = 0;
536
537     for (i = 0; i < ccf->env.nelts; i++) {
538
539         if (var[i].data[var[i].len] == '=') {
540             env[n++] = (char *) var[i].data;
541             continue;
542         }
543
544         for (p = ngx_os_environ; *p; p++) {
545
546             if (ngx_strncmp(*p, var[i].data, var[i].len) == 0
547                 && (*p)[var[i].len] == '=')
548             {
549                 env[n++] = *p;
550                 break;
551             }
552         }
553     }
554
555     env[n] = NULL;
556
557     if (last == NULL) {
558         ccf->environment = env;
559         environ = env;
560     }
561
562     return env;
563 }
564
565
566 ngx_pid_t
567 ngx_exec_new_binary(ngx_cycle_t *cycle, char *const *argv)
568 {
569     char             **env, *var;
570     u_char            *p;
571     ngx_uint_t         i, n;
572     ngx_pid_t          pid;
573     ngx_exec_ctx_t     ctx;
574     ngx_core_conf_t   *ccf;
575     ngx_listening_t   *ls;
576
577     ngx_memzero(&ctx, sizeof(ngx_exec_ctx_t));
578
579     ctx.path = argv[0];
580     ctx.name = "new binary process";
581     ctx.argv = argv;
582
583     n = 2;
584     env = ngx_set_environment(cycle, &n);
585     if (env == NULL) {
586         return NGX_INVALID_PID;
587     }
588
589     var = ngx_alloc(sizeof(NGINX_VAR)
590                     + cycle->listening.nelts * (NGX_INT32_LEN + 1) + 2,
591                     cycle->log);
592     if (var == NULL) {
593         ngx_free(env);
594         return NGX_INVALID_PID;
595     }
596
597     p = ngx_cpymem(var, NGINX_VAR "=", sizeof(NGINX_VAR));
598
599     ls = cycle->listening.elts;
600     for (i = 0; i < cycle->listening.nelts; i++) {
601         p = ngx_sprintf(p, "%ud;", ls[i].fd);
602     }
603
604     *p = '\0';
605
```

```
606     env[n++] = var;
607
608 #if (NGX_SETPROCTITLE_USES_ENV)
609
610     /* allocate the spare 300 bytes for the new binary process title */
611
612     env[n++] = "SPARE=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
613                "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
614                "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
615                "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
616                "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
617
618 #endif
619
620     env[n] = NULL;
621
622 #if (NGX_DEBUG)
623     {
624     char  **e;
625     for (e = env; *e; e++) {
626         ngx_log_debug1(NGX_LOG_DEBUG_CORE, cycle->log, 0, "env: %s", *e);
627     }
628     }
629 #endif
630
631     ctx.envp = (char *const *) env;
632
633     ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
634
635     if (ngx_rename_file(ccf->pid.data, ccf->oldpid.data) == NGX_FILE_ERROR) {
636         ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
637                       ngx_rename_file_n " %s to %s failed "
638                       "before executing new binary process \"%s\"",
639                       ccf->pid.data, ccf->oldpid.data, argv[0]);
640
641         ngx_free(env);
642         ngx_free(var);
643
644         return NGX_INVALID_PID;
645     }
646
647     pid = ngx_execute(cycle, &ctx);
648
649     if (pid == NGX_INVALID_PID) {
650         if (ngx_rename_file(ccf->oldpid.data, ccf->pid.data)
651             == NGX_FILE_ERROR)
652         {
653             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
654                           ngx_rename_file_n " %s back to %s failed after "
655                           "an attempt to execute new binary process \"%s\"",
656                           ccf->oldpid.data, ccf->pid.data, argv[0]);
657         }
658     }
659
660     ngx_free(env);
661     ngx_free(var);
662
663     return pid;
664 }
665
666
667 static ngx_int_t
668 ngx_get_options(int argc, char *const *argv)
669 {
670     u_char     *p;
671     ngx_int_t   i;
672
673     for (i = 1; i < argc; i++) {
674
675         p = (u_char *) argv[i];
676
677         if (*p++ != '-') {
678             ngx_log_stderr(0, "invalid option: \"%s\"", argv[i]);
679             return NGX_ERROR;
680         }
681
```

```
682         while (*p) {
683
684             switch (*p++) {
685
686             case '?':
687             case 'h':
688                 ngx_show_version = 1;
689                 ngx_show_help = 1;
690                 break;
691
692             case 'v':
693                 ngx_show_version = 1;
694                 break;
695
696             case 'V':
697                 ngx_show_version = 1;
698                 ngx_show_configure = 1;
699                 break;
700
701             case 't':
702                 ngx_test_config = 1;
703                 break;
704
705             case 'q':
706                 ngx_quiet_mode = 1;
707                 break;
708
709             case 'p':
710                 if (*p) {
711                     ngx_prefix = p;
712                     goto next;
713                 }
714
715                 if (argv[++i]) {
716                     ngx_prefix = (u_char *) argv[i];
717                     goto next;
718                 }
719
720                 ngx_log_stderr(0, "option \"-p\" requires directory name");
721                 return NGX_ERROR;
722
723             case 'c':
724                 if (*p) {
725                     ngx_conf_file = p;
726                     goto next;
727                 }
728
729                 if (argv[++i]) {
730                     ngx_conf_file = (u_char *) argv[i];
731                     goto next;
732                 }
733
734                 ngx_log_stderr(0, "option \"-c\" requires file name");
735                 return NGX_ERROR;
736
737             case 'g':
738                 if (*p) {
739                     ngx_conf_params = p;
740                     goto next;
741                 }
742
743                 if (argv[++i]) {
744                     ngx_conf_params = (u_char *) argv[i];
745                     goto next;
746                 }
747
748                 ngx_log_stderr(0, "option \"-g\" requires parameter");
749                 return NGX_ERROR;
750
751             case 's':
752                 if (*p) {
753                     ngx_signal = (char *) p;
754
755                 } else if (argv[++i]) {
756                     ngx_signal = argv[i];
757
```

```
                } else {
                    ngx_log_stderr(0, "option \"-s\" requires parameter");
                    return NGX_ERROR;
                }

                if (ngx_strcmp(ngx_signal, "stop") == 0
                    || ngx_strcmp(ngx_signal, "quit") == 0
                    || ngx_strcmp(ngx_signal, "reopen") == 0
                    || ngx_strcmp(ngx_signal, "reload") == 0)
                {
                    ngx_process = NGX_PROCESS_SIGNALLER;
                    goto next;
                }

                ngx_log_stderr(0, "invalid option: \"-s %s\"", ngx_signal);
                return NGX_ERROR;

            default:
                ngx_log_stderr(0, "invalid option: \"%c\"", *(p - 1));
                return NGX_ERROR;
            }
        }

    next:

        continue;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_save_argv(ngx_cycle_t *cycle, int argc, char *const *argv)
{
#if (NGX_FREEBSD)

    ngx_os_argv = (char **) argv;
    ngx_argc = argc;
    ngx_argv = (char **) argv;

#else
    size_t     len;
    ngx_int_t  i;

    ngx_os_argv = (char **) argv;
    ngx_argc = argc;

    ngx_argv = ngx_alloc((argc + 1) * sizeof(char *), cycle->log);
    if (ngx_argv == NULL) {
        return NGX_ERROR;
    }

    for (i = 0; i < argc; i++) {
        len = ngx_strlen(argv[i]) + 1;

        ngx_argv[i] = ngx_alloc(len, cycle->log);
        if (ngx_argv[i] == NULL) {
            return NGX_ERROR;
        }

        (void) ngx_cpystrn((u_char *) ngx_argv[i], (u_char *) argv[i], len);
    }

    ngx_argv[i] = NULL;

#endif

    ngx_os_environ = environ;

    return NGX_OK;
}


static ngx_int_t
ngx_process_options(ngx_cycle_t *cycle)
```

```
{
    u_char  *p;
    size_t   len;

    if (ngx_prefix) {
        len = ngx_strlen(ngx_prefix);
        p = ngx_prefix;

        if (len && !ngx_path_separator(p[len - 1])) {
            p = ngx_pnalloc(cycle->pool, len + 1);
            if (p == NULL) {
                return NGX_ERROR;
            }

            ngx_memcpy(p, ngx_prefix, len);
            p[len++] = '/';
        }

        cycle->conf_prefix.len = len;
        cycle->conf_prefix.data = p;
        cycle->prefix.len = len;
        cycle->prefix.data = p;

    } else {

#ifndef NGX_PREFIX

        p = ngx_pnalloc(cycle->pool, NGX_MAX_PATH);
        if (p == NULL) {
            return NGX_ERROR;
        }

        if (ngx_getcwd(p, NGX_MAX_PATH) == 0) {
            ngx_log_stderr(ngx_errno, "[emerg]: " ngx_getcwd_n " failed");
            return NGX_ERROR;
        }

        len = ngx_strlen(p);

        p[len++] = '/';

        cycle->conf_prefix.len = len;
        cycle->conf_prefix.data = p;
        cycle->prefix.len = len;
        cycle->prefix.data = p;

#else

#ifdef NGX_CONF_PREFIX
        ngx_str_set(&cycle->conf_prefix, NGX_CONF_PREFIX);
#else
        ngx_str_set(&cycle->conf_prefix, NGX_PREFIX);
#endif
        ngx_str_set(&cycle->prefix, NGX_PREFIX);

#endif
    }

    if (ngx_conf_file) {
        cycle->conf_file.len = ngx_strlen(ngx_conf_file);
        cycle->conf_file.data = ngx_conf_file;

    } else {
        ngx_str_set(&cycle->conf_file, NGX_CONF_PATH);
    }

    if (ngx_conf_full_name(cycle, &cycle->conf_file, 0) != NGX_OK) {
        return NGX_ERROR;
    }

    for (p = cycle->conf_file.data + cycle->conf_file.len - 1;
         p > cycle->conf_file.data;
         p--)
    {
        if (ngx_path_separator(*p)) {
            cycle->conf_prefix.len = p - ngx_cycle->conf_file.data + 1;
```

```
            cycle->conf_prefix.data = ngx_cycle->conf_file.data;
            break;
        }
    }

    if (ngx_conf_params) {
        cycle->conf_param.len = ngx_strlen(ngx_conf_params);
        cycle->conf_param.data = ngx_conf_params;
    }

    if (ngx_test_config) {
        cycle->log->log_level = NGX_LOG_INFO;
    }

    return NGX_OK;
}


static void *
ngx_core_module_create_conf(ngx_cycle_t *cycle)
{
    ngx_core_conf_t  *ccf;

    ccf = ngx_pcalloc(cycle->pool, sizeof(ngx_core_conf_t));
    if (ccf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc()
     *
     *     ccf->pid = NULL;
     *     ccf->oldpid = NULL;
     *     ccf->priority = 0;
     *     ccf->cpu_affinity_n = 0;
     *     ccf->cpu_affinity = NULL;
     */

    ccf->daemon = NGX_CONF_UNSET;
    ccf->master = NGX_CONF_UNSET;
    ccf->timer_resolution = NGX_CONF_UNSET_MSEC;

    ccf->worker_processes = NGX_CONF_UNSET;
    ccf->debug_points = NGX_CONF_UNSET;

    ccf->rlimit_nofile = NGX_CONF_UNSET;
    ccf->rlimit_core = NGX_CONF_UNSET;
    ccf->rlimit_sigpending = NGX_CONF_UNSET;

    ccf->user = (ngx_uid_t) NGX_CONF_UNSET_UINT;
    ccf->group = (ngx_gid_t) NGX_CONF_UNSET_UINT;

#if (NGX_THREADS)
    ccf->worker_threads = NGX_CONF_UNSET;
    ccf->thread_stack_size = NGX_CONF_UNSET_SIZE;
#endif

    if (ngx_array_init(&ccf->env, cycle->pool, 1, sizeof(ngx_str_t))
        != NGX_OK)
    {
        return NULL;
    }

    return ccf;
}


static char *
ngx_core_module_init_conf(ngx_cycle_t *cycle, void *conf)
{
    ngx_core_conf_t  *ccf = conf;

    ngx_conf_init_value(ccf->daemon, 1);
    ngx_conf_init_value(ccf->master, 1);
    ngx_conf_init_msec_value(ccf->timer_resolution, 0);

```

```
    ngx_conf_init_value(ccf->worker_processes, 1);
    ngx_conf_init_value(ccf->debug_points, 0);

#if (NGX_HAVE_CPU_AFFINITY)

    if (ccf->cpu_affinity_n
        && ccf->cpu_affinity_n != 1
        && ccf->cpu_affinity_n != (ngx_uint_t) ccf->worker_processes)
    {
        ngx_log_error(NGX_LOG_WARN, cycle->log, 0,
                      "the number of \"worker_processes\" is not equal to "
                      "the number of \"worker_cpu_affinity\" masks, "
                      "using last mask for remaining worker processes");
    }

#endif

#if (NGX_THREADS)

    ngx_conf_init_value(ccf->worker_threads, 0);
    ngx_threads_n = ccf->worker_threads;
    ngx_conf_init_size_value(ccf->thread_stack_size, 2 * 1024 * 1024);

#endif


    if (ccf->pid.len == 0) {
        ngx_str_set(&ccf->pid, NGX_PID_PATH);
    }

    if (ngx_conf_full_name(cycle, &ccf->pid, 0) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    ccf->oldpid.len = ccf->pid.len + sizeof(NGX_OLDPID_EXT);

    ccf->oldpid.data = ngx_pnalloc(cycle->pool, ccf->oldpid.len);
    if (ccf->oldpid.data == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memcpy(ngx_cpymem(ccf->oldpid.data, ccf->pid.data, ccf->pid.len),
               NGX_OLDPID_EXT, sizeof(NGX_OLDPID_EXT));


#if !(NGX_WIN32)

    if (ccf->user == (uid_t) NGX_CONF_UNSET_UINT && geteuid() == 0) {
        struct group   *grp;
        struct passwd  *pwd;

        ngx_set_errno(0);
        pwd = getpwnam(NGX_USER);
        if (pwd == NULL) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "getpwnam(\"" NGX_USER "\") failed");
            return NGX_CONF_ERROR;
        }

        ccf->username = NGX_USER;
        ccf->user = pwd->pw_uid;

        ngx_set_errno(0);
        grp = getgrnam(NGX_GROUP);
        if (grp == NULL) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "getgrnam(\"" NGX_GROUP "\") failed");
            return NGX_CONF_ERROR;
        }

        ccf->group = grp->gr_gid;
    }


    if (ccf->lock_file.len == 0) {
        ngx_str_set(&ccf->lock_file, NGX_LOCK_PATH);
```

```
1062     }
1063
1064     if (ngx_conf_full_name(cycle, &ccf->lock_file, 0) != NGX_OK) {
1065         return NGX_CONF_ERROR;
1066     }
1067
1068     {
1069     ngx_str_t  lock_file;
1070
1071     lock_file = cycle->old_cycle->lock_file;
1072
1073     if (lock_file.len) {
1074         lock_file.len--;
1075
1076         if (ccf->lock_file.len != lock_file.len
1077             || ngx_strncmp(ccf->lock_file.data, lock_file.data, lock_file.len)
1078                != 0)
1079         {
1080             ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
1081                           "\"lock_file\" could not be changed, ignored");
1082         }
1083
1084         cycle->lock_file.len = lock_file.len + 1;
1085         lock_file.len += sizeof(".accept");
1086
1087         cycle->lock_file.data = ngx_pstrdup(cycle->pool, &lock_file);
1088         if (cycle->lock_file.data == NULL) {
1089             return NGX_CONF_ERROR;
1090         }
1091
1092     } else {
1093         cycle->lock_file.len = ccf->lock_file.len + 1;
1094         cycle->lock_file.data = ngx_pnalloc(cycle->pool,
1095                                             ccf->lock_file.len + sizeof(".accept"));
1096         if (cycle->lock_file.data == NULL) {
1097             return NGX_CONF_ERROR;
1098         }
1099
1100         ngx_memcpy(ngx_cpymem(cycle->lock_file.data, ccf->lock_file.data,
1101                               ccf->lock_file.len),
1102                    ".accept", sizeof(".accept"));
1103     }
1104     }
1105
1106 #endif
1107
1108     return NGX_CONF_OK;
1109 }
1110
1111
1112 static char *
1113 ngx_set_user(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1114 {
1115 #if (NGX_WIN32)
1116
1117     ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1118                        "\"user\" is not supported, ignored");
1119
1120     return NGX_CONF_OK;
1121
1122 #else
1123
1124     ngx_core_conf_t  *ccf = conf;
1125
1126     char             *group;
1127     struct passwd    *pwd;
1128     struct group     *grp;
1129     ngx_str_t        *value;
1130
1131     if (ccf->user != (uid_t) NGX_CONF_UNSET_UINT) {
1132         return "is duplicate";
1133     }
1134
1135     if (geteuid() != 0) {
1136         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1137                            "the \"user\" directive makes sense only "
```

```
                           "if the master process runs "
                           "with super-user privileges, ignored");
        return NGX_CONF_OK;
    }

    value = (ngx_str_t *) cf->args->elts;

    ccf->username = (char *) value[1].data;

    ngx_set_errno(0);
    pwd = getpwnam((const char *) value[1].data);
    if (pwd == NULL) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
                           "getpwnam(\"%s\") failed", value[1].data);
        return NGX_CONF_ERROR;
    }

    ccf->user = pwd->pw_uid;

    group = (char *) ((cf->args->nelts == 2) ? value[1].data : value[2].data);

    ngx_set_errno(0);
    grp = getgrnam(group);
    if (grp == NULL) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
                           "getgrnam(\"%s\") failed", group);
        return NGX_CONF_ERROR;
    }

    ccf->group = grp->gr_gid;

    return NGX_CONF_OK;

#endif
}


static char *
ngx_set_env(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_core_conf_t  *ccf = conf;

    ngx_str_t   *value, *var;
    ngx_uint_t   i;

    var = ngx_array_push(&ccf->env);
    if (var == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;
    *var = value[1];

    for (i = 0; i < value[1].len; i++) {

        if (value[1].data[i] == '=') {

            var->len = i;

            return NGX_CONF_OK;
        }
    }

    return NGX_CONF_OK;
}


static char *
ngx_set_priority(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_core_conf_t  *ccf = conf;

    ngx_str_t        *value;
    ngx_uint_t        n, minus;

    if (ccf->priority != 0) {
```

```
        return "is duplicate";
    }

    value = cf->args->elts;

    if (value[1].data[0] == '-') {
        n = 1;
        minus = 1;

    } else if (value[1].data[0] == '+') {
        n = 1;
        minus = 0;

    } else {
        n = 0;
        minus = 0;
    }

    ccf->priority = ngx_atoi(&value[1].data[n], value[1].len - n);
    if (ccf->priority == NGX_ERROR) {
        return "invalid number";
    }

    if (minus) {
        ccf->priority = -ccf->priority;
    }

    return NGX_CONF_OK;
}


static char *
ngx_set_cpu_affinity(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
#if (NGX_HAVE_CPU_AFFINITY)
    ngx_core_conf_t  *ccf = conf;

    u_char            ch;
    uint64_t         *mask;
    ngx_str_t        *value;
    ngx_uint_t        i, n;

    if (ccf->cpu_affinity) {
        return "is duplicate";
    }

    mask = ngx_palloc(cf->pool, (cf->args->nelts - 1) * sizeof(uint64_t));
    if (mask == NULL) {
        return NGX_CONF_ERROR;
    }

    ccf->cpu_affinity_n = cf->args->nelts - 1;
    ccf->cpu_affinity = mask;

    value = cf->args->elts;

    for (n = 1; n < cf->args->nelts; n++) {

        if (value[n].len > 64) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "\"worker_cpu_affinity\" supports up to 64 CPUs only");
            return NGX_CONF_ERROR;
        }

        mask[n - 1] = 0;

        for (i = 0; i < value[n].len; i++) {

            ch = value[n].data[i];

            if (ch == ' ') {
                continue;
            }

            mask[n - 1] <<= 1;

```

```
1290            if (ch == '0') {
1291                continue;
1292            }
1293
1294            if (ch == '1') {
1295                mask[n - 1] |= 1;
1296                continue;
1297            }
1298
1299            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1300                               "invalid character \"%c\" in \"worker_cpu_affinity\"",
1301                               ch);
1302            return NGX_CONF_ERROR;
1303        }
1304    }
1305
1306 #else
1307
1308    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1309                       "\"worker_cpu_affinity\" is not supported "
1310                       "on this platform, ignored");
1311 #endif
1312
1313    return NGX_CONF_OK;
1314 }
1315
1316
1317 uint64_t
1318 ngx_get_cpu_affinity(ngx_uint_t n)
1319 {
1320    ngx_core_conf_t  *ccf;
1321
1322    ccf = (ngx_core_conf_t *) ngx_get_conf(ngx_cycle->conf_ctx,
1323                                           ngx_core_module);
1324
1325    if (ccf->cpu_affinity == NULL) {
1326        return 0;
1327    }
1328
1329    if (ccf->cpu_affinity_n > n) {
1330        return ccf->cpu_affinity[n];
1331    }
1332
1333    return ccf->cpu_affinity[ccf->cpu_affinity_n - 1];
1334 }
1335
1336
1337 static char *
1338 ngx_set_worker_processes(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1339 {
1340    ngx_str_t        *value;
1341    ngx_core_conf_t  *ccf;
1342
1343    ccf = (ngx_core_conf_t *) conf;
1344
1345    if (ccf->worker_processes != NGX_CONF_UNSET) {
1346        return "is duplicate";
1347    }
1348
1349    value = (ngx_str_t *) cf->args->elts;
1350
1351    if (ngx_strcmp(value[1].data, "auto") == 0) {
1352        ccf->worker_processes = ngx_ncpu;
1353        return NGX_CONF_OK;
1354    }
1355
1356    ccf->worker_processes = ngx_atoi(value[1].data, value[1].len);
1357
1358    if (ccf->worker_processes == NGX_ERROR) {
1359        return "invalid value";
1360    }
1361
1362    return NGX_CONF_OK;
1363 }
```

One Level Up Top Level

src/http/modules/perl/ngx_http_perl_module.h - nginx-1.7.10

Data types defined

- nginx
- ngx_http_perl_ctx_t
- ngx_http_perl_var_t

Macros defined

- _NGX_HTTP_PERL_MODULE_H_INCLUDED_
- dTHXa
- dTHXa

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_PERL_MODULE_H_INCLUDED_
9  #define _NGX_HTTP_PERL_MODULE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15  #include <nginx.h>
16
17  #include <EXTERN.h>
18  #include <perl.h>
19
20
21  typedef ngx_http_request_t   *nginx;
22
23  typedef struct {
24      ngx_str_t                 filename;
25      ngx_str_t                 redirect_uri;
26      ngx_str_t                 redirect_args;
27
28      SV                       *next;
29
30      ngx_uint_t                done;       /* unsigned  done:1; */
31
32      ngx_array_t              *variables;  /* array of ngx_http_perl_var_t */
33
34  #if (NGX_HTTP_SSI)
35      ngx_http_ssi_ctx_t       *ssi;
36  #endif
37  } ngx_http_perl_ctx_t;
38
39
40  typedef struct {
41      ngx_uint_t    hash;
42      ngx_str_t     name;
43      ngx_str_t     value;
44  } ngx_http_perl_var_t;
45
46
47  extern ngx_module_t  ngx_http_perl_module;
48
```

```
49
50 /*
51  * workaround for "unused variable `Perl___notused'" warning
52  * when building with perl 5.6.1
53  */
54 #ifndef PERL_IMPLICIT_CONTEXT
55 #undef  dTHXa
56 #define dTHXa(a)
57 #endif
58
59
60 extern void boot_DynaLoader(pTHX_ CV* cv);
61
62
63 void ngx_http_perl_handle_request(ngx_http_request_t *r);
64 void ngx_http_perl_sleep_handler(ngx_http_request_t *r);
65
66
67 #endif /* _NGX_HTTP_PERL_MODULE_H_INCLUDED_ */
```

One Level Up Top Level

src/http/modules/perl/ - nginx-1.7.10

- nginx.pm
- ngx_http_perl_module.c
- ngx_http_perl_module.h

src/http/modules/ - nginx-1.7.10

- ngx_http_access_module.c
- ngx_http_addition_filter_module.c
- ngx_http_auth_basic_module.c
- ngx_http_auth_request_module.c
- ngx_http_autoindex_module.c
- ngx_http_browser_module.c
- ngx_http_charset_filter_module.c
- ngx_http_chunked_filter_module.c
- ngx_http_dav_module.c
- ngx_http_degradation_module.c
- ngx_http_empty_gif_module.c
- ngx_http_fastcgi_module.c
- ngx_http_flv_module.c
- ngx_http_geo_module.c
- ngx_http_geoip_module.c
- ngx_http_gunzip_filter_module.c
- ngx_http_gzip_filter_module.c
- ngx_http_gzip_static_module.c
- ngx_http_headers_filter_module.c
- ngx_http_image_filter_module.c
- ngx_http_index_module.c
- ngx_http_limit_conn_module.c
- ngx_http_limit_req_module.c
- ngx_http_log_module.c
- ngx_http_map_module.c
- ngx_http_memcached_module.c
- ngx_http_mp4_module.c
- ngx_http_not_modified_filter_module.c
- ngx_http_proxy_module.c
- ngx_http_random_index_module.c
- ngx_http_range_filter_module.c

- ngx_http_realip_module.c
- ngx_http_referer_module.c
- ngx_http_rewrite_module.c
- ngx_http_scgi_module.c
- ngx_http_secure_link_module.c
- ngx_http_split_clients_module.c
- ngx_http_ssi_filter_module.c
- ngx_http_ssi_filter_module.h
- ngx_http_ssl_module.c
- ngx_http_ssl_module.h
- ngx_http_static_module.c
- ngx_http_stub_status_module.c
- ngx_http_sub_filter_module.c
- ngx_http_upstream_hash_module.c
- ngx_http_upstream_ip_hash_module.c
- ngx_http_upstream_keepalive_module.c
- ngx_http_upstream_least_conn_module.c
- ngx_http_userid_filter_module.c
- ngx_http_uwsgi_module.c
- ngx_http_xslt_filter_module.c
- perl/

One Level Up Top Level

src/http/ - nginx-1.7.10

- modules/
- ngx_http.c
- ngx_http.h
- ngx_http_busy_lock.c
- ngx_http_busy_lock.h
- ngx_http_cache.h
- ngx_http_config.h
- ngx_http_copy_filter_module.c
- ngx_http_core_module.c
- ngx_http_core_module.h
- ngx_http_file_cache.c
- ngx_http_header_filter_module.c
- ngx_http_parse.c
- ngx_http_parse_time.c
- ngx_http_postpone_filter_module.c
- ngx_http_request.c
- ngx_http_request.h
- ngx_http_request_body.c
- ngx_http_script.c
- ngx_http_script.h
- ngx_http_spdy.c
- ngx_http_spdy.h
- ngx_http_spdy_filter_module.c
- ngx_http_spdy_module.c
- ngx_http_spdy_module.h
- ngx_http_special_response.c
- ngx_http_upstream.c
- ngx_http_upstream.h
- ngx_http_upstream_round_robin.c
- ngx_http_upstream_round_robin.h
- ngx_http_variables.c

- ngx_http_variables.h
- ngx_http_write_filter_module.c

One Level Up Top Level

src/http/ngx_http.c - nginx-1.7.10

Global variables defined

- ngx_http_commands
- ngx_http_html_default_types
- ngx_http_max_module
- ngx_http_module
- ngx_http_module_ctx
- ngx_http_top_body_filter
- ngx_http_top_header_filter

Functions defined

- ngx_http_add_address
- ngx_http_add_addresses
- ngx_http_add_addrs
- ngx_http_add_addrs6
- ngx_http_add_listen
- ngx_http_add_listening
- ngx_http_add_location
- ngx_http_add_server
- ngx_http_block
- ngx_http_cmp_conf_addrs
- ngx_http_cmp_dns_wildcards
- ngx_http_cmp_locations
- ngx_http_create_locations_list
- ngx_http_create_locations_tree
- ngx_http_init_headers_in_hash
- ngx_http_init_listening
- ngx_http_init_locations
- ngx_http_init_phase_handlers
- ngx_http_init_phases
- ngx_http_init_static_location_trees
- ngx_http_join_exact_locations

- ngx_http_merge_locations
- ngx_http_merge_servers
- ngx_http_merge_types
- ngx_http_optimize_servers
- ngx_http_server_names
- ngx_http_set_default_types
- ngx_http_types_slot

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static char *ngx_http_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
14  static ngx_int_t ngx_http_init_phases(ngx_conf_t *cf,
15      ngx_http_core_main_conf_t *cmcf);
16  static ngx_int_t ngx_http_init_headers_in_hash(ngx_conf_t *cf,
17      ngx_http_core_main_conf_t *cmcf);
18  static ngx_int_t ngx_http_init_phase_handlers(ngx_conf_t *cf,
19      ngx_http_core_main_conf_t *cmcf);
20
21  static ngx_int_t ngx_http_add_addresses(ngx_conf_t *cf,
22      ngx_http_core_srv_conf_t *cscf, ngx_http_conf_port_t *port,
23      ngx_http_listen_opt_t *lsopt);
24  static ngx_int_t ngx_http_add_address(ngx_conf_t *cf,
25      ngx_http_core_srv_conf_t *cscf, ngx_http_conf_port_t *port,
26      ngx_http_listen_opt_t *lsopt);
27  static ngx_int_t ngx_http_add_server(ngx_conf_t *cf,
28      ngx_http_core_srv_conf_t *cscf, ngx_http_conf_addr_t *addr);
29
30  static char *ngx_http_merge_servers(ngx_conf_t *cf,
31      ngx_http_core_main_conf_t *cmcf, ngx_http_module_t *module,
32      ngx_uint_t ctx_index);
33  static char *ngx_http_merge_locations(ngx_conf_t *cf,
34      ngx_queue_t *locations, void **loc_conf, ngx_http_module_t *module,
35      ngx_uint_t ctx_index);
36  static ngx_int_t ngx_http_init_locations(ngx_conf_t *cf,
37      ngx_http_core_srv_conf_t *cscf, ngx_http_core_loc_conf_t *pclcf);
38  static ngx_int_t ngx_http_init_static_location_trees(ngx_conf_t *cf,
39      ngx_http_core_loc_conf_t *pclcf);
40  static ngx_int_t ngx_http_cmp_locations(const ngx_queue_t *one,
41      const ngx_queue_t *two);
42  static ngx_int_t ngx_http_join_exact_locations(ngx_conf_t *cf,
43      ngx_queue_t *locations);
44  static void ngx_http_create_locations_list(ngx_queue_t *locations,
45      ngx_queue_t *q);
46  static ngx_http_location_tree_node_t *
47      ngx_http_create_locations_tree(ngx_conf_t *cf, ngx_queue_t *locations,
48      size_t prefix);
49
50  static ngx_int_t ngx_http_optimize_servers(ngx_conf_t *cf,
51      ngx_http_core_main_conf_t *cmcf, ngx_array_t *ports);
52  static ngx_int_t ngx_http_server_names(ngx_conf_t *cf,
53      ngx_http_core_main_conf_t *cmcf, ngx_http_conf_addr_t *addr);
54  static ngx_int_t ngx_http_cmp_conf_addrs(const void *one, const void *two);
55  static int ngx_libc_cdecl ngx_http_cmp_dns_wildcards(const void *one,
56      const void *two);
```

```
57
58  static ngx_int_t ngx_http_init_listening(ngx_conf_t *cf,
59      ngx_http_conf_port_t *port);
60  static ngx_listening_t *ngx_http_add_listening(ngx_conf_t *cf,
61      ngx_http_conf_addr_t *addr);
62  static ngx_int_t ngx_http_add_addrs(ngx_conf_t *cf, ngx_http_port_t *hport,
63      ngx_http_conf_addr_t *addr);
64  #if (NGX_HAVE_INET6)
65  static ngx_int_t ngx_http_add_addrs6(ngx_conf_t *cf, ngx_http_port_t *hport,
66      ngx_http_conf_addr_t *addr);
67  #endif
68
69  ngx_uint_t   ngx_http_max_module;
70
71
72  ngx_int_t  (*ngx_http_top_header_filter) (ngx_http_request_t *r);
73  ngx_int_t  (*ngx_http_top_body_filter) (ngx_http_request_t *r, ngx_chain_t *ch);
74
75
76  ngx_str_t  ngx_http_html_default_types[] = {
77      ngx_string("text/html"),
78      ngx_null_string
79  };
80
81
82  static ngx_command_t  ngx_http_commands[] = {
83
84      { ngx_string("http"),
85        NGX_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
86        ngx_http_block,
87        0,
88        0,
89        NULL },
90
91        ngx_null_command
92  };
93
94
95  static ngx_core_module_t  ngx_http_module_ctx = {
96      ngx_string("http"),
97      NULL,
98      NULL
99  };
100
101
102  ngx_module_t  ngx_http_module = {
103      NGX_MODULE_V1,
104      &ngx_http_module_ctx,                  /* module context */
105      ngx_http_commands,                     /* module directives */
106      NGX_CORE_MODULE,                       /* module type */
107      NULL,                                  /* init master */
108      NULL,                                  /* init module */
109      NULL,                                  /* init process */
110      NULL,                                  /* init thread */
111      NULL,                                  /* exit thread */
112      NULL,                                  /* exit process */
113      NULL,                                  /* exit master */
114      NGX_MODULE_V1_PADDING
115  };
116
117
118  static char *
119  ngx_http_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
120  {
121      char                        *rv;
122      ngx_uint_t                   mi, m, s;
123      ngx_conf_t                   pcf;
124      ngx_http_module_t           *module;
125      ngx_http_conf_ctx_t         *ctx;
126      ngx_http_core_loc_conf_t    *clcf;
127      ngx_http_core_srv_conf_t   **cscfp;
128      ngx_http_core_main_conf_t   *cmcf;
129
130      /* the main http context */
131
132      ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_conf_ctx_t));
```

```
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    *(ngx_http_conf_ctx_t **) conf = ctx;


    /* count the number of the http modules and set up their indices */

    ngx_http_max_module = 0;
    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
            continue;
        }

        ngx_modules[m]->ctx_index = ngx_http_max_module++;
    }


    /* the http main_conf context, it is the same in the all http contexts */

    ctx->main_conf = ngx_pcalloc(cf->pool,
                                 sizeof(void *) * ngx_http_max_module);
    if (ctx->main_conf == NULL) {
        return NGX_CONF_ERROR;
    }


    /*
     * the http null srv_conf context, it is used to merge
     * the server{}s' srv_conf's
     */

    ctx->srv_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->srv_conf == NULL) {
        return NGX_CONF_ERROR;
    }


    /*
     * the http null loc_conf context, it is used to merge
     * the server{}s' loc_conf's
     */

    ctx->loc_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->loc_conf == NULL) {
        return NGX_CONF_ERROR;
    }


    /*
     * create the main_conf's, the null srv_conf's, and the null loc_conf's
     * of the all http modules
     */

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;
        mi = ngx_modules[m]->ctx_index;

        if (module->create_main_conf) {
            ctx->main_conf[mi] = module->create_main_conf(cf);
            if (ctx->main_conf[mi] == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        if (module->create_srv_conf) {
            ctx->srv_conf[mi] = module->create_srv_conf(cf);
            if (ctx->srv_conf[mi] == NULL) {
                return NGX_CONF_ERROR;
            }
        }
```

```
209
210         if (module->create_loc_conf) {
211             ctx->loc_conf[mi] = module->create_loc_conf(cf);
212             if (ctx->loc_conf[mi] == NULL) {
213                 return NGX_CONF_ERROR;
214             }
215         }
216     }
217
218     pcf = *cf;
219     cf->ctx = ctx;
220
221     for (m = 0; ngx_modules[m]; m++) {
222         if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
223             continue;
224         }
225
226         module = ngx_modules[m]->ctx;
227
228         if (module->preconfiguration) {
229             if (module->preconfiguration(cf) != NGX_OK) {
230                 return NGX_CONF_ERROR;
231             }
232         }
233     }
234
235     /* parse inside the http{} block */
236
237     cf->module_type = NGX_HTTP_MODULE;
238     cf->cmd_type = NGX_HTTP_MAIN_CONF;
239     rv = ngx_conf_parse(cf, NULL);
240
241     if (rv != NGX_CONF_OK) {
242         goto failed;
243     }
244
245     /*
246      * init http{} main_conf's, merge the server{}s' srv_conf's
247      * and its location{}s' loc_conf's
248      */
249
250     cmcf = ctx->main_conf[ngx_http_core_module.ctx_index];
251     cscfp = cmcf->servers.elts;
252
253     for (m = 0; ngx_modules[m]; m++) {
254         if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
255             continue;
256         }
257
258         module = ngx_modules[m]->ctx;
259         mi = ngx_modules[m]->ctx_index;
260
261         /* init http{} main_conf's */
262
263         if (module->init_main_conf) {
264             rv = module->init_main_conf(cf, ctx->main_conf[mi]);
265             if (rv != NGX_CONF_OK) {
266                 goto failed;
267             }
268         }
269
270         rv = ngx_http_merge_servers(cf, cmcf, module, mi);
271         if (rv != NGX_CONF_OK) {
272             goto failed;
273         }
274     }
275
276
277     /* create location trees */
278
279     for (s = 0; s < cmcf->servers.nelts; s++) {
280
281         clcf = cscfp[s]->ctx->loc_conf[ngx_http_core_module.ctx_index];
282
283         if (ngx_http_init_locations(cf, cscfp[s], clcf) != NGX_OK) {
284             return NGX_CONF_ERROR;
```

```
285         }
286
287         if (ngx_http_init_static_location_trees(cf, clcf) != NGX_OK) {
288             return NGX_CONF_ERROR;
289         }
290     }
291
292
293     if (ngx_http_init_phases(cf, cmcf) != NGX_OK) {
294         return NGX_CONF_ERROR;
295     }
296
297     if (ngx_http_init_headers_in_hash(cf, cmcf) != NGX_OK) {
298         return NGX_CONF_ERROR;
299     }
300
301
302     for (m = 0; ngx_modules[m]; m++) {
303         if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
304             continue;
305         }
306
307         module = ngx_modules[m]->ctx;
308
309         if (module->postconfiguration) {
310             if (module->postconfiguration(cf) != NGX_OK) {
311                 return NGX_CONF_ERROR;
312             }
313         }
314     }
315
316     if (ngx_http_variables_init_vars(cf) != NGX_OK) {
317         return NGX_CONF_ERROR;
318     }
319
320     /*
321      * http{}'s cf->ctx was needed while the configuration merging
322      * and in postconfiguration process
323      */
324
325     *cf = pcf;
326
327
328     if (ngx_http_init_phase_handlers(cf, cmcf) != NGX_OK) {
329         return NGX_CONF_ERROR;
330     }
331
332
333     /* optimize the lists of ports, addresses and server names */
334
335     if (ngx_http_optimize_servers(cf, cmcf, cmcf->ports) != NGX_OK) {
336         return NGX_CONF_ERROR;
337     }
338
339     return NGX_CONF_OK;
340
341 failed:
342
343     *cf = pcf;
344
345     return rv;
346 }
347
348
349 static ngx_int_t
350 ngx_http_init_phases(ngx_conf_t *cf, ngx_http_core_main_conf_t *cmcf)
351 {
352     if (ngx_array_init(&cmcf->phases[NGX_HTTP_POST_READ_PHASE].handlers,
353                        cf->pool, 1, sizeof(ngx_http_handler_pt))
354         != NGX_OK)
355     {
356         return NGX_ERROR;
357     }
358
359     if (ngx_array_init(&cmcf->phases[NGX_HTTP_SERVER_REWRITE_PHASE].handlers,
360                        cf->pool, 1, sizeof(ngx_http_handler_pt))
```

```
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&cmcf->phases[NGX_HTTP_REWRITE_PHASE].handlers,
                       cf->pool, 1, sizeof(ngx_http_handler_pt))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&cmcf->phases[NGX_HTTP_PREACCESS_PHASE].handlers,
                       cf->pool, 1, sizeof(ngx_http_handler_pt))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&cmcf->phases[NGX_HTTP_ACCESS_PHASE].handlers,
                       cf->pool, 2, sizeof(ngx_http_handler_pt))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers,
                       cf->pool, 4, sizeof(ngx_http_handler_pt))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&cmcf->phases[NGX_HTTP_LOG_PHASE].handlers,
                       cf->pool, 1, sizeof(ngx_http_handler_pt))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_init_headers_in_hash(ngx_conf_t *cf, ngx_http_core_main_conf_t *cmcf)
{
    ngx_array_t         headers_in;
    ngx_hash_key_t     *hk;
    ngx_hash_init_t     hash;
    ngx_http_header_t  *header;

    if (ngx_array_init(&headers_in, cf->temp_pool, 32, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    for (header = ngx_http_headers_in; header->name.len; header++) {
        hk = ngx_array_push(&headers_in);
        if (hk == NULL) {
            return NGX_ERROR;
        }

        hk->key = header->name;
        hk->key_hash = ngx_hash_key_lc(header->name.data, header->name.len);
        hk->value = header;
    }

    hash.hash = &cmcf->headers_in_hash;
    hash.key = ngx_hash_key_lc;
    hash.max_size = 512;
    hash.bucket_size = ngx_align(64, ngx_cacheline_size);
    hash.name = "headers_in_hash";
    hash.pool = cf->pool;
    hash.temp_pool = NULL;
```

```
437
438     if (ngx_hash_init(&hash, headers_in.elts, headers_in.nelts) != NGX_OK) {
439         return NGX_ERROR;
440     }
441
442     return NGX_OK;
443 }
444
445
446 static ngx_int_t
447 ngx_http_init_phase_handlers(ngx_conf_t *cf, ngx_http_core_main_conf_t *cmcf)
448 {
449     ngx_int_t                   j;
450     ngx_uint_t                  i, n;
451     ngx_uint_t                  find_config_index, use_rewrite, use_access;
452     ngx_http_handler_pt        *h;
453     ngx_http_phase_handler_t   *ph;
454     ngx_http_phase_handler_pt   checker;
455
456     cmcf->phase_engine.server_rewrite_index = (ngx_uint_t) -1;
457     cmcf->phase_engine.location_rewrite_index = (ngx_uint_t) -1;
458     find_config_index = 0;
459     use_rewrite = cmcf->phases[NGX_HTTP_REWRITE_PHASE].handlers.nelts ? 1 : 0;
460     use_access = cmcf->phases[NGX_HTTP_ACCESS_PHASE].handlers.nelts ? 1 : 0;
461
462     n = use_rewrite + use_access + cmcf->try_files + 1 /* find config phase */;
463
464     for (i = 0; i < NGX_HTTP_LOG_PHASE; i++) {
465         n += cmcf->phases[i].handlers.nelts;
466     }
467
468     ph = ngx_pcalloc(cf->pool,
469                      n * sizeof(ngx_http_phase_handler_t) + sizeof(void *));
470     if (ph == NULL) {
471         return NGX_ERROR;
472     }
473
474     cmcf->phase_engine.handlers = ph;
475     n = 0;
476
477     for (i = 0; i < NGX_HTTP_LOG_PHASE; i++) {
478         h = cmcf->phases[i].handlers.elts;
479
480         switch (i) {
481
482         case NGX_HTTP_SERVER_REWRITE_PHASE:
483             if (cmcf->phase_engine.server_rewrite_index == (ngx_uint_t) -1) {
484                 cmcf->phase_engine.server_rewrite_index = n;
485             }
486             checker = ngx_http_core_rewrite_phase;
487
488             break;
489
490         case NGX_HTTP_FIND_CONFIG_PHASE:
491             find_config_index = n;
492
493             ph->checker = ngx_http_core_find_config_phase;
494             n++;
495             ph++;
496
497             continue;
498
499         case NGX_HTTP_REWRITE_PHASE:
500             if (cmcf->phase_engine.location_rewrite_index == (ngx_uint_t) -1) {
501                 cmcf->phase_engine.location_rewrite_index = n;
502             }
503             checker = ngx_http_core_rewrite_phase;
504
505             break;
506
507         case NGX_HTTP_POST_REWRITE_PHASE:
508             if (use_rewrite) {
509                 ph->checker = ngx_http_core_post_rewrite_phase;
510                 ph->next = find_config_index;
511                 n++;
512                 ph++;
```

```
513             }
514
515             continue;
516
517         case NGX_HTTP_ACCESS_PHASE:
518             checker = ngx_http_core_access_phase;
519             n++;
520             break;
521
522         case NGX_HTTP_POST_ACCESS_PHASE:
523             if (use_access) {
524                 ph->checker = ngx_http_core_post_access_phase;
525                 ph->next = n;
526                 ph++;
527             }
528
529             continue;
530
531         case NGX_HTTP_TRY_FILES_PHASE:
532             if (cmcf->try_files) {
533                 ph->checker = ngx_http_core_try_files_phase;
534                 n++;
535                 ph++;
536             }
537
538             continue;
539
540         case NGX_HTTP_CONTENT_PHASE:
541             checker = ngx_http_core_content_phase;
542             break;
543
544         default:
545             checker = ngx_http_core_generic_phase;
546         }
547
548         n += cmcf->phases[i].handlers.nelts;
549
550         for (j = cmcf->phases[i].handlers.nelts - 1; j >=0; j--) {
551             ph->checker = checker;
552             ph->handler = h[j];
553             ph->next = n;
554             ph++;
555         }
556     }
557
558     return NGX_OK;
559 }
560
561
562 static char *
563 ngx_http_merge_servers(ngx_conf_t *cf, ngx_http_core_main_conf_t *cmcf,
564     ngx_http_module_t *module, ngx_uint_t ctx_index)
565 {
566     char                        *rv;
567     ngx_uint_t                   s;
568     ngx_http_conf_ctx_t         *ctx, saved;
569     ngx_http_core_loc_conf_t    *clcf;
570     ngx_http_core_srv_conf_t   **cscfp;
571
572     cscfp = cmcf->servers.elts;
573     ctx = (ngx_http_conf_ctx_t *) cf->ctx;
574     saved = *ctx;
575     rv = NGX_CONF_OK;
576
577     for (s = 0; s < cmcf->servers.nelts; s++) {
578
579         /* merge the server{}s' srv_conf's */
580
581         ctx->srv_conf = cscfp[s]->ctx->srv_conf;
582
583         if (module->merge_srv_conf) {
584             rv = module->merge_srv_conf(cf, saved.srv_conf[ctx_index],
585                                         cscfp[s]->ctx->srv_conf[ctx_index]);
586             if (rv != NGX_CONF_OK) {
587                 goto failed;
588             }
```

```
        }

        if (module->merge_loc_conf) {

            /* merge the server{}'s loc_conf */

            ctx->loc_conf = cscfp[s]->ctx->loc_conf;

            rv = module->merge_loc_conf(cf, saved.loc_conf[ctx_index],
                                        cscfp[s]->ctx->loc_conf[ctx_index]);
            if (rv != NGX_CONF_OK) {
                goto failed;
            }

            /* merge the locations{}' loc_conf's */

            clcf = cscfp[s]->ctx->loc_conf[ngx_http_core_module.ctx_index];

            rv = ngx_http_merge_locations(cf, clcf->locations,
                                          cscfp[s]->ctx->loc_conf,
                                          module, ctx_index);
            if (rv != NGX_CONF_OK) {
                goto failed;
            }
        }
    }

failed:

    *ctx = saved;

    return rv;
}


static char *
ngx_http_merge_locations(ngx_conf_t *cf, ngx_queue_t *locations,
    void **loc_conf, ngx_http_module_t *module, ngx_uint_t ctx_index)
{
    char                       *rv;
    ngx_queue_t                *q;
    ngx_http_conf_ctx_t        *ctx, saved;
    ngx_http_core_loc_conf_t   *clcf;
    ngx_http_location_queue_t  *lq;

    if (locations == NULL) {
        return NGX_CONF_OK;
    }

    ctx = (ngx_http_conf_ctx_t *) cf->ctx;
    saved = *ctx;

    for (q = ngx_queue_head(locations);
         q != ngx_queue_sentinel(locations);
         q = ngx_queue_next(q))
    {
        lq = (ngx_http_location_queue_t *) q;

        clcf = lq->exact ? lq->exact : lq->inclusive;
        ctx->loc_conf = clcf->loc_conf;

        rv = module->merge_loc_conf(cf, loc_conf[ctx_index],
                                    clcf->loc_conf[ctx_index]);
        if (rv != NGX_CONF_OK) {
            return rv;
        }

        rv = ngx_http_merge_locations(cf, clcf->locations, clcf->loc_conf,
                                      module, ctx_index);
        if (rv != NGX_CONF_OK) {
            return rv;
        }
    }

    *ctx = saved;

```

```
    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_init_locations(ngx_conf_t *cf, ngx_http_core_srv_conf_t *cscf,
    ngx_http_core_loc_conf_t *pclcf)
{
    ngx_uint_t                  n;
    ngx_queue_t                *q, *locations, *named, tail;
    ngx_http_core_loc_conf_t    *clcf;
    ngx_http_location_queue_t   *lq;
    ngx_http_core_loc_conf_t   **clcfp;
#if (NGX_PCRE)
    ngx_uint_t                  r;
    ngx_queue_t                *regex;
#endif

    locations = pclcf->locations;

    if (locations == NULL) {
        return NGX_OK;
    }

    ngx_queue_sort(locations, ngx_http_cmp_locations);

    named = NULL;
    n = 0;
#if (NGX_PCRE)
    regex = NULL;
    r = 0;
#endif

    for (q = ngx_queue_head(locations);
         q != ngx_queue_sentinel(locations);
         q = ngx_queue_next(q))
    {
        lq = (ngx_http_location_queue_t *) q;

        clcf = lq->exact ? lq->exact : lq->inclusive;

        if (ngx_http_init_locations(cf, NULL, clcf) != NGX_OK) {
            return NGX_ERROR;
        }

#if (NGX_PCRE)

        if (clcf->regex) {
            r++;

            if (regex == NULL) {
                regex = q;
            }

            continue;
        }

#endif

        if (clcf->named) {
            n++;

            if (named == NULL) {
                named = q;
            }

            continue;
        }

        if (clcf->noname) {
            break;
        }
    }

    if (q != ngx_queue_sentinel(locations)) {
        ngx_queue_split(locations, q, &tail);
```

```
    }

    if (named) {
        clcfp = ngx_palloc(cf->pool,
                           (n + 1) * sizeof(ngx_http_core_loc_conf_t *));
        if (clcfp == NULL) {
            return NGX_ERROR;
        }

        cscf->named_locations = clcfp;

        for (q = named;
             q != ngx_queue_sentinel(locations);
             q = ngx_queue_next(q))
        {
            lq = (ngx_http_location_queue_t *) q;

            *(clcfp++) = lq->exact;
        }

        *clcfp = NULL;

        ngx_queue_split(locations, named, &tail);
    }

#if (NGX_PCRE)

    if (regex) {

        clcfp = ngx_palloc(cf->pool,
                           (r + 1) * sizeof(ngx_http_core_loc_conf_t *));
        if (clcfp == NULL) {
            return NGX_ERROR;
        }

        pclcf->regex_locations = clcfp;

        for (q = regex;
             q != ngx_queue_sentinel(locations);
             q = ngx_queue_next(q))
        {
            lq = (ngx_http_location_queue_t *) q;

            *(clcfp++) = lq->exact;
        }

        *clcfp = NULL;

        ngx_queue_split(locations, regex, &tail);
    }

#endif

    return NGX_OK;
}


static ngx_int_t
ngx_http_init_static_location_trees(ngx_conf_t *cf,
    ngx_http_core_loc_conf_t *pclcf)
{
    ngx_queue_t                *q, *locations;
    ngx_http_core_loc_conf_t   *clcf;
    ngx_http_location_queue_t  *lq;

    locations = pclcf->locations;

    if (locations == NULL) {
        return NGX_OK;
    }

    if (ngx_queue_empty(locations)) {
        return NGX_OK;
    }

    for (q = ngx_queue_head(locations);
```

```
817          q != ngx_queue_sentinel(locations);
818          q = ngx_queue_next(q))
819     {
820         lq = (ngx_http_location_queue_t *) q;
821
822         clcf = lq->exact ? lq->exact : lq->inclusive;
823
824         if (ngx_http_init_static_location_trees(cf, clcf) != NGX_OK) {
825             return NGX_ERROR;
826         }
827     }
828
829     if (ngx_http_join_exact_locations(cf, locations) != NGX_OK) {
830         return NGX_ERROR;
831     }
832
833     ngx_http_create_locations_list(locations, ngx_queue_head(locations));
834
835     pclcf->static_locations = ngx_http_create_locations_tree(cf, locations, 0);
836     if (pclcf->static_locations == NULL) {
837         return NGX_ERROR;
838     }
839
840     return NGX_OK;
841 }
842
843
844 ngx_int_t
845 ngx_http_add_location(ngx_conf_t *cf, ngx_queue_t **locations,
846     ngx_http_core_loc_conf_t *clcf)
847 {
848     ngx_http_location_queue_t  *lq;
849
850     if (*locations == NULL) {
851         *locations = ngx_palloc(cf->temp_pool,
852                                 sizeof(ngx_http_location_queue_t));
853         if (*locations == NULL) {
854             return NGX_ERROR;
855         }
856
857         ngx_queue_init(*locations);
858     }
859
860     lq = ngx_palloc(cf->temp_pool, sizeof(ngx_http_location_queue_t));
861     if (lq == NULL) {
862         return NGX_ERROR;
863     }
864
865     if (clcf->exact_match
866 #if (NGX_PCRE)
867         || clcf->regex
868 #endif
869         || clcf->named || clcf->noname)
870     {
871         lq->exact = clcf;
872         lq->inclusive = NULL;
873
874     } else {
875         lq->exact = NULL;
876         lq->inclusive = clcf;
877     }
878
879     lq->name = &clcf->name;
880     lq->file_name = cf->conf_file->file.name.data;
881     lq->line = cf->conf_file->line;
882
883     ngx_queue_init(&lq->list);
884
885     ngx_queue_insert_tail(*locations, &lq->queue);
886
887     return NGX_OK;
888 }
889
890
891 static ngx_int_t
892 ngx_http_cmp_locations(const ngx_queue_t *one, const ngx_queue_t *two)
```

```
{
    ngx_int_t                   rc;
    ngx_http_core_loc_conf_t   *first, *second;
    ngx_http_location_queue_t  *lq1, *lq2;

    lq1 = (ngx_http_location_queue_t *) one;
    lq2 = (ngx_http_location_queue_t *) two;

    first = lq1->exact ? lq1->exact : lq1->inclusive;
    second = lq2->exact ? lq2->exact : lq2->inclusive;

    if (first->noname && !second->noname) {
        /* shift no named locations to the end */
        return 1;
    }

    if (!first->noname && second->noname) {
        /* shift no named locations to the end */
        return -1;
    }

    if (first->noname || second->noname) {
        /* do not sort no named locations */
        return 0;
    }

    if (first->named && !second->named) {
        /* shift named locations to the end */
        return 1;
    }

    if (!first->named && second->named) {
        /* shift named locations to the end */
        return -1;
    }

    if (first->named && second->named) {
        return ngx_strcmp(first->name.data, second->name.data);
    }

#if (NGX_PCRE)

    if (first->regex && !second->regex) {
        /* shift the regex matches to the end */
        return 1;
    }

    if (!first->regex && second->regex) {
        /* shift the regex matches to the end */
        return -1;
    }

    if (first->regex || second->regex) {
        /* do not sort the regex matches */
        return 0;
    }

#endif

    rc = ngx_filename_cmp(first->name.data, second->name.data,
                          ngx_min(first->name.len, second->name.len) + 1);

    if (rc == 0 && !first->exact_match && second->exact_match) {
        /* an exact match must be before the same inclusive one */
        return 1;
    }

    return rc;
}


static ngx_int_t
ngx_http_join_exact_locations(ngx_conf_t *cf, ngx_queue_t *locations)
{
    ngx_queue_t                *q, *x;
    ngx_http_location_queue_t  *lq, *lx;
```

```
969
970     q = ngx_queue_head(locations);
971
972     while (q != ngx_queue_last(locations)) {
973
974         x = ngx_queue_next(q);
975
976         lq = (ngx_http_location_queue_t *) q;
977         lx = (ngx_http_location_queue_t *) x;
978
979         if (lq->name->len == lx->name->len
980             && ngx_filename_cmp(lq->name->data, lx->name->data, lx->name->len)
981                == 0)
982         {
983             if ((lq->exact && lx->exact) || (lq->inclusive && lx->inclusive)) {
984                 ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
985                               "duplicate location \"%V\" in %s:%ui",
986                               lx->name, lx->file_name, lx->line);
987
988                 return NGX_ERROR;
989             }
990
991             lq->inclusive = lx->inclusive;
992
993             ngx_queue_remove(x);
994
995             continue;
996         }
997
998         q = ngx_queue_next(q);
999     }
1000
1001     return NGX_OK;
1002 }
1003
1004
1005 static void
1006 ngx_http_create_locations_list(ngx_queue_t *locations, ngx_queue_t *q)
1007 {
1008     u_char                     *name;
1009     size_t                      len;
1010     ngx_queue_t                *x, tail;
1011     ngx_http_location_queue_t  *lq, *lx;
1012
1013     if (q == ngx_queue_last(locations)) {
1014         return;
1015     }
1016
1017     lq = (ngx_http_location_queue_t *) q;
1018
1019     if (lq->inclusive == NULL) {
1020         ngx_http_create_locations_list(locations, ngx_queue_next(q));
1021         return;
1022     }
1023
1024     len = lq->name->len;
1025     name = lq->name->data;
1026
1027     for (x = ngx_queue_next(q);
1028          x != ngx_queue_sentinel(locations);
1029          x = ngx_queue_next(x))
1030     {
1031         lx = (ngx_http_location_queue_t *) x;
1032
1033         if (len > lx->name->len
1034             || ngx_filename_cmp(name, lx->name->data, len) != 0)
1035         {
1036             break;
1037         }
1038     }
1039
1040     q = ngx_queue_next(q);
1041
1042     if (q == x) {
1043         ngx_http_create_locations_list(locations, x);
1044         return;
```

```
    }

    ngx_queue_split(locations, q, &tail);
    ngx_queue_add(&lq->list, &tail);

    if (x == ngx_queue_sentinel(locations)) {
        ngx_http_create_locations_list(&lq->list, ngx_queue_head(&lq->list));
        return;
    }

    ngx_queue_split(&lq->list, x, &tail);
    ngx_queue_add(locations, &tail);

    ngx_http_create_locations_list(&lq->list, ngx_queue_head(&lq->list));

    ngx_http_create_locations_list(locations, x);
}


/*
* to keep cache locality for left leaf nodes, allocate nodes in following
* order: node, left subtree, right subtree, inclusive subtree
*/

static ngx_http_location_tree_node_t *
ngx_http_create_locations_tree(ngx_conf_t *cf, ngx_queue_t *locations,
    size_t prefix)
{
    size_t                          len;
    ngx_queue_t                    *q, tail;
    ngx_http_location_queue_t      *lq;
    ngx_http_location_tree_node_t  *node;

    q = ngx_queue_middle(locations);

    lq = (ngx_http_location_queue_t *) q;
    len = lq->name->len - prefix;

    node = ngx_palloc(cf->pool,
                      offsetof(ngx_http_location_tree_node_t, name) + len);
    if (node == NULL) {
        return NULL;
    }

    node->left = NULL;
    node->right = NULL;
    node->tree = NULL;
    node->exact = lq->exact;
    node->inclusive = lq->inclusive;

    node->auto_redirect = (u_char) ((lq->exact && lq->exact->auto_redirect)
                           || (lq->inclusive && lq->inclusive->auto_redirect));

    node->len = (u_char) len;
    ngx_memcpy(node->name, &lq->name->data[prefix], len);

    ngx_queue_split(locations, q, &tail);

    if (ngx_queue_empty(locations)) {
        /*
         * ngx_queue_split() insures that if left part is empty,
         * then right one is empty too
         */
        goto inclusive;
    }

    node->left = ngx_http_create_locations_tree(cf, locations, prefix);
    if (node->left == NULL) {
        return NULL;
    }

    ngx_queue_remove(q);

    if (ngx_queue_empty(&tail)) {
        goto inclusive;
    }
```

```
1121
1122        node->right = ngx_http_create_locations_tree(cf, &tail, prefix);
1123        if (node->right == NULL) {
1124            return NULL;
1125        }
1126
1127    inclusive:
1128
1129        if (ngx_queue_empty(&lq->list)) {
1130            return node;
1131        }
1132
1133        node->tree = ngx_http_create_locations_tree(cf, &lq->list, prefix + len);
1134        if (node->tree == NULL) {
1135            return NULL;
1136        }
1137
1138        return node;
1139    }
1140
1141
1142    ngx_int_t
1143    ngx_http_add_listen(ngx_conf_t *cf, ngx_http_core_srv_conf_t *cscf,
1144        ngx_http_listen_opt_t *lsopt)
1145    {
1146        in_port_t                   p;
1147        ngx_uint_t                  i;
1148        struct sockaddr            *sa;
1149        struct sockaddr_in         *sin;
1150        ngx_http_conf_port_t       *port;
1151        ngx_http_core_main_conf_t  *cmcf;
1152    #if (NGX_HAVE_INET6)
1153        struct sockaddr_in6        *sin6;
1154    #endif
1155
1156        cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
1157
1158        if (cmcf->ports == NULL) {
1159            cmcf->ports = ngx_array_create(cf->temp_pool, 2,
1160                                           sizeof(ngx_http_conf_port_t));
1161            if (cmcf->ports == NULL) {
1162                return NGX_ERROR;
1163            }
1164        }
1165
1166        sa = &lsopt->u.sockaddr;
1167
1168        switch (sa->sa_family) {
1169
1170    #if (NGX_HAVE_INET6)
1171        case AF_INET6:
1172            sin6 = &lsopt->u.sockaddr_in6;
1173            p = sin6->sin6_port;
1174            break;
1175    #endif
1176
1177    #if (NGX_HAVE_UNIX_DOMAIN)
1178        case AF_UNIX:
1179            p = 0;
1180            break;
1181    #endif
1182
1183        default: /* AF_INET */
1184            sin = &lsopt->u.sockaddr_in;
1185            p = sin->sin_port;
1186            break;
1187        }
1188
1189        port = cmcf->ports->elts;
1190        for (i = 0; i < cmcf->ports->nelts; i++) {
1191
1192            if (p != port[i].port || sa->sa_family != port[i].family) {
1193                continue;
1194            }
1195
1196            /* a port is already in the port list */
```

```
1197
1198            return ngx_http_add_addresses(cf, cscf, &port[i], lsopt);
1199        }
1200
1201        /* add a port to the port list */
1202
1203        port = ngx_array_push(cmcf->ports);
1204        if (port == NULL) {
1205            return NGX_ERROR;
1206        }
1207
1208        port->family = sa->sa_family;
1209        port->port = p;
1210        port->addrs.elts = NULL;
1211
1212        return ngx_http_add_address(cf, cscf, port, lsopt);
1213    }
1214
1215
1216    static ngx_int_t
1217    ngx_http_add_addresses(ngx_conf_t *cf, ngx_http_core_srv_conf_t *cscf,
1218        ngx_http_conf_port_t *port, ngx_http_listen_opt_t *lsopt)
1219    {
1220        u_char                *p;
1221        size_t                 len, off;
1222        ngx_uint_t             i, default_server;
1223        struct sockaddr       *sa;
1224        ngx_http_conf_addr_t  *addr;
1225    #if (NGX_HAVE_UNIX_DOMAIN)
1226        struct sockaddr_un    *saun;
1227    #endif
1228    #if (NGX_HTTP_SSL)
1229        ngx_uint_t             ssl;
1230    #endif
1231    #if (NGX_HTTP_SPDY)
1232        ngx_uint_t             spdy;
1233    #endif
1234
1235        /*
1236         * we cannot compare whole sockaddr struct's as kernel
1237         * may fill some fields in inherited sockaddr struct's
1238         */
1239
1240        sa = &lsopt->u.sockaddr;
1241
1242        switch (sa->sa_family) {
1243
1244    #if (NGX_HAVE_INET6)
1245        case AF_INET6:
1246            off = offsetof(struct sockaddr_in6, sin6_addr);
1247            len = 16;
1248            break;
1249    #endif
1250
1251    #if (NGX_HAVE_UNIX_DOMAIN)
1252        case AF_UNIX:
1253            off = offsetof(struct sockaddr_un, sun_path);
1254            len = sizeof(saun->sun_path);
1255            break;
1256    #endif
1257
1258        default: /* AF_INET */
1259            off = offsetof(struct sockaddr_in, sin_addr);
1260            len = 4;
1261            break;
1262        }
1263
1264        p = lsopt->u.sockaddr_data + off;
1265
1266        addr = port->addrs.elts;
1267
1268        for (i = 0; i < port->addrs.nelts; i++) {
1269
1270            if (ngx_memcmp(p, addr[i].opt.u.sockaddr_data + off, len) != 0) {
1271                continue;
1272            }
```

```
1273
1274        /* the address is already in the address list */
1275
1276        if (ngx_http_add_server(cf, cscf, &addr[i]) != NGX_OK) {
1277            return NGX_ERROR;
1278        }
1279
1280        /* preserve default_server bit during listen options overwriting */
1281        default_server = addr[i].opt.default_server;
1282
1283 #if (NGX_HTTP_SSL)
1284        ssl = lsopt->ssl || addr[i].opt.ssl;
1285 #endif
1286 #if (NGX_HTTP_SPDY)
1287        spdy = lsopt->spdy || addr[i].opt.spdy;
1288 #endif
1289
1290        if (lsopt->set) {
1291
1292            if (addr[i].opt.set) {
1293                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1294                        "duplicate listen options for %s", addr[i].opt.addr);
1295                return NGX_ERROR;
1296            }
1297
1298            addr[i].opt = *lsopt;
1299        }
1300
1301        /* check the duplicate "default" server for this address:port */
1302
1303        if (lsopt->default_server) {
1304
1305            if (default_server) {
1306                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1307                        "a duplicate default server for %s", addr[i].opt.addr);
1308                return NGX_ERROR;
1309            }
1310
1311            default_server = 1;
1312            addr[i].default_server = cscf;
1313        }
1314
1315        addr[i].opt.default_server = default_server;
1316 #if (NGX_HTTP_SSL)
1317        addr[i].opt.ssl = ssl;
1318 #endif
1319 #if (NGX_HTTP_SPDY)
1320        addr[i].opt.spdy = spdy;
1321 #endif
1322
1323        return NGX_OK;
1324    }
1325
1326    /* add the address to the addresses list that bound to this port */
1327
1328    return ngx_http_add_address(cf, cscf, port, lsopt);
1329 }
1330
1331
1332 /*
1333  * add the server address, the server names and the server core module
1334  * configurations to the port list
1335  */
1336
1337 static ngx_int_t
1338 ngx_http_add_address(ngx_conf_t *cf, ngx_http_core_srv_conf_t *cscf,
1339     ngx_http_conf_port_t *port, ngx_http_listen_opt_t *lsopt)
1340 {
1341     ngx_http_conf_addr_t  *addr;
1342
1343     if (port->addrs.elts == NULL) {
1344         if (ngx_array_init(&port->addrs, cf->temp_pool, 4,
1345                            sizeof(ngx_http_conf_addr_t))
1346             != NGX_OK)
1347         {
1348             return NGX_ERROR;
```

```
        }
    }

#if (NGX_HTTP_SPDY && NGX_HTTP_SSL                                           \
     && !defined TLSEXT_TYPE_application_layer_protocol_negotiation          \
     && !defined TLSEXT_TYPE_next_proto_neg)
    if (lsopt->spdy && lsopt->ssl) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "nginx was built without OpenSSL ALPN or NPN "
                           "support, SPDY is not enabled for %s", lsopt->addr);
    }
#endif

    addr = ngx_array_push(&port->addrs);
    if (addr == NULL) {
        return NGX_ERROR;
    }

    addr->opt = *lsopt;
    addr->hash.buckets = NULL;
    addr->hash.size = 0;
    addr->wc_head = NULL;
    addr->wc_tail = NULL;
#if (NGX_PCRE)
    addr->nregex = 0;
    addr->regex = NULL;
#endif
    addr->default_server = cscf;
    addr->servers.elts = NULL;

    return ngx_http_add_server(cf, cscf, addr);
}


/* add the server core module configuration to the address:port */

static ngx_int_t
ngx_http_add_server(ngx_conf_t *cf, ngx_http_core_srv_conf_t *cscf,
    ngx_http_conf_addr_t *addr)
{
    ngx_uint_t                  i;
    ngx_http_core_srv_conf_t  **server;

    if (addr->servers.elts == NULL) {
        if (ngx_array_init(&addr->servers, cf->temp_pool, 4,
                           sizeof(ngx_http_core_srv_conf_t *))
            != NGX_OK)
        {
            return NGX_ERROR;
        }

    } else {
        server = addr->servers.elts;
        for (i = 0; i < addr->servers.nelts; i++) {
            if (server[i] == cscf) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "a duplicate listen %s", addr->opt.addr);
                return NGX_ERROR;
            }
        }
    }

    server = ngx_array_push(&addr->servers);
    if (server == NULL) {
        return NGX_ERROR;
    }

    *server = cscf;

    return NGX_OK;
}


static ngx_int_t
ngx_http_optimize_servers(ngx_conf_t *cf, ngx_http_core_main_conf_t *cmcf,
    ngx_array_t *ports)
```

```
{
    ngx_uint_t             p, a;
    ngx_http_conf_port_t  *port;
    ngx_http_conf_addr_t  *addr;

    if (ports == NULL) {
        return NGX_OK;
    }

    port = ports->elts;
    for (p = 0; p < ports->nelts; p++) {

        ngx_sort(port[p].addrs.elts, (size_t) port[p].addrs.nelts,
                 sizeof(ngx_http_conf_addr_t), ngx_http_cmp_conf_addrs);

        /*
         * check whether all name-based servers have the same
         * configuration as a default server for given address:port
         */

        addr = port[p].addrs.elts;
        for (a = 0; a < port[p].addrs.nelts; a++) {

            if (addr[a].servers.nelts > 1
#if (NGX_PCRE)
                || addr[a].default_server->captures
#endif
               )
            {
                if (ngx_http_server_names(cf, cmcf, &addr[a]) != NGX_OK) {
                    return NGX_ERROR;
                }
            }
        }

        if (ngx_http_init_listening(cf, &port[p]) != NGX_OK) {
            return NGX_ERROR;
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_server_names(ngx_conf_t *cf, ngx_http_core_main_conf_t *cmcf,
    ngx_http_conf_addr_t *addr)
{
    ngx_int_t                   rc;
    ngx_uint_t                  n, s;
    ngx_hash_init_t             hash;
    ngx_hash_keys_arrays_t      ha;
    ngx_http_server_name_t     *name;
    ngx_http_core_srv_conf_t  **cscfp;
#if (NGX_PCRE)
    ngx_uint_t                  regex, i;

    regex = 0;
#endif

    ngx_memzero(&ha, sizeof(ngx_hash_keys_arrays_t));

    ha.temp_pool = ngx_create_pool(NGX_DEFAULT_POOL_SIZE, cf->log);
    if (ha.temp_pool == NULL) {
        return NGX_ERROR;
    }

    ha.pool = cf->pool;

    if (ngx_hash_keys_array_init(&ha, NGX_HASH_LARGE) != NGX_OK) {
        goto failed;
    }

    cscfp = addr->servers.elts;

    for (s = 0; s < addr->servers.nelts; s++) {
```

```
1501
1502            name = cscfp[s]->server_names.elts;
1503
1504            for (n = 0; n < cscfp[s]->server_names.nelts; n++) {
1505
1506        #if (NGX_PCRE)
1507                if (name[n].regex) {
1508                    regex++;
1509                    continue;
1510                }
1511        #endif
1512
1513                rc = ngx_hash_add_key(&ha, &name[n].name, name[n].server,
1514                                      NGX_HASH_WILDCARD_KEY);
1515
1516                if (rc == NGX_ERROR) {
1517                    return NGX_ERROR;
1518                }
1519
1520                if (rc == NGX_DECLINED) {
1521                    ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
1522                                  "invalid server name or wildcard \"%V\" on %s",
1523                                  &name[n].name, addr->opt.addr);
1524                    return NGX_ERROR;
1525                }
1526
1527                if (rc == NGX_BUSY) {
1528                    ngx_log_error(NGX_LOG_WARN, cf->log, 0,
1529                                  "conflicting server name \"%V\" on %s, ignored",
1530                                  &name[n].name, addr->opt.addr);
1531                }
1532            }
1533        }
1534
1535        hash.key = ngx_hash_key_lc;
1536        hash.max_size = cmcf->server_names_hash_max_size;
1537        hash.bucket_size = cmcf->server_names_hash_bucket_size;
1538        hash.name = "server_names_hash";
1539        hash.pool = cf->pool;
1540
1541        if (ha.keys.nelts) {
1542            hash.hash = &addr->hash;
1543            hash.temp_pool = NULL;
1544
1545            if (ngx_hash_init(&hash, ha.keys.elts, ha.keys.nelts) != NGX_OK) {
1546                goto failed;
1547            }
1548        }
1549
1550        if (ha.dns_wc_head.nelts) {
1551
1552            ngx_qsort(ha.dns_wc_head.elts, (size_t) ha.dns_wc_head.nelts,
1553                      sizeof(ngx_hash_key_t), ngx_http_cmp_dns_wildcards);
1554
1555            hash.hash = NULL;
1556            hash.temp_pool = ha.temp_pool;
1557
1558            if (ngx_hash_wildcard_init(&hash, ha.dns_wc_head.elts,
1559                                       ha.dns_wc_head.nelts)
1560                != NGX_OK)
1561            {
1562                goto failed;
1563            }
1564
1565            addr->wc_head = (ngx_hash_wildcard_t *) hash.hash;
1566        }
1567
1568        if (ha.dns_wc_tail.nelts) {
1569
1570            ngx_qsort(ha.dns_wc_tail.elts, (size_t) ha.dns_wc_tail.nelts,
1571                      sizeof(ngx_hash_key_t), ngx_http_cmp_dns_wildcards);
1572
1573            hash.hash = NULL;
1574            hash.temp_pool = ha.temp_pool;
1575
1576            if (ngx_hash_wildcard_init(&hash, ha.dns_wc_tail.elts,
```

```
                                    ha.dns_wc_tail.nelts)
            != NGX_OK)
        {
            goto failed;
        }

        addr->wc_tail = (ngx_hash_wildcard_t *) hash.hash;
    }

    ngx_destroy_pool(ha.temp_pool);

#if (NGX_PCRE)

    if (regex == 0) {
        return NGX_OK;
    }

    addr->nregex = regex;
    addr->regex = ngx_palloc(cf->pool, regex * sizeof(ngx_http_server_name_t));
    if (addr->regex == NULL) {
        return NGX_ERROR;
    }

    i = 0;

    for (s = 0; s < addr->servers.nelts; s++) {

        name = cscfp[s]->server_names.elts;

        for (n = 0; n < cscfp[s]->server_names.nelts; n++) {
            if (name[n].regex) {
                addr->regex[i++] = name[n];
            }
        }
    }

#endif

    return NGX_OK;

failed:

    ngx_destroy_pool(ha.temp_pool);

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_cmp_conf_addrs(const void *one, const void *two)
{
    ngx_http_conf_addr_t  *first, *second;

    first = (ngx_http_conf_addr_t *) one;
    second = (ngx_http_conf_addr_t *) two;

    if (first->opt.wildcard) {
        /* a wildcard address must be the last resort, shift it to the end */
        return 1;
    }

    if (second->opt.wildcard) {
        /* a wildcard address must be the last resort, shift it to the end */
        return -1;
    }

    if (first->opt.bind && !second->opt.bind) {
        /* shift explicit bind()ed addresses to the start */
        return -1;
    }

    if (!first->opt.bind && second->opt.bind) {
        /* shift explicit bind()ed addresses to the start */
        return 1;
    }

```

```
1653      /* do not sort by default */
1654
1655      return 0;
1656  }
1657
1658
1659  static int ngx_libc_cdecl
1660  ngx_http_cmp_dns_wildcards(const void *one, const void *two)
1661  {
1662      ngx_hash_key_t  *first, *second;
1663
1664      first = (ngx_hash_key_t *) one;
1665      second = (ngx_hash_key_t *) two;
1666
1667      return ngx_dns_strcmp(first->key.data, second->key.data);
1668  }
1669
1670
1671  static ngx_int_t
1672  ngx_http_init_listening(ngx_conf_t *cf, ngx_http_conf_port_t *port)
1673  {
1674      ngx_uint_t                 i, last, bind_wildcard;
1675      ngx_listening_t           *ls;
1676      ngx_http_port_t           *hport;
1677      ngx_http_conf_addr_t      *addr;
1678
1679      addr = port->addrs.elts;
1680      last = port->addrs.nelts;
1681
1682      /*
1683       * If there is a binding to an "*:port" then we need to bind() to
1684       * the "*:port" only and ignore other implicit bindings.  The bindings
1685       * have been already sorted: explicit bindings are on the start, then
1686       * implicit bindings go, and wildcard binding is in the end.
1687       */
1688
1689      if (addr[last - 1].opt.wildcard) {
1690          addr[last - 1].opt.bind = 1;
1691          bind_wildcard = 1;
1692
1693      } else {
1694          bind_wildcard = 0;
1695      }
1696
1697      i = 0;
1698
1699      while (i < last) {
1700
1701          if (bind_wildcard && !addr[i].opt.bind) {
1702              i++;
1703              continue;
1704          }
1705
1706          ls = ngx_http_add_listening(cf, &addr[i]);
1707          if (ls == NULL) {
1708              return NGX_ERROR;
1709          }
1710
1711          hport = ngx_pcalloc(cf->pool, sizeof(ngx_http_port_t));
1712          if (hport == NULL) {
1713              return NGX_ERROR;
1714          }
1715
1716          ls->servers = hport;
1717
1718          if (i == last - 1) {
1719              hport->naddrs = last;
1720
1721          } else {
1722              hport->naddrs = 1;
1723              i = 0;
1724          }
1725
1726          switch (ls->sockaddr->sa_family) {
1727
1728  #if (NGX_HAVE_INET6)
```

```
        case AF_INET6:
            if (ngx_http_add_addrs6(cf, hport, addr) != NGX_OK) {
                return NGX_ERROR;
            }
            break;
#endif
        default: /* AF_INET */
            if (ngx_http_add_addrs(cf, hport, addr) != NGX_OK) {
                return NGX_ERROR;
            }
            break;
        }

        addr++;
        last--;
    }

    return NGX_OK;
}


static ngx_listening_t *
ngx_http_add_listening(ngx_conf_t *cf, ngx_http_conf_addr_t *addr)
{
    ngx_listening_t           *ls;
    ngx_http_core_loc_conf_t  *clcf;
    ngx_http_core_srv_conf_t  *cscf;

    ls = ngx_create_listening(cf, &addr->opt.u.sockaddr, addr->opt.socklen);
    if (ls == NULL) {
        return NULL;
    }

    ls->addr_ntop = 1;

    ls->handler = ngx_http_init_connection;

    cscf = addr->default_server;
    ls->pool_size = cscf->connection_pool_size;
    ls->post_accept_timeout = cscf->client_header_timeout;

    clcf = cscf->ctx->loc_conf[ngx_http_core_module.ctx_index];

    ls->logp = clcf->error_log;
    ls->log.data = &ls->addr_text;
    ls->log.handler = ngx_accept_log_error;

#if (NGX_WIN32)
    {
    ngx_iocp_conf_t  *iocpcf = NULL;

    if (ngx_get_conf(cf->cycle->conf_ctx, ngx_events_module)) {
        iocpcf = ngx_event_get_conf(cf->cycle->conf_ctx, ngx_iocp_module);
    }
    if (iocpcf && iocpcf->acceptex_read) {
        ls->post_accept_buffer_size = cscf->client_header_buffer_size;
    }
    }
#endif

    ls->backlog = addr->opt.backlog;
    ls->rcvbuf = addr->opt.rcvbuf;
    ls->sndbuf = addr->opt.sndbuf;

    ls->keepalive = addr->opt.so_keepalive;
#if (NGX_HAVE_KEEPALIVE_TUNABLE)
    ls->keepidle = addr->opt.tcp_keepidle;
    ls->keepintvl = addr->opt.tcp_keepintvl;
    ls->keepcnt = addr->opt.tcp_keepcnt;
#endif

#if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
    ls->accept_filter = addr->opt.accept_filter;
#endif

#if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
```

```
1805      ls->deferred_accept = addr->opt.deferred_accept;
1806  #endif
1807
1808  #if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
1809      ls->ipv6only = addr->opt.ipv6only;
1810  #endif
1811
1812  #if (NGX_HAVE_SETFIB)
1813      ls->setfib = addr->opt.setfib;
1814  #endif
1815
1816  #if (NGX_HAVE_TCP_FASTOPEN)
1817      ls->fastopen = addr->opt.fastopen;
1818  #endif
1819
1820      return ls;
1821  }
1822
1823
1824  static ngx_int_t
1825  ngx_http_add_addrs(ngx_conf_t *cf, ngx_http_port_t *hport,
1826      ngx_http_conf_addr_t *addr)
1827  {
1828      ngx_uint_t                 i;
1829      ngx_http_in_addr_t        *addrs;
1830      struct sockaddr_in        *sin;
1831      ngx_http_virtual_names_t  *vn;
1832
1833      hport->addrs = ngx_pcalloc(cf->pool,
1834                                 hport->naddrs * sizeof(ngx_http_in_addr_t));
1835      if (hport->addrs == NULL) {
1836          return NGX_ERROR;
1837      }
1838
1839      addrs = hport->addrs;
1840
1841      for (i = 0; i < hport->naddrs; i++) {
1842
1843          sin = &addr[i].opt.u.sockaddr_in;
1844          addrs[i].addr = sin->sin_addr.s_addr;
1845          addrs[i].conf.default_server = addr[i].default_server;
1846  #if (NGX_HTTP_SSL)
1847          addrs[i].conf.ssl = addr[i].opt.ssl;
1848  #endif
1849  #if (NGX_HTTP_SPDY)
1850          addrs[i].conf.spdy = addr[i].opt.spdy;
1851  #endif
1852          addrs[i].conf.proxy_protocol = addr[i].opt.proxy_protocol;
1853
1854          if (addr[i].hash.buckets == NULL
1855              && (addr[i].wc_head == NULL
1856                  || addr[i].wc_head->hash.buckets == NULL)
1857              && (addr[i].wc_tail == NULL
1858                  || addr[i].wc_tail->hash.buckets == NULL)
1859  #if (NGX_PCRE)
1860              && addr[i].nregex == 0
1861  #endif
1862              )
1863          {
1864              continue;
1865          }
1866
1867          vn = ngx_palloc(cf->pool, sizeof(ngx_http_virtual_names_t));
1868          if (vn == NULL) {
1869              return NGX_ERROR;
1870          }
1871
1872          addrs[i].conf.virtual_names = vn;
1873
1874          vn->names.hash = addr[i].hash;
1875          vn->names.wc_head = addr[i].wc_head;
1876          vn->names.wc_tail = addr[i].wc_tail;
1877  #if (NGX_PCRE)
1878          vn->nregex = addr[i].nregex;
1879          vn->regex = addr[i].regex;
1880  #endif
```

```
    }

    return NGX_OK;
}


#if (NGX_HAVE_INET6)

static ngx_int_t
ngx_http_add_addrs6(ngx_conf_t *cf, ngx_http_port_t *hport,
    ngx_http_conf_addr_t *addr)
{
    ngx_uint_t                 i;
    ngx_http_in6_addr_t       *addrs6;
    struct sockaddr_in6       *sin6;
    ngx_http_virtual_names_t  *vn;

    hport->addrs = ngx_pcalloc(cf->pool,
                               hport->naddrs * sizeof(ngx_http_in6_addr_t));
    if (hport->addrs == NULL) {
        return NGX_ERROR;
    }

    addrs6 = hport->addrs;

    for (i = 0; i < hport->naddrs; i++) {

        sin6 = &addr[i].opt.u.sockaddr_in6;
        addrs6[i].addr6 = sin6->sin6_addr;
        addrs6[i].conf.default_server = addr[i].default_server;
#if (NGX_HTTP_SSL)
        addrs6[i].conf.ssl = addr[i].opt.ssl;
#endif
#if (NGX_HTTP_SPDY)
        addrs6[i].conf.spdy = addr[i].opt.spdy;
#endif

        if (addr[i].hash.buckets == NULL
            && (addr[i].wc_head == NULL
                || addr[i].wc_head->hash.buckets == NULL)
            && (addr[i].wc_tail == NULL
                || addr[i].wc_tail->hash.buckets == NULL)
#if (NGX_PCRE)
            && addr[i].nregex == 0
#endif
            )
        {
            continue;
        }

        vn = ngx_palloc(cf->pool, sizeof(ngx_http_virtual_names_t));
        if (vn == NULL) {
            return NGX_ERROR;
        }

        addrs6[i].conf.virtual_names = vn;

        vn->names.hash = addr[i].hash;
        vn->names.wc_head = addr[i].wc_head;
        vn->names.wc_tail = addr[i].wc_tail;
#if (NGX_PCRE)
        vn->nregex = addr[i].nregex;
        vn->regex = addr[i].regex;
#endif
    }

    return NGX_OK;
}

#endif


char *
ngx_http_types_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;
```

```
1957
1958        ngx_array_t     **types;
1959        ngx_str_t        *value, *default_type;
1960        ngx_uint_t        i, n, hash;
1961        ngx_hash_key_t   *type;
1962
1963        types = (ngx_array_t **) (p + cmd->offset);
1964
1965        if (*types == (void *) -1) {
1966            return NGX_CONF_OK;
1967        }
1968
1969        default_type = cmd->post;
1970
1971        if (*types == NULL) {
1972            *types = ngx_array_create(cf->temp_pool, 1, sizeof(ngx_hash_key_t));
1973            if (*types == NULL) {
1974                return NGX_CONF_ERROR;
1975            }
1976
1977            if (default_type) {
1978                type = ngx_array_push(*types);
1979                if (type == NULL) {
1980                    return NGX_CONF_ERROR;
1981                }
1982
1983                type->key = *default_type;
1984                type->key_hash = ngx_hash_key(default_type->data,
1985                                              default_type->len);
1986                type->value = (void *) 4;
1987            }
1988        }
1989
1990        value = cf->args->elts;
1991
1992        for (i = 1; i < cf->args->nelts; i++) {
1993
1994            if (value[i].len == 1 && value[i].data[0] == '*') {
1995                *types = (void *) -1;
1996                return NGX_CONF_OK;
1997            }
1998
1999            hash = ngx_hash_strlow(value[i].data, value[i].data, value[i].len);
2000            value[i].data[value[i].len] = '\0';
2001
2002            type = (*types)->elts;
2003            for (n = 0; n < (*types)->nelts; n++) {
2004
2005                if (ngx_strcmp(value[i].data, type[n].key.data) == 0) {
2006                    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
2007                                       "duplicate MIME type \"%V\"", &value[i]);
2008                    goto next;
2009                }
2010            }
2011
2012            type = ngx_array_push(*types);
2013            if (type == NULL) {
2014                return NGX_CONF_ERROR;
2015            }
2016
2017            type->key = value[i];
2018            type->key_hash = hash;
2019            type->value = (void *) 4;
2020
2021        next:
2022
2023            continue;
2024        }
2025
2026        return NGX_CONF_OK;
2027    }
2028
2029
2030    char *
2031    ngx_http_merge_types(ngx_conf_t *cf, ngx_array_t **keys, ngx_hash_t *types_hash,
2032        ngx_array_t **prev_keys, ngx_hash_t *prev_types_hash,
```

```
2033      ngx_str_t *default_types)
2034  {
2035      ngx_hash_init_t  hash;
2036
2037      if (*keys) {
2038
2039          if (*keys == (void *) -1) {
2040              return NGX_CONF_OK;
2041          }
2042
2043          hash.hash = types_hash;
2044          hash.key = NULL;
2045          hash.max_size = 2048;
2046          hash.bucket_size = 64;
2047          hash.name = "test_types_hash";
2048          hash.pool = cf->pool;
2049          hash.temp_pool = NULL;
2050
2051          if (ngx_hash_init(&hash, (*keys)->elts, (*keys)->nelts) != NGX_OK) {
2052              return NGX_CONF_ERROR;
2053          }
2054
2055          return NGX_CONF_OK;
2056      }
2057
2058      if (prev_types_hash->buckets == NULL) {
2059
2060          if (*prev_keys == NULL) {
2061
2062              if (ngx_http_set_default_types(cf, prev_keys, default_types)
2063                  != NGX_OK)
2064              {
2065                  return NGX_CONF_ERROR;
2066              }
2067
2068          } else if (*prev_keys == (void *) -1) {
2069              *keys = *prev_keys;
2070              return NGX_CONF_OK;
2071          }
2072
2073          hash.hash = prev_types_hash;
2074          hash.key = NULL;
2075          hash.max_size = 2048;
2076          hash.bucket_size = 64;
2077          hash.name = "test_types_hash";
2078          hash.pool = cf->pool;
2079          hash.temp_pool = NULL;
2080
2081          if (ngx_hash_init(&hash, (*prev_keys)->elts, (*prev_keys)->nelts)
2082              != NGX_OK)
2083          {
2084              return NGX_CONF_ERROR;
2085          }
2086      }
2087
2088      *types_hash = *prev_types_hash;
2089
2090      return NGX_CONF_OK;
2091
2092  }
2093
2094
2095  ngx_int_t
2096  ngx_http_set_default_types(ngx_conf_t *cf, ngx_array_t **types,
2097      ngx_str_t *default_type)
2098  {
2099      ngx_hash_key_t  *type;
2100
2101      *types = ngx_array_create(cf->temp_pool, 1, sizeof(ngx_hash_key_t));
2102      if (*types == NULL) {
2103          return NGX_ERROR;
2104      }
2105
2106      while (default_type->len) {
2107
2108          type = ngx_array_push(*types);
```

```
2109         if (type == NULL) {
2110             return NGX_ERROR;
2111         }
2112 
2113         type->key = *default_type;
2114         type->key_hash = ngx_hash_key(default_type->data,
2115                                       default_type->len);
2116         type->value = (void *) 4;
2117 
2118         default_type++;
2119     }
2120 
2121     return NGX_OK;
2122 }
```

One Level Up Top Level

src/core/ngx_core.h - nginx-1.7.10

Data types defined

- ngx_chain_t
- ngx_command_t
- ngx_conf_t
- ngx_connection_handler_pt
- ngx_connection_t
- ngx_cycle_t
- ngx_event_aio_t
- ngx_event_handler_pt
- ngx_event_t
- ngx_file_t
- ngx_log_t
- ngx_module_t
- ngx_open_file_t
- ngx_pool_t

Macros defined

- CR
- CRLF
- LF
- NGX_ABORT
- NGX_AGAIN
- NGX_BUSY
- NGX_DECLINED
- NGX_DISABLE_SYMLINKS_NOTOWNER
- NGX_DISABLE_SYMLINKS_OFF
- NGX_DISABLE_SYMLINKS_ON
- NGX_DONE
- NGX_ERROR
- NGX_OK
- _NGX_CORE_H_INCLUDED_

- ngx_abs
- ngx_max
- ngx_min

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_CORE_H_INCLUDED_
9  #define _NGX_CORE_H_INCLUDED_
10
11
12  typedef struct ngx_module_s      ngx_module_t;
13  typedef struct ngx_conf_s        ngx_conf_t;
14  typedef struct ngx_cycle_s       ngx_cycle_t;
15  typedef struct ngx_pool_s        ngx_pool_t;
16  typedef struct ngx_chain_s       ngx_chain_t;
17  typedef struct ngx_log_s         ngx_log_t;
18  typedef struct ngx_open_file_s   ngx_open_file_t;
19  typedef struct ngx_command_s     ngx_command_t;
20  typedef struct ngx_file_s        ngx_file_t;
21  typedef struct ngx_event_s       ngx_event_t;
22  typedef struct ngx_event_aio_s   ngx_event_aio_t;
23  typedef struct ngx_connection_s  ngx_connection_t;
24
25  typedef void (*ngx_event_handler_pt)(ngx_event_t *ev);
26  typedef void (*ngx_connection_handler_pt)(ngx_connection_t *c);
27
28
29  #define  NGX_OK          0
30  #define  NGX_ERROR      -1
31  #define  NGX_AGAIN      -2
32  #define  NGX_BUSY       -3
33  #define  NGX_DONE       -4
34  #define  NGX_DECLINED   -5
35  #define  NGX_ABORT      -6
36
37
38  #include <ngx_errno.h>
39  #include <ngx_atomic.h>
40  #include <ngx_thread.h>
41  #include <ngx_rbtree.h>
42  #include <ngx_time.h>
43  #include <ngx_socket.h>
44  #include <ngx_string.h>
45  #include <ngx_files.h>
46  #include <ngx_shmem.h>
47  #include <ngx_process.h>
48  #include <ngx_user.h>
49  #include <ngx_parse.h>
50  #include <ngx_log.h>
51  #include <ngx_alloc.h>
52  #include <ngx_palloc.h>
53  #include <ngx_buf.h>
54  #include <ngx_queue.h>
55  #include <ngx_array.h>
56  #include <ngx_list.h>
57  #include <ngx_hash.h>
58  #include <ngx_file.h>
59  #include <ngx_crc.h>
60  #include <ngx_crc32.h>
61  #include <ngx_murmurhash.h>
62  #if (NGX_PCRE)
63  #include <ngx_regex.h>
64  #endif
65  #include <ngx_radix_tree.h>
```

```
66  #include <ngx_times.h>
67  #include <ngx_shmtx.h>
68  #include <ngx_slab.h>
69  #include <ngx_inet.h>
70  #include <ngx_cycle.h>
71  #include <ngx_resolver.h>
72  #if (NGX_OPENSSL)
73  #include <ngx_event_openssl.h>
74  #endif
75  #include <ngx_process_cycle.h>
76  #include <ngx_conf_file.h>
77  #include <ngx_open_file_cache.h>
78  #include <ngx_os.h>
79  #include <ngx_connection.h>
80  #include <ngx_syslog.h>
81  #include <ngx_proxy_protocol.h>
82
83
84  #define LF     (u_char) '\n'
85  #define CR     (u_char) '\r'
86  #define CRLF   "\r\n"
87
88
89  #define ngx_abs(value)       (((value) >= 0) ? (value) : - (value))
90  #define ngx_max(val1, val2)  ((val1 < val2) ? (val2) : (val1))
91  #define ngx_min(val1, val2)  ((val1 > val2) ? (val2) : (val1))
92
93  void ngx_cpuinfo(void);
94
95  #if (NGX_HAVE_OPENAT)
96  #define NGX_DISABLE_SYMLINKS_OFF        0
97  #define NGX_DISABLE_SYMLINKS_ON         1
98  #define NGX_DISABLE_SYMLINKS_NOTOWNER   2
99  #endif
100
101  #endif /* _NGX_CORE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_conf_file.h - nginx-1.7.10

Data types defined

- ngx_command_s
- ngx_conf_bitmask_t
- ngx_conf_deprecated_t
- ngx_conf_enum_t
- ngx_conf_file_t
- ngx_conf_handler_pt
- ngx_conf_num_bounds_t
- ngx_conf_post_handler_pt
- ngx_conf_post_t
- ngx_conf_s
- ngx_core_module_t
- ngx_module_s
- ngx_open_file_s

Macros defined

- NGX_ANY_CONF
- NGX_CONF_1MORE
- NGX_CONF_2MORE
- NGX_CONF_ANY
- NGX_CONF_ARGS_NUMBER
- NGX_CONF_BITMASK_SET
- NGX_CONF_BLOCK
- NGX_CONF_BLOCK_DONE
- NGX_CONF_BLOCK_START
- NGX_CONF_ERROR
- NGX_CONF_FILE_DONE
- NGX_CONF_FLAG
- NGX_CONF_MAX_ARGS
- NGX_CONF_MODULE
- NGX_CONF_MULTI

- NGX_CONF_NOARGS
- NGX_CONF_OK
- NGX_CONF_TAKE1
- NGX_CONF_TAKE12
- NGX_CONF_TAKE123
- NGX_CONF_TAKE1234
- NGX_CONF_TAKE13
- NGX_CONF_TAKE2
- NGX_CONF_TAKE23
- NGX_CONF_TAKE3
- NGX_CONF_TAKE4
- NGX_CONF_TAKE5
- NGX_CONF_TAKE6
- NGX_CONF_TAKE7
- NGX_CONF_UNSET
- NGX_CONF_UNSET_MSEC
- NGX_CONF_UNSET_PTR
- NGX_CONF_UNSET_SIZE
- NGX_CONF_UNSET_UINT
- NGX_CORE_MODULE
- NGX_DIRECT_CONF
- NGX_MAIN_CONF
- NGX_MAX_CONF_ERRSTR
- NGX_MODULE_V1
- NGX_MODULE_V1_PADDING
- _NGX_CONF_FILE_H_INCLUDED_
- ngx_conf_init_msec_value
- ngx_conf_init_ptr_value
- ngx_conf_init_size_value
- ngx_conf_init_uint_value
- ngx_conf_init_value
- ngx_conf_merge_bitmask_value
- ngx_conf_merge_bufs_value

- ngx_conf_merge_msec_value
- ngx_conf_merge_off_value
- ngx_conf_merge_ptr_value
- ngx_conf_merge_sec_value
- ngx_conf_merge_size_value
- ngx_conf_merge_str_value
- ngx_conf_merge_uint_value
- ngx_conf_merge_value
- ngx_get_conf
- ngx_null_command

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_CONF_FILE_H_INCLUDED_
9  #define _NGX_CONF_FILE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  /*
17  *        AAAA  number of arguments
18  *      FF      command flags
19  *    TT        command type, i.e. HTTP "location" or "server" command
20  */
21
22  #define NGX_CONF_NOARGS      0x00000001
23  #define NGX_CONF_TAKE1       0x00000002
24  #define NGX_CONF_TAKE2       0x00000004
25  #define NGX_CONF_TAKE3       0x00000008
26  #define NGX_CONF_TAKE4       0x00000010
27  #define NGX_CONF_TAKE5       0x00000020
28  #define NGX_CONF_TAKE6       0x00000040
29  #define NGX_CONF_TAKE7       0x00000080
30
31  #define NGX_CONF_MAX_ARGS    8
32
33  #define NGX_CONF_TAKE12      (NGX_CONF_TAKE1|NGX_CONF_TAKE2)
34  #define NGX_CONF_TAKE13      (NGX_CONF_TAKE1|NGX_CONF_TAKE3)
35
36  #define NGX_CONF_TAKE23      (NGX_CONF_TAKE2|NGX_CONF_TAKE3)
37
38  #define NGX_CONF_TAKE123     (NGX_CONF_TAKE1|NGX_CONF_TAKE2|NGX_CONF_TAKE3)
39  #define NGX_CONF_TAKE1234    (NGX_CONF_TAKE1|NGX_CONF_TAKE2|NGX_CONF_TAKE3   \
40                                |NGX_CONF_TAKE4)
41
42  #define NGX_CONF_ARGS_NUMBER 0x000000ff
43  #define NGX_CONF_BLOCK       0x00000100
44  #define NGX_CONF_FLAG        0x00000200
45  #define NGX_CONF_ANY         0x00000400
46  #define NGX_CONF_1MORE       0x00000800
47  #define NGX_CONF_2MORE       0x00001000
48  #define NGX_CONF_MULTI       0x00000000  /* compatibility */
49
50  #define NGX_DIRECT_CONF      0x00010000
```

```
51
52  #define NGX_MAIN_CONF        0x01000000
53  #define NGX_ANY_CONF         0x0F000000
54
55
56
57  #define NGX_CONF_UNSET       -1
58  #define NGX_CONF_UNSET_UINT  (ngx_uint_t) -1
59  #define NGX_CONF_UNSET_PTR   (void *) -1
60  #define NGX_CONF_UNSET_SIZE  (size_t) -1
61  #define NGX_CONF_UNSET_MSEC  (ngx_msec_t) -1
62
63
64  #define NGX_CONF_OK          NULL
65  #define NGX_CONF_ERROR       (void *) -1
66
67  #define NGX_CONF_BLOCK_START 1
68  #define NGX_CONF_BLOCK_DONE  2
69  #define NGX_CONF_FILE_DONE   3
70
71  #define NGX_CORE_MODULE      0x45524F43  /* "CORE" */
72  #define NGX_CONF_MODULE      0x464E4F43  /* "CONF" */
73
74
75  #define NGX_MAX_CONF_ERRSTR  1024
76
77
78  struct ngx_command_s {
79      ngx_str_t             name;
80      ngx_uint_t            type;
81      char               *(*set)(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
82      ngx_uint_t            conf;
83      ngx_uint_t            offset;
84      void                 *post;
85  };
86
87  #define ngx_null_command  { ngx_null_string, 0, NULL, 0, 0, NULL }
88
89
90  struct ngx_open_file_s {
91      ngx_fd_t              fd;
92      ngx_str_t             name;
93
94      void                (*flush)(ngx_open_file_t *file, ngx_log_t *log);
95      void                 *data;
96  };
97
98
99  #define NGX_MODULE_V1          0, 0, 0, 0, 0, 0, 1
100 #define NGX_MODULE_V1_PADDING  0, 0, 0, 0, 0, 0, 0, 0
101
102 struct ngx_module_s {
103     ngx_uint_t            ctx_index;
104     ngx_uint_t            index;
105
106     ngx_uint_t            spare0;
107     ngx_uint_t            spare1;
108     ngx_uint_t            spare2;
109     ngx_uint_t            spare3;
110
111     ngx_uint_t            version;
112
113     void                 *ctx;
114     ngx_command_t        *commands;
115     ngx_uint_t            type;
116
117     ngx_int_t           (*init_master)(ngx_log_t *log);
118
119     ngx_int_t           (*init_module)(ngx_cycle_t *cycle);
120
121     ngx_int_t           (*init_process)(ngx_cycle_t *cycle);
122     ngx_int_t           (*init_thread)(ngx_cycle_t *cycle);
123     void                (*exit_thread)(ngx_cycle_t *cycle);
124     void                (*exit_process)(ngx_cycle_t *cycle);
125
126     void                (*exit_master)(ngx_cycle_t *cycle);
```

```
127
128     uintptr_t             spare_hook0;
129     uintptr_t             spare_hook1;
130     uintptr_t             spare_hook2;
131     uintptr_t             spare_hook3;
132     uintptr_t             spare_hook4;
133     uintptr_t             spare_hook5;
134     uintptr_t             spare_hook6;
135     uintptr_t             spare_hook7;
136 };
137
138
139 typedef struct {
140     ngx_str_t             name;
141     void               *(*create_conf)(ngx_cycle_t *cycle);
142     char               *(*init_conf)(ngx_cycle_t *cycle, void *conf);
143 } ngx_core_module_t;
144
145
146 typedef struct {
147     ngx_file_t            file;
148     ngx_buf_t            *buffer;
149     ngx_uint_t            line;
150 } ngx_conf_file_t;
151
152
153 typedef char *(*ngx_conf_handler_pt)(ngx_conf_t *cf,
154     ngx_command_t *dummy, void *conf);
155
156
157 struct ngx_conf_s {
158     char                 *name;
159     ngx_array_t          *args;
160
161     ngx_cycle_t          *cycle;
162     ngx_pool_t           *pool;
163     ngx_pool_t           *temp_pool;
164     ngx_conf_file_t      *conf_file;
165     ngx_log_t            *log;
166
167     void                 *ctx;
168     ngx_uint_t            module_type;
169     ngx_uint_t            cmd_type;
170
171     ngx_conf_handler_pt   handler;
172     char                 *handler_conf;
173 };
174
175
176 typedef char *(*ngx_conf_post_handler_pt) (ngx_conf_t *cf,
177     void *data, void *conf);
178
179 typedef struct {
180     ngx_conf_post_handler_pt  post_handler;
181 } ngx_conf_post_t;
182
183
184 typedef struct {
185     ngx_conf_post_handler_pt  post_handler;
186     char                     *old_name;
187     char                     *new_name;
188 } ngx_conf_deprecated_t;
189
190
191 typedef struct {
192     ngx_conf_post_handler_pt  post_handler;
193     ngx_int_t                 low;
194     ngx_int_t                 high;
195 } ngx_conf_num_bounds_t;
196
197
198 typedef struct {
199     ngx_str_t                 name;
200     ngx_uint_t                value;
201 } ngx_conf_enum_t;
202
```

```
203
204 #define NGX_CONF_BITMASK_SET  1
205
206 typedef struct {
207     ngx_str_t                 name;
208     ngx_uint_t                mask;
209 } ngx_conf_bitmask_t;
210
211
212
213 char * ngx_conf_deprecated(ngx_conf_t *cf, void *post, void *data);
214 char *ngx_conf_check_num_bounds(ngx_conf_t *cf, void *post, void *data);
215
216
217 #define ngx_get_conf(conf_ctx, module)  conf_ctx[module.index]
218
219
220
221 #define ngx_conf_init_value(conf, default)                                   \
222     if (conf == NGX_CONF_UNSET) {                                            \
223         conf = default;                                                      \
224     }
225
226 #define ngx_conf_init_ptr_value(conf, default)                               \
227     if (conf == NGX_CONF_UNSET_PTR) {                                        \
228         conf = default;                                                      \
229     }
230
231 #define ngx_conf_init_uint_value(conf, default)                              \
232     if (conf == NGX_CONF_UNSET_UINT) {                                       \
233         conf = default;                                                      \
234     }
235
236 #define ngx_conf_init_size_value(conf, default)                              \
237     if (conf == NGX_CONF_UNSET_SIZE) {                                       \
238         conf = default;                                                      \
239     }
240
241 #define ngx_conf_init_msec_value(conf, default)                              \
242     if (conf == NGX_CONF_UNSET_MSEC) {                                       \
243         conf = default;                                                      \
244     }
245
246 #define ngx_conf_merge_value(conf, prev, default)                            \
247     if (conf == NGX_CONF_UNSET) {                                            \
248         conf = (prev == NGX_CONF_UNSET) ? default : prev;                    \
249     }
250
251 #define ngx_conf_merge_ptr_value(conf, prev, default)                        \
252     if (conf == NGX_CONF_UNSET_PTR) {                                        \
253         conf = (prev == NGX_CONF_UNSET_PTR) ? default : prev;                \
254     }
255
256 #define ngx_conf_merge_uint_value(conf, prev, default)                       \
257     if (conf == NGX_CONF_UNSET_UINT) {                                       \
258         conf = (prev == NGX_CONF_UNSET_UINT) ? default : prev;               \
259     }
260
261 #define ngx_conf_merge_msec_value(conf, prev, default)                       \
262     if (conf == NGX_CONF_UNSET_MSEC) {                                       \
263         conf = (prev == NGX_CONF_UNSET_MSEC) ? default : prev;               \
264     }
265
266 #define ngx_conf_merge_sec_value(conf, prev, default)                        \
267     if (conf == NGX_CONF_UNSET) {                                            \
268         conf = (prev == NGX_CONF_UNSET) ? default : prev;                    \
269     }
270
271 #define ngx_conf_merge_size_value(conf, prev, default)                       \
272     if (conf == NGX_CONF_UNSET_SIZE) {                                       \
273         conf = (prev == NGX_CONF_UNSET_SIZE) ? default : prev;               \
274     }
275
276 #define ngx_conf_merge_off_value(conf, prev, default)                        \
277     if (conf == NGX_CONF_UNSET) {                                            \
278         conf = (prev == NGX_CONF_UNSET) ? default : prev;                    \
```

```
    }

#define ngx_conf_merge_str_value(conf, prev, default)                        \
    if (conf.data == NULL) {                                                 \
        if (prev.data) {                                                     \
            conf.len = prev.len;                                             \
            conf.data = prev.data;                                           \
        } else {                                                             \
            conf.len = sizeof(default) - 1;                                  \
            conf.data = (u_char *) default;                                  \
        }                                                                    \
    }

#define ngx_conf_merge_bufs_value(conf, prev, default_num, default_size)     \
    if (conf.num == 0) {                                                     \
        if (prev.num) {                                                      \
            conf.num = prev.num;                                             \
            conf.size = prev.size;                                           \
        } else {                                                             \
            conf.num = default_num;                                          \
            conf.size = default_size;                                        \
        }                                                                    \
    }

#define ngx_conf_merge_bitmask_value(conf, prev, default)                    \
    if (conf == 0) {                                                         \
        conf = (prev == 0) ? default : prev;                                 \
    }


char *ngx_conf_param(ngx_conf_t *cf);
char *ngx_conf_parse(ngx_conf_t *cf, ngx_str_t *filename);
char *ngx_conf_include(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);


ngx_int_t ngx_conf_full_name(ngx_cycle_t *cycle, ngx_str_t *name,
    ngx_uint_t conf_prefix);
ngx_open_file_t *ngx_conf_open_file(ngx_cycle_t *cycle, ngx_str_t *name);
void ngx_cdecl ngx_conf_log_error(ngx_uint_t level, ngx_conf_t *cf,
    ngx_err_t err, const char *fmt, ...);


char *ngx_conf_set_flag_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_str_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_str_array_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
char *ngx_conf_set_keyval_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_num_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_size_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_off_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_msec_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_sec_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_bufs_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_enum_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
char *ngx_conf_set_bitmask_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);


extern ngx_uint_t     ngx_max_module;
extern ngx_module_t  *ngx_modules[];


#endif /* _NGX_CONF_FILE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_config.h - nginx-1.7.10

Data types defined

- ngx_flag_t
- ngx_int_t
- ngx_uint_t

Macros defined

- INADDR_NONE
- NGX_ALIGNMENT
- NGX_CHANGEBIN_SIGNAL
- NGX_CHANGEBIN_SIGNAL
- NGX_HAVE_SO_SNDLOWAT
- NGX_INT32_LEN
- NGX_INT64_LEN
- NGX_INT_T_LEN
- NGX_INT_T_LEN
- NGX_INVALID_ARRAY_INDEX
- NGX_MAXHOSTNAMELEN
- NGX_MAXHOSTNAMELEN
- NGX_MAX_INT32_VALUE
- NGX_MAX_UINT32_VALUE
- NGX_MAX_UINT32_VALUE
- NGX_NOACCEPT_SIGNAL
- NGX_RECONFIGURE_SIGNAL
- NGX_REOPEN_SIGNAL
- NGX_REOPEN_SIGNAL
- NGX_SHUTDOWN_SIGNAL
- NGX_TERMINATE_SIGNAL
- _NGX_CONFIG_H_INCLUDED_
- __FreeBSD__
- __FreeBSD_version
- ngx_abort

- ngx_align
- ngx_align_ptr
- ngx_cdecl
- ngx_inline
- ngx_libc_cdecl
- ngx_random
- ngx_signal_helper
- ngx_signal_value

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_CONFIG_H_INCLUDED_
9  #define _NGX_CONFIG_H_INCLUDED_
10
11
12  #include <ngx_auto_headers.h>
13
14
15  #if defined __DragonFly__ && !defined __FreeBSD__
16  #define __FreeBSD__        4
17  #define __FreeBSD_version  480101
18  #endif
19
20
21  #if (NGX_FREEBSD)
22  #include <ngx_freebsd_config.h>
23
24
25  #elif (NGX_LINUX)
26  #include <ngx_linux_config.h>
27
28
29  #elif (NGX_SOLARIS)
30  #include <ngx_solaris_config.h>
31
32
33  #elif (NGX_DARWIN)
34  #include <ngx_darwin_config.h>
35
36
37  #elif (NGX_WIN32)
38  #include <ngx_win32_config.h>
39
40
41  #else /* POSIX */
42  #include <ngx_posix_config.h>
43
44  #endif
45
46
47  #ifndef NGX_HAVE_SO_SNDLOWAT
48  #define NGX_HAVE_SO_SNDLOWAT     1
49  #endif
50
51
52  #if !(NGX_WIN32)
53
54  #define ngx_signal_helper(n)     SIG##n
```

```
#define ngx_signal_value(n)      ngx_signal_helper(n)

#define ngx_random               random

/* TODO: #ifndef */
#define NGX_SHUTDOWN_SIGNAL      QUIT
#define NGX_TERMINATE_SIGNAL     TERM
#define NGX_NOACCEPT_SIGNAL      WINCH
#define NGX_RECONFIGURE_SIGNAL   HUP

#if (NGX_LINUXTHREADS)
#define NGX_REOPEN_SIGNAL        INFO
#define NGX_CHANGEBIN_SIGNAL     XCPU
#else
#define NGX_REOPEN_SIGNAL        USR1
#define NGX_CHANGEBIN_SIGNAL     USR2
#endif

#define ngx_cdecl
#define ngx_libc_cdecl

#endif

typedef intptr_t        ngx_int_t;
typedef uintptr_t       ngx_uint_t;
typedef intptr_t        ngx_flag_t;


#define NGX_INT32_LEN   (sizeof("-2147483648") - 1)
#define NGX_INT64_LEN   (sizeof("-9223372036854775808") - 1)

#if (NGX_PTR_SIZE == 4)
#define NGX_INT_T_LEN   NGX_INT32_LEN
#else
#define NGX_INT_T_LEN   NGX_INT64_LEN
#endif


#ifndef NGX_ALIGNMENT
#define NGX_ALIGNMENT   sizeof(unsigned long)    /* platform word */
#endif

#define ngx_align(d, a)     (((d) + (a - 1)) & ~(a - 1))
#define ngx_align_ptr(p, a)                                                   \
    (u_char *) (((uintptr_t) (p) + ((uintptr_t) a - 1)) & ~((uintptr_t) a - 1))


 #define ngx_abort       abort


 /* TODO: platform specific: array[NGX_INVALID_ARRAY_INDEX] must cause SIGSEGV */
 #define NGX_INVALID_ARRAY_INDEX 0x80000000


 /* TODO: auto_conf: ngx_inline   inline __inline __inline__ */
 #ifndef ngx_inline
 #define ngx_inline      inline
 #endif

 #ifndef INADDR_NONE  /* Solaris */
 #define INADDR_NONE  ((unsigned int) -1)
 #endif

 #ifdef MAXHOSTNAMELEN
 #define NGX_MAXHOSTNAMELEN  MAXHOSTNAMELEN
 #else
 #define NGX_MAXHOSTNAMELEN  256
 #endif


 #if ((__GNU__ == 2) && (__GNUC_MINOR__ < 8))
 #define NGX_MAX_UINT32_VALUE  (uint32_t) 0xffffffffLL
 #else
 #define NGX_MAX_UINT32_VALUE  (uint32_t) 0xffffffff
 #endif

```

```
131 #define NGX_MAX_INT32_VALUE   (uint32_t) 0x7fffffff
132
133
134 #endif /* _NGX_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_linux_config.h - nginx-1.7.10

Data types defined

- ngx_aiocb_t

Macros defined

- NGX_HAVE_INHERITED_NONBLOCK
- NGX_HAVE_OS_SPECIFIC_INIT
- NGX_HAVE_SO_SNDLOWAT
- NGX_LISTEN_BACKLOG
- NGX_SENDFILE_LIMIT
- _FILE_OFFSET_BITS
- _GNU_SOURCE
- _NGX_LINUX_CONFIG_H_INCLUDED_
- ngx_debug_init

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_LINUX_CONFIG_H_INCLUDED_
9  #define _NGX_LINUX_CONFIG_H_INCLUDED_
10
11
12  #ifndef _GNU_SOURCE
13  #define _GNU_SOURCE             /* pread(), pwrite(), gethostname() */
14  #endif
15
16  #define _FILE_OFFSET_BITS  64
17
18  #include <sys/types.h>
19  #include <sys/time.h>
20  #include <unistd.h>
21  #include <stdarg.h>
22  #include <stddef.h>             /* offsetof() */
23  #include <stdio.h>
24  #include <stdlib.h>
25  #include <ctype.h>
26  #include <errno.h>
27  #include <string.h>
28  #include <signal.h>
29  #include <pwd.h>
30  #include <grp.h>
31  #include <dirent.h>
32  #include <glob.h>
33  #include <sys/vfs.h>            /* statfs() */
34
35  #include <sys/uio.h>
36  #include <sys/stat.h>
37  #include <fcntl.h>
38
```

```c
#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/resource.h>
#include <sched.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>        /* TCP_NODELAY, TCP_CORK */
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/un.h>

#include <time.h>               /* tzset() */
#include <malloc.h>             /* memalign() */
#include <limits.h>             /* IOV_MAX */
#include <sys/ioctl.h>
#include <crypt.h>
#include <sys/utsname.h>        /* uname() */


#include <ngx_auto_config.h>


#if (NGX_HAVE_POSIX_SEM)
#include <semaphore.h>
#endif


#if (NGX_HAVE_SYS_PRCTL_H)
#include <sys/prctl.h>
#endif


#if (NGX_HAVE_SENDFILE64)
#include <sys/sendfile.h>
#else
extern ssize_t sendfile(int s, int fd, int32_t *offset, size_t size);
#define NGX_SENDFILE_LIMIT  0x80000000
#endif


#if (NGX_HAVE_POLL)
#include <poll.h>
#endif


#if (NGX_HAVE_RTSIG)
#include <poll.h>
#include <sys/sysctl.h>
#endif


#if (NGX_HAVE_EPOLL)
#include <sys/epoll.h>
#endif


#if (NGX_HAVE_FILE_AIO)
#if (NGX_HAVE_SYS_EVENTFD_H)
#include <sys/eventfd.h>
#endif
 #include <sys/syscall.h>
 #include <linux/aio_abi.h>
 typedef struct iocb  ngx_aiocb_t;
 #endif


 #define NGX_LISTEN_BACKLOG        511


 #ifndef NGX_HAVE_SO_SNDLOWAT
 /* setsockopt(SO_SNDLOWAT) returns ENOPROTOOPT */
 #define NGX_HAVE_SO_SNDLOWAT         0
 #endif
```

```
#ifndef NGX_HAVE_INHERITED_NONBLOCK
#define NGX_HAVE_INHERITED_NONBLOCK  0
#endif


#define NGX_HAVE_OS_SPECIFIC_INIT    1
#define ngx_debug_init()


extern char **environ;


#endif /* _NGX_LINUX_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ - nginx-1.7.10

- ngx_aio_read.c
- ngx_aio_read_chain.c
- ngx_aio_write.c
- ngx_aio_write_chain.c
- ngx_alloc.c
- ngx_alloc.h
- ngx_atomic.h
- ngx_channel.c
- ngx_channel.h
- ngx_daemon.c
- ngx_darwin.h
- ngx_darwin_config.h
- ngx_darwin_init.c
- ngx_darwin_sendfile_chain.c
- ngx_errno.c
- ngx_errno.h
- ngx_file_aio_read.c
- ngx_files.c
- ngx_files.h
- ngx_freebsd.h
- ngx_freebsd_config.h
- ngx_freebsd_init.c
- ngx_freebsd_rfork_thread.c
- ngx_freebsd_rfork_thread.h
- ngx_freebsd_sendfile_chain.c
- ngx_gcc_atomic_amd64.h
- ngx_gcc_atomic_ppc.h
- ngx_gcc_atomic_sparc64.h
- ngx_gcc_atomic_x86.h
- ngx_linux.h
- ngx_linux_aio_read.c

- ngx_linux_config.h
- ngx_linux_init.c
- ngx_linux_sendfile_chain.c
- ngx_os.h
- ngx_posix_config.h
- ngx_posix_init.c
- ngx_process.c
- ngx_process.h
- ngx_process_cycle.c
- ngx_process_cycle.h
- ngx_pthread_thread.c
- ngx_readv_chain.c
- ngx_recv.c
- ngx_send.c
- ngx_setaffinity.c
- ngx_setaffinity.h
- ngx_setproctitle.c
- ngx_setproctitle.h
- ngx_shmem.c
- ngx_shmem.h
- ngx_socket.c
- ngx_socket.h
- ngx_solaris.h
- ngx_solaris_config.h
- ngx_solaris_init.c
- ngx_solaris_sendfilev_chain.c
- ngx_sunpro_atomic_sparc64.h
- ngx_thread.h
- ngx_time.c
- ngx_time.h
- ngx_udp_recv.c
- ngx_user.c
- ngx_user.h

- ngx_writev_chain.c
- rfork_thread.S

One Level Up Top Level

src/os/ - nginx-1.7.10

- unix/

src/os/unix/ngx_aio_read.c - nginx-1.7.10

Functions defined

- ngx_aio_read

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  extern int  ngx_kqueue;
14
15
16  ssize_t
17  ngx_aio_read(ngx_connection_t *c, u_char *buf, size_t size)
18  {
19      int           n;
20      ngx_event_t  *rev;
21
22      rev = c->read;
23
24      if (!rev->ready) {
25          ngx_log_error(NGX_LOG_ALERT, c->log, 0, "second aio post");
26          return NGX_AGAIN;
27      }
28
29      ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
30                     "rev->complete: %d", rev->complete);
31      ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
32                     "aio size: %d", size);
33
34      if (!rev->complete) {
35          ngx_memzero(&rev->aiocb, sizeof(struct aiocb));
36
37          rev->aiocb.aio_fildes = c->fd;
38          rev->aiocb.aio_buf = buf;
39          rev->aiocb.aio_nbytes = size;
40
41  #if (NGX_HAVE_KQUEUE)
42          rev->aiocb.aio_sigevent.sigev_notify_kqueue = ngx_kqueue;
43          rev->aiocb.aio_sigevent.sigev_notify = SIGEV_KEVENT;
44          rev->aiocb.aio_sigevent.sigev_value.sigval_ptr = rev;
45  #endif
46
47          if (aio_read(&rev->aiocb) == -1) {
48              ngx_log_error(NGX_LOG_CRIT, rev->log, ngx_errno,
49                            "aio_read() failed");
50              rev->error = 1;
51              return NGX_ERROR;
52          }
53
54          ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
55                         "aio_read: #%d OK", c->fd);
56
57          rev->active = 1;
58          rev->ready = 0;
59      }
60
61      rev->complete = 0;
62
```

```
63      n = aio_error(&rev->aiocb);
64      if (n == -1) {
65          ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno, "aio_error() failed");
66          rev->error = 1;
67          return NGX_ERROR;
68      }
69
70      if (n != 0) {
71          if (n == NGX_EINPROGRESS) {
72              if (rev->ready) {
73                  ngx_log_error(NGX_LOG_ALERT, c->log, n,
74                                "aio_read() still in progress");
75                  rev->ready = 0;
76              }
77              return NGX_AGAIN;
78          }
79
80          ngx_log_error(NGX_LOG_CRIT, c->log, n, "aio_read() failed");
81          rev->error = 1;
82          rev->ready = 0;
83          return NGX_ERROR;
84      }
85
86      n = aio_return(&rev->aiocb);
87      if (n == -1) {
88          ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
89                        "aio_return() failed");
90
91          rev->error = 1;
92          rev->ready = 0;
93          return NGX_ERROR;
94      }
95
96      ngx_log_debug2(NGX_LOG_DEBUG_EVENT, rev->log, 0,
97                     "aio_read: #%d %d", c->fd, n);
98
99      if (n == 0) {
100          rev->eof = 1;
101          rev->ready = 0;
102      } else {
103          rev->ready = 1;
104      }
105
106      rev->active = 0;
107
108      return n;
109  }
```

One Level Up Top Level

src/event/modules/ngx_kqueue_module.c - nginx-1.7.10

Global variables defined

- change_list
- change_list0
- change_list1
- event_list
- kevent_mutex
- kqueue_name
- list_mutex
- max_changes
- nchanges
- nevents
- ngx_kqueue
- ngx_kqueue_commands
- ngx_kqueue_module
- ngx_kqueue_module_ctx

Data types defined

- ngx_kqueue_conf_t

Functions defined

- ngx_kqueue_add_event
- ngx_kqueue_create_conf
- ngx_kqueue_del_event
- ngx_kqueue_done
- ngx_kqueue_dump_event
- ngx_kqueue_init
- ngx_kqueue_init_conf
- ngx_kqueue_process_changes
- ngx_kqueue_process_events
- ngx_kqueue_set_event

Source code

```
1
```

```c
/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_event.h>


 typedef struct {
     ngx_uint_t  changes;
     ngx_uint_t  events;
 } ngx_kqueue_conf_t;


 static ngx_int_t ngx_kqueue_init(ngx_cycle_t *cycle, ngx_msec_t timer);
 static void ngx_kqueue_done(ngx_cycle_t *cycle);
 static ngx_int_t ngx_kqueue_add_event(ngx_event_t *ev, ngx_int_t event,
     ngx_uint_t flags);
 static ngx_int_t ngx_kqueue_del_event(ngx_event_t *ev, ngx_int_t event,
     ngx_uint_t flags);
 static ngx_int_t ngx_kqueue_set_event(ngx_event_t *ev, ngx_int_t filter,
     ngx_uint_t flags);
 static ngx_int_t ngx_kqueue_process_changes(ngx_cycle_t *cycle, ngx_uint_t try);
 static ngx_int_t ngx_kqueue_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
     ngx_uint_t flags);
 static ngx_inline void ngx_kqueue_dump_event(ngx_log_t *log,
     struct kevent *kev);

 static void *ngx_kqueue_create_conf(ngx_cycle_t *cycle);
 static char *ngx_kqueue_init_conf(ngx_cycle_t *cycle, void *conf);


 int                     ngx_kqueue = -1;

 /*
 * The "change_list" should be declared as ngx_thread_volatile.
 * However, the use of the change_list is localized in kqueue functions and
 * is protected by the mutex so even the "icc -ipo" should not build the code
 * with the race condition.  Thus we avoid the declaration to make a more
 * readable code.
 */

 static struct kevent  *change_list, *change_list0, *change_list1;
 static struct kevent  *event_list;
 static ngx_uint_t      max_changes, nchanges, nevents;

 #if (NGX_THREADS)
 static ngx_mutex_t    *list_mutex;
 static ngx_mutex_t    *kevent_mutex;
 #endif



 static ngx_str_t      kqueue_name = ngx_string("kqueue");

 static ngx_command_t  ngx_kqueue_commands[] = {

     { ngx_string("kqueue_changes"),
       NGX_EVENT_CONF|NGX_CONF_TAKE1,
       ngx_conf_set_num_slot,
       0,
       offsetof(ngx_kqueue_conf_t, changes),
       NULL },

     { ngx_string("kqueue_events"),
       NGX_EVENT_CONF|NGX_CONF_TAKE1,
       ngx_conf_set_num_slot,
       0,
       offsetof(ngx_kqueue_conf_t, events),
       NULL },

       ngx_null_command
 };
```

```
78
79
80  ngx_event_module_t  ngx_kqueue_module_ctx = {
81      &kqueue_name,
82      ngx_kqueue_create_conf,                /* create configuration */
83      ngx_kqueue_init_conf,                  /* init configuration */
84
85      {
86          ngx_kqueue_add_event,              /* add an event */
87          ngx_kqueue_del_event,              /* delete an event */
88          ngx_kqueue_add_event,              /* enable an event */
89          ngx_kqueue_del_event,              /* disable an event */
90          NULL,                              /* add an connection */
91          NULL,                              /* delete an connection */
92          ngx_kqueue_process_changes,        /* process the changes */
93          ngx_kqueue_process_events,         /* process the events */
94          ngx_kqueue_init,                   /* init the events */
95          ngx_kqueue_done                    /* done the events */
96      }
97
98  };
99
100 ngx_module_t  ngx_kqueue_module = {
101     NGX_MODULE_V1,
102     &ngx_kqueue_module_ctx,                 /* module context */
103     ngx_kqueue_commands,                    /* module directives */
104     NGX_EVENT_MODULE,                       /* module type */
105     NULL,                                   /* init master */
106     NULL,                                   /* init module */
107     NULL,                                   /* init process */
108     NULL,                                   /* init thread */
109     NULL,                                   /* exit thread */
110     NULL,                                   /* exit process */
111     NULL,                                   /* exit master */
112     NGX_MODULE_V1_PADDING
113 };
114
115
116 static ngx_int_t
117 ngx_kqueue_init(ngx_cycle_t *cycle, ngx_msec_t timer)
118 {
119     ngx_kqueue_conf_t  *kcf;
120     struct timespec     ts;
121 #if (NGX_HAVE_TIMER_EVENT)
122     struct kevent       kev;
123 #endif
124
125     kcf = ngx_event_get_conf(cycle->conf_ctx, ngx_kqueue_module);
126
127     if (ngx_kqueue == -1) {
128         ngx_kqueue = kqueue();
129
130         if (ngx_kqueue == -1) {
131             ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
132                           "kqueue() failed");
133             return NGX_ERROR;
134         }
135
136 #if (NGX_THREADS)
137
138         list_mutex = ngx_mutex_init(cycle->log, 0);
139         if (list_mutex == NULL) {
140             return NGX_ERROR;
141         }
142
143         kevent_mutex = ngx_mutex_init(cycle->log, 0);
144         if (kevent_mutex == NULL) {
145             return NGX_ERROR;
146         }
147
148 #endif
149     }
150
151     if (max_changes < kcf->changes) {
152         if (nchanges) {
153             ts.tv_sec = 0;
```

```
            ts.tv_nsec = 0;

            if (kevent(ngx_kqueue, change_list, (int) nchanges, NULL, 0, &ts)
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                              "kevent() failed");
                return NGX_ERROR;
            }
            nchanges = 0;
        }

        if (change_list0) {
            ngx_free(change_list0);
        }

        change_list0 = ngx_alloc(kcf->changes * sizeof(struct kevent),
                                 cycle->log);
        if (change_list0 == NULL) {
            return NGX_ERROR;
        }

        if (change_list1) {
            ngx_free(change_list1);
        }

        change_list1 = ngx_alloc(kcf->changes * sizeof(struct kevent),
                                 cycle->log);
        if (change_list1 == NULL) {
            return NGX_ERROR;
        }

        change_list = change_list0;
    }

    max_changes = kcf->changes;

    if (nevents < kcf->events) {
        if (event_list) {
            ngx_free(event_list);
        }

        event_list = ngx_alloc(kcf->events * sizeof(struct kevent), cycle->log);
        if (event_list == NULL) {
            return NGX_ERROR;
        }
    }

    ngx_event_flags = NGX_USE_ONESHOT_EVENT
                      |NGX_USE_KQUEUE_EVENT
                      |NGX_USE_VNODE_EVENT;

#if (NGX_HAVE_TIMER_EVENT)

    if (timer) {
        kev.ident = 0;
        kev.filter = EVFILT_TIMER;
        kev.flags = EV_ADD|EV_ENABLE;
        kev.fflags = 0;
        kev.data = timer;
        kev.udata = 0;

        ts.tv_sec = 0;
        ts.tv_nsec = 0;

        if (kevent(ngx_kqueue, &kev, 1, NULL, 0, &ts) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "kevent(EVFILT_TIMER) failed");
            return NGX_ERROR;
        }

        ngx_event_flags |= NGX_USE_TIMER_EVENT;
    }

#endif

```

```
#if (NGX_HAVE_CLEAR_EVENT)
    ngx_event_flags |= NGX_USE_CLEAR_EVENT;
#else
    ngx_event_flags |= NGX_USE_LEVEL_EVENT;
#endif

#if (NGX_HAVE_LOWAT_EVENT)
    ngx_event_flags |= NGX_USE_LOWAT_EVENT;
#endif

    nevents = kcf->events;

    ngx_io = ngx_os_io;

    ngx_event_actions = ngx_kqueue_module_ctx.actions;

    return NGX_OK;
}


static void
ngx_kqueue_done(ngx_cycle_t *cycle)
{
    if (close(ngx_kqueue) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "kqueue close() failed");
    }

    ngx_kqueue = -1;

#if (NGX_THREADS)
    ngx_mutex_destroy(kevent_mutex);
    ngx_mutex_destroy(list_mutex);
#endif

    ngx_free(change_list1);
    ngx_free(change_list0);
    ngx_free(event_list);

    change_list1 = NULL;
    change_list0 = NULL;
    change_list = NULL;
    event_list = NULL;
    max_changes = 0;
    nchanges = 0;
    nevents = 0;
}


static ngx_int_t
ngx_kqueue_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_int_t          rc;
#if 0
    ngx_event_t       *e;
    ngx_connection_t  *c;
#endif

    ev->active = 1;
    ev->disabled = 0;
    ev->oneshot = (flags & NGX_ONESHOT_EVENT) ? 1 : 0;

    ngx_mutex_lock(list_mutex);

#if 0

    if (ev->index < nchanges
        && ((uintptr_t) change_list[ev->index].udata & (uintptr_t) ~1)
            == (uintptr_t) ev)
    {
        if (change_list[ev->index].flags == EV_DISABLE) {

            /*
             * if the EV_DISABLE is still not passed to a kernel
             * we will not pass it
             */
```

```
            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                           "kevent activated: %d: ft:%i",
                           ngx_event_ident(ev->data), event);

            if (ev->index < --nchanges) {
                e = (ngx_event_t *)
                    ((uintptr_t) change_list[nchanges].udata & (uintptr_t) ~1);
                change_list[ev->index] = change_list[nchanges];
                e->index = ev->index;
            }

            ngx_mutex_unlock(list_mutex);

            return NGX_OK;
        }

        c = ev->data;

        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "previous event on #%d were not passed in kernel", c->fd);

        ngx_mutex_unlock(list_mutex);

        return NGX_ERROR;
    }

#endif

    rc = ngx_kqueue_set_event(ev, event, EV_ADD|EV_ENABLE|flags);

    ngx_mutex_unlock(list_mutex);

    return rc;
}


static ngx_int_t
ngx_kqueue_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_int_t     rc;
    ngx_event_t  *e;

    ev->active = 0;
    ev->disabled = 0;

    ngx_mutex_lock(list_mutex);

    if (ev->index < nchanges
        && ((uintptr_t) change_list[ev->index].udata & (uintptr_t) ~1)
            == (uintptr_t) ev)
    {
        ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                       "kevent deleted: %d: ft:%i",
                       ngx_event_ident(ev->data), event);

        /* if the event is still not passed to a kernel we will not pass it */

        nchanges--;

        if (ev->index < nchanges) {
            e = (ngx_event_t *)
                    ((uintptr_t) change_list[nchanges].udata & (uintptr_t) ~1);
            change_list[ev->index] = change_list[nchanges];
            e->index = ev->index;
        }

        ngx_mutex_unlock(list_mutex);

        return NGX_OK;
    }

    /*
     * when the file descriptor is closed the kqueue automatically deletes
     * its filters so we do not need to delete explicitly the event
     * before the closing the file descriptor.
```

```
382      */
383
384     if (flags & NGX_CLOSE_EVENT) {
385         ngx_mutex_unlock(list_mutex);
386         return NGX_OK;
387     }
388
389     if (flags & NGX_DISABLE_EVENT) {
390         ev->disabled = 1;
391
392     } else {
393         flags |= EV_DELETE;
394     }
395
396     rc = ngx_kqueue_set_event(ev, event, flags);
397
398     ngx_mutex_unlock(list_mutex);
399
400     return rc;
401 }
402
403
404 static ngx_int_t
405 ngx_kqueue_set_event(ngx_event_t *ev, ngx_int_t filter, ngx_uint_t flags)
406 {
407     struct kevent     *kev;
408     struct timespec    ts;
409     ngx_connection_t  *c;
410
411     c = ev->data;
412
413     ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ev->log, 0,
414                    "kevent set event: %d: ft:%i fl:%04Xi",
415                    c->fd, filter, flags);
416
417     if (nchanges >= max_changes) {
418         ngx_log_error(NGX_LOG_WARN, ev->log, 0,
419                       "kqueue change list is filled up");
420
421         ts.tv_sec = 0;
422         ts.tv_nsec = 0;
423
424         if (kevent(ngx_kqueue, change_list, (int) nchanges, NULL, 0, &ts)
425             == -1)
426         {
427             ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno, "kevent() failed");
428             return NGX_ERROR;
429         }
430
431         nchanges = 0;
432     }
433
434     kev = &change_list[nchanges];
435
436     kev->ident = c->fd;
437     kev->filter = (short) filter;
438     kev->flags = (u_short) flags;
439     kev->udata = NGX_KQUEUE_UDATA_T ((uintptr_t) ev | ev->instance);
440
441     if (filter == EVFILT_VNODE) {
442         kev->fflags = NOTE_DELETE|NOTE_WRITE|NOTE_EXTEND
443                                  |NOTE_ATTRIB|NOTE_RENAME
444 #if (__FreeBSD__ == 4 && __FreeBSD_version >= 430000) \
445     || __FreeBSD_version >= 500018
446                                  |NOTE_REVOKE
447 #endif
448                       ;
449         kev->data = 0;
450
451     } else {
452 #if (NGX_HAVE_LOWAT_EVENT)
453         if (flags & NGX_LOWAT_EVENT) {
454             kev->fflags = NOTE_LOWAT;
455             kev->data = ev->available;
456
457         } else {
```

```
            kev->fflags = 0;
            kev->data = 0;
        }
#else
        kev->fflags = 0;
        kev->data = 0;
#endif
    }

    ev->index = nchanges;
    nchanges++;

    if (flags & NGX_FLUSH_EVENT) {
        ts.tv_sec = 0;
        ts.tv_nsec = 0;

        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ev->log, 0, "kevent flush");

        if (kevent(ngx_kqueue, change_list, (int) nchanges, NULL, 0, &ts)
            == -1)
        {
            ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno, "kevent() failed");
            return NGX_ERROR;
        }

        nchanges = 0;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_kqueue_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
    ngx_uint_t flags)
{
    int               events, n;
    ngx_int_t         i, instance;
    ngx_uint_t        level;
    ngx_err_t         err;
    ngx_event_t      *ev;
    ngx_queue_t      *queue;
    struct timespec   ts, *tp;

    if (ngx_threaded) {
        if (ngx_kqueue_process_changes(cycle, 0) == NGX_ERROR) {
            return NGX_ERROR;
        }

        n = 0;

    } else {
        n = (int) nchanges;
        nchanges = 0;
    }

    if (timer == NGX_TIMER_INFINITE) {
        tp = NULL;

    } else {

        ts.tv_sec = timer / 1000;
        ts.tv_nsec = (timer % 1000) * 1000000;

        /*
         * 64-bit Darwin kernel has the bug: kernel level ts.tv_nsec is
         * the int32_t while user level ts.tv_nsec is the long (64-bit),
         * so on the big endian PowerPC all nanoseconds are lost.
         */

#if (NGX_DARWIN_KEVENT_BUG)
        ts.tv_nsec <<= 32;
#endif

        tp = &ts;
    }
```

```
534
535     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
536                    "kevent timer: %M, changes: %d", timer, n);
537
538     events = kevent(ngx_kqueue, change_list, n, event_list, (int) nevents, tp);
539
540     err = (events == -1) ? ngx_errno : 0;
541
542     if (flags & NGX_UPDATE_TIME || ngx_event_timer_alarm) {
543         ngx_time_update();
544     }
545
546     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
547                    "kevent events: %d", events);
548
549     if (err) {
550         if (err == NGX_EINTR) {
551
552             if (ngx_event_timer_alarm) {
553                 ngx_event_timer_alarm = 0;
554                 return NGX_OK;
555             }
556
557             level = NGX_LOG_INFO;
558
559         } else {
560             level = NGX_LOG_ALERT;
561         }
562
563         ngx_log_error(level, cycle->log, err, "kevent() failed");
564         return NGX_ERROR;
565     }
566
567     if (events == 0) {
568         if (timer != NGX_TIMER_INFINITE) {
569             return NGX_OK;
570         }
571
572         ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
573                       "kevent() returned no events without timeout");
574         return NGX_ERROR;
575     }
576
577     for (i = 0; i < events; i++) {
578
579         ngx_kqueue_dump_event(cycle->log, &event_list[i]);
580
581         if (event_list[i].flags & EV_ERROR) {
582             ngx_log_error(NGX_LOG_ALERT, cycle->log, event_list[i].data,
583                           "kevent() error on %d filter:%d flags:%04Xd",
584                           event_list[i].ident, event_list[i].filter,
585                           event_list[i].flags);
586             continue;
587         }
588
589 #if (NGX_HAVE_TIMER_EVENT)
590
591         if (event_list[i].filter == EVFILT_TIMER) {
592             ngx_time_update();
593             continue;
594         }
595
596 #endif
597
598         ev = (ngx_event_t *) event_list[i].udata;
599
600         switch (event_list[i].filter) {
601
602         case EVFILT_READ:
603         case EVFILT_WRITE:
604
605             instance = (uintptr_t) ev & 1;
606             ev = (ngx_event_t *) ((uintptr_t) ev & (uintptr_t) ~1);
607
608             if (ev->closed || ev->instance != instance) {
609
```

```
                /*
                 * the stale event from a file descriptor
                 * that was just closed in this iteration
                 */

                ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                               "kevent: stale event %p", ev);
                continue;
            }

            if (ev->log && (ev->log->log_level & NGX_LOG_DEBUG_CONNECTION)) {
                ngx_kqueue_dump_event(ev->log, &event_list[i]);
            }

            if (ev->oneshot) {
                ev->active = 0;
            }

            ev->available = event_list[i].data;

            if (event_list[i].flags & EV_EOF) {
                ev->pending_eof = 1;
                ev->kq_errno = event_list[i].fflags;
            }

            ev->ready = 1;

            break;

        case EVFILT_VNODE:
            ev->kq_vnode = 1;

            break;

        case EVFILT_AIO:
            ev->complete = 1;
            ev->ready = 1;

            break;

        default:
            ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                          "unexpected kevent() filter %d",
                          event_list[i].filter);
            continue;
        }

        if (flags & NGX_POST_EVENTS) {
            queue = ev->accept ? &ngx_posted_accept_events
                               : &ngx_posted_events;

            ngx_post_event(ev, queue);

            continue;
        }

        ev->handler(ev);
    }

    return NGX_OK;
}


static ngx_int_t
ngx_kqueue_process_changes(ngx_cycle_t *cycle, ngx_uint_t try)
{
    int               n;
    ngx_int_t         rc;
    ngx_err_t         err;
    struct timespec   ts;
    struct kevent    *changes;

    ngx_mutex_lock(kevent_mutex);

    ngx_mutex_lock(list_mutex);

```

```
    if (nchanges == 0) {
        ngx_mutex_unlock(list_mutex);
        ngx_mutex_unlock(kevent_mutex);
        return NGX_OK;
    }

    changes = change_list;
    if (change_list == change_list0) {
        change_list = change_list1;
    } else {
        change_list = change_list0;
    }

    n = (int) nchanges;
    nchanges = 0;

    ngx_mutex_unlock(list_mutex);

    ts.tv_sec = 0;
    ts.tv_nsec = 0;

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "kevent changes: %d", n);

    if (kevent(ngx_kqueue, changes, n, NULL, 0, &ts) == -1) {
        err = ngx_errno;
        ngx_log_error((err == NGX_EINTR) ? NGX_LOG_INFO : NGX_LOG_ALERT,
                      cycle->log, err, "kevent() failed");
        rc = NGX_ERROR;

    } else {
        rc = NGX_OK;
    }

    ngx_mutex_unlock(kevent_mutex);

    return rc;
}


static ngx_inline void
ngx_kqueue_dump_event(ngx_log_t *log, struct kevent *kev)
{
    ngx_log_debug6(NGX_LOG_DEBUG_EVENT, log, 0,
                   (kev->ident > 0x8000000 && kev->ident != (unsigned) -1) ?
                    "kevent: %p: ft:%d fl:%04Xd ff:%08Xd d:%d ud:%p":
                    "kevent: %d: ft:%d fl:%04Xd ff:%08Xd d:%d ud:%p",
                    kev->ident, kev->filter,
                    kev->flags, kev->fflags,
                    kev->data, kev->udata);
}


static void *
ngx_kqueue_create_conf(ngx_cycle_t *cycle)
{
    ngx_kqueue_conf_t  *kcf;

    kcf = ngx_palloc(cycle->pool, sizeof(ngx_kqueue_conf_t));
    if (kcf == NULL) {
        return NULL;
    }

    kcf->changes = NGX_CONF_UNSET;
    kcf->events = NGX_CONF_UNSET;

    return kcf;
}


static char *
ngx_kqueue_init_conf(ngx_cycle_t *cycle, void *conf)
{
    ngx_kqueue_conf_t *kcf = conf;

    ngx_conf_init_uint_value(kcf->changes, 512);
```

```
762     ngx_conf_init_uint_value(kcf->events, 512);
763
764     return NGX_CONF_OK;
765 }
```

One Level Up Top Level

src/event/modules/ - nginx-1.7.10

- ngx_aio_module.c
- ngx_devpoll_module.c
- ngx_epoll_module.c
- ngx_eventport_module.c
- ngx_kqueue_module.c
- ngx_poll_module.c
- ngx_rtsig_module.c
- ngx_select_module.c
- ngx_win32_select_module.c

src/event/ - nginx-1.7.10

- modules/
- ngx_event.c
- ngx_event.h
- ngx_event_accept.c
- ngx_event_busy_lock.c
- ngx_event_busy_lock.h
- ngx_event_connect.c
- ngx_event_connect.h
- ngx_event_mutex.c
- ngx_event_openssl.c
- ngx_event_openssl.h
- ngx_event_openssl_stapling.c
- ngx_event_pipe.c
- ngx_event_pipe.h
- ngx_event_posted.c
- ngx_event_posted.h
- ngx_event_timer.c
- ngx_event_timer.h

src/event/ngx_event.c - nginx-1.7.10

Global variables defined

- connection_counter
- event_core_name
- ngx_accept_disabled
- ngx_accept_events
- ngx_accept_mutex
- ngx_accept_mutex_delay
- ngx_accept_mutex_held
- ngx_accept_mutex_ptr
- ngx_connection_counter
- ngx_event_actions
- ngx_event_core_commands
- ngx_event_core_module
- ngx_event_core_module_ctx
- ngx_event_flags
- ngx_event_max_module
- ngx_event_timer_alarm
- ngx_events_commands
- ngx_events_module
- ngx_events_module_ctx
- ngx_stat_accepted
- ngx_stat_accepted0
- ngx_stat_active
- ngx_stat_active0
- ngx_stat_handled
- ngx_stat_handled0
- ngx_stat_reading
- ngx_stat_reading0
- ngx_stat_requests
- ngx_stat_requests0
- ngx_stat_waiting

- ngx_stat_waiting0
- ngx_stat_writing
- ngx_stat_writing0
- ngx_timer_resolution
- ngx_use_accept_mutex

Functions defined

- ngx_event_connections
- ngx_event_core_create_conf
- ngx_event_core_init_conf
- ngx_event_debug_connection
- ngx_event_init_conf
- ngx_event_module_init
- ngx_event_process_init
- ngx_event_use
- ngx_events_block
- ngx_handle_read_event
- ngx_handle_write_event
- ngx_process_events_and_timers
- ngx_send_lowat
- ngx_timer_signal_handler

Macros defined

- DEFAULT_CONNECTIONS

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #define DEFAULT_CONNECTIONS  512
14
15
16  extern ngx_module_t ngx_kqueue_module;
17  extern ngx_module_t ngx_eventport_module;
18  extern ngx_module_t ngx_devpoll_module;
19  extern ngx_module_t ngx_epoll_module;
20  extern ngx_module_t ngx_rtsig_module;
```

```
extern ngx_module_t ngx_select_module;


static char *ngx_event_init_conf(ngx_cycle_t *cycle, void *conf);
static ngx_int_t ngx_event_module_init(ngx_cycle_t *cycle);
static ngx_int_t ngx_event_process_init(ngx_cycle_t *cycle);
static char *ngx_events_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);

static char *ngx_event_connections(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_event_use(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static char *ngx_event_debug_connection(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);

static void *ngx_event_core_create_conf(ngx_cycle_t *cycle);
static char *ngx_event_core_init_conf(ngx_cycle_t *cycle, void *conf);


static ngx_uint_t     ngx_timer_resolution;
sig_atomic_t          ngx_event_timer_alarm;

static ngx_uint_t     ngx_event_max_module;

ngx_uint_t            ngx_event_flags;
ngx_event_actions_t   ngx_event_actions;


static ngx_atomic_t   connection_counter = 1;
ngx_atomic_t         *ngx_connection_counter = &connection_counter;


ngx_atomic_t         *ngx_accept_mutex_ptr;
ngx_shmtx_t           ngx_accept_mutex;
ngx_uint_t            ngx_use_accept_mutex;
ngx_uint_t            ngx_accept_events;
ngx_uint_t            ngx_accept_mutex_held;
ngx_msec_t            ngx_accept_mutex_delay;
ngx_int_t             ngx_accept_disabled;


#if (NGX_STAT_STUB)

ngx_atomic_t   ngx_stat_accepted0;
ngx_atomic_t  *ngx_stat_accepted = &ngx_stat_accepted0;
ngx_atomic_t   ngx_stat_handled0;
ngx_atomic_t  *ngx_stat_handled = &ngx_stat_handled0;
ngx_atomic_t   ngx_stat_requests0;
ngx_atomic_t  *ngx_stat_requests = &ngx_stat_requests0;
ngx_atomic_t   ngx_stat_active0;
ngx_atomic_t  *ngx_stat_active = &ngx_stat_active0;
ngx_atomic_t   ngx_stat_reading0;
ngx_atomic_t  *ngx_stat_reading = &ngx_stat_reading0;
ngx_atomic_t   ngx_stat_writing0;
ngx_atomic_t  *ngx_stat_writing = &ngx_stat_writing0;
ngx_atomic_t   ngx_stat_waiting0;
ngx_atomic_t  *ngx_stat_waiting = &ngx_stat_waiting0;

#endif



static ngx_command_t  ngx_events_commands[] = {

    { ngx_string("events"),
      NGX_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
      ngx_events_block,
      0,
      0,
      NULL },

      ngx_null_command
};


static ngx_core_module_t  ngx_events_module_ctx = {
    ngx_string("events"),
```

```
    NULL,
    ngx_event_init_conf
};


ngx_module_t  ngx_events_module = {
    NGX_MODULE_V1,
    &ngx_events_module_ctx,                /* module context */
    ngx_events_commands,                   /* module directives */
    NGX_CORE_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t  event_core_name = ngx_string("event_core");


static ngx_command_t  ngx_event_core_commands[] = {

    { ngx_string("worker_connections"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_event_connections,
      0,
      0,
      NULL },

    { ngx_string("connections"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_event_connections,
      0,
      0,
      NULL },

    { ngx_string("use"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_event_use,
      0,
      0,
      NULL },

    { ngx_string("multi_accept"),
      NGX_EVENT_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      0,
      offsetof(ngx_event_conf_t, multi_accept),
      NULL },

    { ngx_string("accept_mutex"),
      NGX_EVENT_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      0,
      offsetof(ngx_event_conf_t, accept_mutex),
      NULL },

    { ngx_string("accept_mutex_delay"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      0,
      offsetof(ngx_event_conf_t, accept_mutex_delay),
      NULL },

    { ngx_string("debug_connection"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_event_debug_connection,
      0,
      0,
      NULL },

      ngx_null_command
```

```
};


ngx_event_module_t  ngx_event_core_module_ctx = {
    &event_core_name,
    ngx_event_core_create_conf,            /* create configuration */
    ngx_event_core_init_conf,              /* init configuration */

    { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL }
};


ngx_module_t  ngx_event_core_module = {
    NGX_MODULE_V1,
    &ngx_event_core_module_ctx,            /* module context */
    ngx_event_core_commands,               /* module directives */
    NGX_EVENT_MODULE,                      /* module type */
    NULL,                                  /* init master */
    ngx_event_module_init,                 /* init module */
    ngx_event_process_init,                /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


void
ngx_process_events_and_timers(ngx_cycle_t *cycle)
{
    ngx_uint_t  flags;
    ngx_msec_t  timer, delta;

    if (ngx_timer_resolution) {
        timer = NGX_TIMER_INFINITE;
        flags = 0;

    } else {
        timer = ngx_event_find_timer();
        flags = NGX_UPDATE_TIME;

#if (NGX_THREADS)

        if (timer == NGX_TIMER_INFINITE || timer > 500) {
            timer = 500;
        }

#endif
    }

    if (ngx_use_accept_mutex) {
        if (ngx_accept_disabled > 0) {
            ngx_accept_disabled--;

        } else {
            if (ngx_trylock_accept_mutex(cycle) == NGX_ERROR) {
                return;
            }

            if (ngx_accept_mutex_held) {
                flags |= NGX_POST_EVENTS;

            } else {
                if (timer == NGX_TIMER_INFINITE
                    || timer > ngx_accept_mutex_delay)
                {
                    timer = ngx_accept_mutex_delay;
                }
            }
        }
    }

    delta = ngx_current_msec;

    (void) ngx_process_events(cycle, timer, flags);
```

```
249
250     delta = ngx_current_msec - delta;
251
252     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
253                    "timer delta: %M", delta);
254
255     ngx_event_process_posted(cycle, &ngx_posted_accept_events);
256
257     if (ngx_accept_mutex_held) {
258         ngx_shmtx_unlock(&ngx_accept_mutex);
259     }
260
261     if (delta) {
262         ngx_event_expire_timers();
263     }
264
265     ngx_event_process_posted(cycle, &ngx_posted_events);
266 }
267
268
269 ngx_int_t
270 ngx_handle_read_event(ngx_event_t *rev, ngx_uint_t flags)
271 {
272     if (ngx_event_flags & NGX_USE_CLEAR_EVENT) {
273
274         /* kqueue, epoll */
275
276         if (!rev->active && !rev->ready) {
277             if (ngx_add_event(rev, NGX_READ_EVENT, NGX_CLEAR_EVENT)
278                 == NGX_ERROR)
279             {
280                 return NGX_ERROR;
281             }
282         }
283
284         return NGX_OK;
285
286     } else if (ngx_event_flags & NGX_USE_LEVEL_EVENT) {
287
288         /* select, poll, /dev/poll */
289
290         if (!rev->active && !rev->ready) {
291             if (ngx_add_event(rev, NGX_READ_EVENT, NGX_LEVEL_EVENT)
292                 == NGX_ERROR)
293             {
294                 return NGX_ERROR;
295             }
296
297             return NGX_OK;
298         }
299
300         if (rev->active && (rev->ready || (flags & NGX_CLOSE_EVENT))) {
301             if (ngx_del_event(rev, NGX_READ_EVENT, NGX_LEVEL_EVENT | flags)
302                 == NGX_ERROR)
303             {
304                 return NGX_ERROR;
305             }
306
307             return NGX_OK;
308         }
309
310     } else if (ngx_event_flags & NGX_USE_EVENTPORT_EVENT) {
311
312         /* event ports */
313
314         if (!rev->active && !rev->ready) {
315             if (ngx_add_event(rev, NGX_READ_EVENT, 0) == NGX_ERROR) {
316                 return NGX_ERROR;
317             }
318
319             return NGX_OK;
320         }
321
322         if (rev->oneshot && !rev->ready) {
323             if (ngx_del_event(rev, NGX_READ_EVENT, 0) == NGX_ERROR) {
324                 return NGX_ERROR;
```

```
            }

            return NGX_OK;
        }
    }

    /* aio, iocp, rtsig */

    return NGX_OK;
}


ngx_int_t
ngx_handle_write_event(ngx_event_t *wev, size_t lowat)
{
    ngx_connection_t  *c;

    if (lowat) {
        c = wev->data;

        if (ngx_send_lowat(c, lowat) == NGX_ERROR) {
            return NGX_ERROR;
        }
    }

    if (ngx_event_flags & NGX_USE_CLEAR_EVENT) {

        /* kqueue, epoll */

        if (!wev->active && !wev->ready) {
            if (ngx_add_event(wev, NGX_WRITE_EVENT,
                              NGX_CLEAR_EVENT | (lowat ? NGX_LOWAT_EVENT : 0))
                == NGX_ERROR)
            {
                return NGX_ERROR;
            }
        }

        return NGX_OK;

    } else if (ngx_event_flags & NGX_USE_LEVEL_EVENT) {

        /* select, poll, /dev/poll */

        if (!wev->active && !wev->ready) {
            if (ngx_add_event(wev, NGX_WRITE_EVENT, NGX_LEVEL_EVENT)
                == NGX_ERROR)
            {
                return NGX_ERROR;
            }

            return NGX_OK;
        }

        if (wev->active && wev->ready) {
            if (ngx_del_event(wev, NGX_WRITE_EVENT, NGX_LEVEL_EVENT)
                == NGX_ERROR)
            {
                return NGX_ERROR;
            }

            return NGX_OK;
        }

    } else if (ngx_event_flags & NGX_USE_EVENTPORT_EVENT) {

        /* event ports */

        if (!wev->active && !wev->ready) {
            if (ngx_add_event(wev, NGX_WRITE_EVENT, 0) == NGX_ERROR) {
                return NGX_ERROR;
            }

            return NGX_OK;
        }

```

```
401         if (wev->oneshot && wev->ready) {
402             if (ngx_del_event(wev, NGX_WRITE_EVENT, 0) == NGX_ERROR) {
403                 return NGX_ERROR;
404             }
405
406             return NGX_OK;
407         }
408     }
409
410     /* aio, iocp, rtsig */
411
412     return NGX_OK;
413 }
414
415
416 static char *
417 ngx_event_init_conf(ngx_cycle_t *cycle, void *conf)
418 {
419     if (ngx_get_conf(cycle->conf_ctx, ngx_events_module) == NULL) {
420         ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
421                       "no \"events\" section in configuration");
422         return NGX_CONF_ERROR;
423     }
424
425     return NGX_CONF_OK;
426 }
427
428
429 static ngx_int_t
430 ngx_event_module_init(ngx_cycle_t *cycle)
431 {
432     void              ***cf;
433     u_char              *shared;
434     size_t               size, cl;
435     ngx_shm_t            shm;
436     ngx_time_t          *tp;
437     ngx_core_conf_t     *ccf;
438     ngx_event_conf_t    *ecf;
439
440     cf = ngx_get_conf(cycle->conf_ctx, ngx_events_module);
441     ecf = (*cf)[ngx_event_core_module.ctx_index];
442
443     if (!ngx_test_config && ngx_process <= NGX_PROCESS_MASTER) {
444         ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0,
445                       "using the \"%s\" event method", ecf->name);
446     }
447
448     ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
449
450     ngx_timer_resolution = ccf->timer_resolution;
451
452 #if !(NGX_WIN32)
453     {
454     ngx_int_t      limit;
455     struct rlimit  rlmt;
456
457     if (getrlimit(RLIMIT_NOFILE, &rlmt) == -1) {
458         ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
459                       "getrlimit(RLIMIT_NOFILE) failed, ignored");
460
461     } else {
462         if (ecf->connections > (ngx_uint_t) rlmt.rlim_cur
463             && (ccf->rlimit_nofile == NGX_CONF_UNSET
464                 || ecf->connections > (ngx_uint_t) ccf->rlimit_nofile))
465         {
466             limit = (ccf->rlimit_nofile == NGX_CONF_UNSET) ?
467                         (ngx_int_t) rlmt.rlim_cur : ccf->rlimit_nofile;
468
469             ngx_log_error(NGX_LOG_WARN, cycle->log, 0,
470                           "%ui worker_connections exceed "
471                           "open file resource limit: %i",
472                           ecf->connections, limit);
473         }
474     }
475     }
476 #endif /* !(NGX_WIN32) */
```

```
477
478
479     if (ccf->master == 0) {
480         return NGX_OK;
481     }
482
483     if (ngx_accept_mutex_ptr) {
484         return NGX_OK;
485     }
486
487
488     /* cl should be equal to or greater than cache line size */
489
490     cl = 128;
491
492     size = cl            /* ngx_accept_mutex */
493            + cl          /* ngx_connection_counter */
494            + cl;         /* ngx_temp_number */
495
496 #if (NGX_STAT_STUB)
497
498     size += cl           /* ngx_stat_accepted */
499            + cl          /* ngx_stat_handled */
500            + cl          /* ngx_stat_requests */
501            + cl          /* ngx_stat_active */
502            + cl          /* ngx_stat_reading */
503            + cl          /* ngx_stat_writing */
504            + cl;         /* ngx_stat_waiting */
505
506 #endif
507
508     shm.size = size;
509     shm.name.len = sizeof("nginx_shared_zone");
510     shm.name.data = (u_char *) "nginx_shared_zone";
511     shm.log = cycle->log;
512
513     if (ngx_shm_alloc(&shm) != NGX_OK) {
514         return NGX_ERROR;
515     }
516
517     shared = shm.addr;
518
519     ngx_accept_mutex_ptr = (ngx_atomic_t *) shared;
520     ngx_accept_mutex.spin = (ngx_uint_t) -1;
521
522     if (ngx_shmtx_create(&ngx_accept_mutex, (ngx_shmtx_sh_t *) shared,
523                          cycle->lock_file.data)
524         != NGX_OK)
525     {
526         return NGX_ERROR;
527     }
528
529     ngx_connection_counter = (ngx_atomic_t *) (shared + 1 * cl);
530
531     (void) ngx_atomic_cmp_set(ngx_connection_counter, 0, 1);
532
533     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
534                    "counter: %p, %d",
535                    ngx_connection_counter, *ngx_connection_counter);
536
537     ngx_temp_number = (ngx_atomic_t *) (shared + 2 * cl);
538
539     tp = ngx_timeofday();
540
541     ngx_random_number = (tp->msec << 16) + ngx_pid;
542
543 #if (NGX_STAT_STUB)
544
545     ngx_stat_accepted = (ngx_atomic_t *) (shared + 3 * cl);
546     ngx_stat_handled = (ngx_atomic_t *) (shared + 4 * cl);
547     ngx_stat_requests = (ngx_atomic_t *) (shared + 5 * cl);
548     ngx_stat_active = (ngx_atomic_t *) (shared + 6 * cl);
549     ngx_stat_reading = (ngx_atomic_t *) (shared + 7 * cl);
550     ngx_stat_writing = (ngx_atomic_t *) (shared + 8 * cl);
551     ngx_stat_waiting = (ngx_atomic_t *) (shared + 9 * cl);
552
```

```
#endif

    return NGX_OK;
}


#if !(NGX_WIN32)

static void
ngx_timer_signal_handler(int signo)
{
    ngx_event_timer_alarm = 1;

#if 1
    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ngx_cycle->log, 0, "timer signal");
#endif
}

#endif


static ngx_int_t
ngx_event_process_init(ngx_cycle_t *cycle)
{
    ngx_uint_t           m, i;
    ngx_event_t         *rev, *wev;
    ngx_listening_t     *ls;
    ngx_connection_t    *c, *next, *old;
    ngx_core_conf_t     *ccf;
    ngx_event_conf_t    *ecf;
    ngx_event_module_t  *module;

    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
    ecf = ngx_event_get_conf(cycle->conf_ctx, ngx_event_core_module);

    if (ccf->master && ccf->worker_processes > 1 && ecf->accept_mutex) {
        ngx_use_accept_mutex = 1;
        ngx_accept_mutex_held = 0;
        ngx_accept_mutex_delay = ecf->accept_mutex_delay;

    } else {
        ngx_use_accept_mutex = 0;
    }

#if (NGX_WIN32)

    /*
     * disable accept mutex on win32 as it may cause deadlock if
     * grabbed by a process which can't accept connections
     */

    ngx_use_accept_mutex = 0;

#endif

    ngx_queue_init(&ngx_posted_accept_events);
    ngx_queue_init(&ngx_posted_events);

    if (ngx_event_timer_init(cycle->log) == NGX_ERROR) {
        return NGX_ERROR;
    }

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_EVENT_MODULE) {
            continue;
        }

        if (ngx_modules[m]->ctx_index != ecf->use) {
            continue;
        }

        module = ngx_modules[m]->ctx;

        if (module->actions.init(cycle, ngx_timer_resolution) != NGX_OK) {
            /* fatal */
            exit(2);
```

```
        }

        break;
    }

#if !(NGX_WIN32)

    if (ngx_timer_resolution && !(ngx_event_flags & NGX_USE_TIMER_EVENT)) {
        struct sigaction  sa;
        struct itimerval  itv;

        ngx_memzero(&sa, sizeof(struct sigaction));
        sa.sa_handler = ngx_timer_signal_handler;
        sigemptyset(&sa.sa_mask);

        if (sigaction(SIGALRM, &sa, NULL) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "sigaction(SIGALRM) failed");
            return NGX_ERROR;
        }

        itv.it_interval.tv_sec = ngx_timer_resolution / 1000;
        itv.it_interval.tv_usec = (ngx_timer_resolution % 1000) * 1000;
        itv.it_value.tv_sec = ngx_timer_resolution / 1000;
        itv.it_value.tv_usec = (ngx_timer_resolution % 1000 ) * 1000;

        if (setitimer(ITIMER_REAL, &itv, NULL) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "setitimer() failed");
        }
    }

    if (ngx_event_flags & NGX_USE_FD_EVENT) {
        struct rlimit  rlmt;

        if (getrlimit(RLIMIT_NOFILE, &rlmt) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "getrlimit(RLIMIT_NOFILE) failed");
            return NGX_ERROR;
        }

        cycle->files_n = (ngx_uint_t) rlmt.rlim_cur;

        cycle->files = ngx_calloc(sizeof(ngx_connection_t *) * cycle->files_n,
                                  cycle->log);
        if (cycle->files == NULL) {
            return NGX_ERROR;
        }
    }

#endif

    cycle->connections =
        ngx_alloc(sizeof(ngx_connection_t) * cycle->connection_n, cycle->log);
    if (cycle->connections == NULL) {
        return NGX_ERROR;
    }

    c = cycle->connections;

    cycle->read_events = ngx_alloc(sizeof(ngx_event_t) * cycle->connection_n,
                                   cycle->log);
    if (cycle->read_events == NULL) {
        return NGX_ERROR;
    }

    rev = cycle->read_events;
    for (i = 0; i < cycle->connection_n; i++) {
        rev[i].closed = 1;
        rev[i].instance = 1;
    }

    cycle->write_events = ngx_alloc(sizeof(ngx_event_t) * cycle->connection_n,
                                    cycle->log);
    if (cycle->write_events == NULL) {
        return NGX_ERROR;
```

```
    }

    wev = cycle->write_events;
    for (i = 0; i < cycle->connection_n; i++) {
        wev[i].closed = 1;
    }

    i = cycle->connection_n;
    next = NULL;

    do {
        i--;

        c[i].data = next;
        c[i].read = &cycle->read_events[i];
        c[i].write = &cycle->write_events[i];
        c[i].fd = (ngx_socket_t) -1;

        next = &c[i];

#if (NGX_THREADS)
        c[i].lock = 0;
#endif
    } while (i);

    cycle->free_connections = next;
    cycle->free_connection_n = cycle->connection_n;

    /* for each listening socket */

    ls = cycle->listening.elts;
    for (i = 0; i < cycle->listening.nelts; i++) {

        c = ngx_get_connection(ls[i].fd, cycle->log);

        if (c == NULL) {
            return NGX_ERROR;
        }

        c->log = &ls[i].log;

        c->listening = &ls[i];
        ls[i].connection = c;

        rev = c->read;

        rev->log = c->log;
        rev->accept = 1;

#if (NGX_HAVE_DEFERRED_ACCEPT)
        rev->deferred_accept = ls[i].deferred_accept;
#endif

        if (!(ngx_event_flags & NGX_USE_IOCP_EVENT)) {
            if (ls[i].previous) {

                /*
                 * delete the old accept events that were bound to
                 * the old cycle read events array
                 */

                old = ls[i].previous->connection;

                if (ngx_del_event(old->read, NGX_READ_EVENT, NGX_CLOSE_EVENT)
                    == NGX_ERROR)
                {
                    return NGX_ERROR;
                }

                old->fd = (ngx_socket_t) -1;
            }
        }

#if (NGX_WIN32)

        if (ngx_event_flags & NGX_USE_IOCP_EVENT) {
```

```
781             ngx_iocp_conf_t  *iocpcf;
782
783             rev->handler = ngx_event_acceptex;
784
785             if (ngx_use_accept_mutex) {
786                 continue;
787             }
788
789             if (ngx_add_event(rev, 0, NGX_IOCP_ACCEPT) == NGX_ERROR) {
790                 return NGX_ERROR;
791             }
792
793             ls[i].log.handler = ngx_acceptex_log_error;
794
795             iocpcf = ngx_event_get_conf(cycle->conf_ctx, ngx_iocp_module);
796             if (ngx_event_post_acceptex(&ls[i], iocpcf->post_acceptex)
797                 == NGX_ERROR)
798             {
799                 return NGX_ERROR;
800             }
801
802         } else {
803             rev->handler = ngx_event_accept;
804
805             if (ngx_use_accept_mutex) {
806                 continue;
807             }
808
809             if (ngx_add_event(rev, NGX_READ_EVENT, 0) == NGX_ERROR) {
810                 return NGX_ERROR;
811             }
812         }
813
814 #else
815
816         rev->handler = ngx_event_accept;
817
818         if (ngx_use_accept_mutex) {
819             continue;
820         }
821
822         if (ngx_event_flags & NGX_USE_RTSIG_EVENT) {
823             if (ngx_add_conn(c) == NGX_ERROR) {
824                 return NGX_ERROR;
825             }
826
827         } else {
828             if (ngx_add_event(rev, NGX_READ_EVENT, 0) == NGX_ERROR) {
829                 return NGX_ERROR;
830             }
831         }
832
833 #endif
834
835     }
836
837     return NGX_OK;
838 }
839
840
841 ngx_int_t
842 ngx_send_lowat(ngx_connection_t *c, size_t lowat)
843 {
844     int  sndlowat;
845
846 #if (NGX_HAVE_LOWAT_EVENT)
847
848     if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
849         c->write->available = lowat;
850         return NGX_OK;
851     }
852
853 #endif
854
855     if (lowat == 0 || c->sndlowat) {
856         return NGX_OK;
```

```
    }

    sndlowat = (int) lowat;

    if (setsockopt(c->fd, SOL_SOCKET, SO_SNDLOWAT,
                   (const void *) &sndlowat, sizeof(int))
        == -1)
    {
        ngx_connection_error(c, ngx_socket_errno,
                             "setsockopt(SO_SNDLOWAT) failed");
        return NGX_ERROR;
    }

    c->sndlowat = 1;

    return NGX_OK;
}


static char *
ngx_events_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char                 *rv;
    void               ***ctx;
    ngx_uint_t            i;
    ngx_conf_t            pcf;
    ngx_event_module_t   *m;

    if (*(void **) conf) {
        return "is duplicate";
    }

    /* count the number of the event modules and set up their indices */

    ngx_event_max_module = 0;
    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_EVENT_MODULE) {
            continue;
        }

        ngx_modules[i]->ctx_index = ngx_event_max_module++;
    }

    ctx = ngx_pcalloc(cf->pool, sizeof(void *));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    *ctx = ngx_pcalloc(cf->pool, ngx_event_max_module * sizeof(void *));
    if (*ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    *(void **) conf = ctx;

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_EVENT_MODULE) {
            continue;
        }

        m = ngx_modules[i]->ctx;

        if (m->create_conf) {
            (*ctx)[ngx_modules[i]->ctx_index] = m->create_conf(cf->cycle);
            if ((*ctx)[ngx_modules[i]->ctx_index] == NULL) {
                return NGX_CONF_ERROR;
            }
        }
    }

    pcf = *cf;
    cf->ctx = ctx;
    cf->module_type = NGX_EVENT_MODULE;
    cf->cmd_type = NGX_EVENT_CONF;

    rv = ngx_conf_parse(cf, NULL);
```

```
933
934     *cf = pcf;
935
936     if (rv != NGX_CONF_OK)
937         return rv;
938
939     for (i = 0; ngx_modules[i]; i++) {
940         if (ngx_modules[i]->type != NGX_EVENT_MODULE) {
941             continue;
942         }
943
944         m = ngx_modules[i]->ctx;
945
946         if (m->init_conf) {
947             rv = m->init_conf(cf->cycle, (*ctx)[ngx_modules[i]->ctx_index]);
948             if (rv != NGX_CONF_OK) {
949                 return rv;
950             }
951         }
952     }
953
954     return NGX_CONF_OK;
955 }
956
957
958 static char *
959 ngx_event_connections(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
960 {
961     ngx_event_conf_t  *ecf = conf;
962
963     ngx_str_t  *value;
964
965     if (ecf->connections != NGX_CONF_UNSET_UINT) {
966         return "is duplicate";
967     }
968
969     if (ngx_strcmp(cmd->name.data, "connections") == 0) {
970         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
971                            "the \"connections\" directive is deprecated, "
972                            "use the \"worker_connections\" directive instead");
973     }
974
975     value = cf->args->elts;
976     ecf->connections = ngx_atoi(value[1].data, value[1].len);
977     if (ecf->connections == (ngx_uint_t) NGX_ERROR) {
978         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
979                            "invalid number \"%V\"", &value[1]);
980
981         return NGX_CONF_ERROR;
982     }
983
984     cf->cycle->connection_n = ecf->connections;
985
986     return NGX_CONF_OK;
987 }
988
989
990 static char *
991 ngx_event_use(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
992 {
993     ngx_event_conf_t  *ecf = conf;
994
995     ngx_int_t             m;
996     ngx_str_t            *value;
997     ngx_event_conf_t     *old_ecf;
998     ngx_event_module_t   *module;
999
1000     if (ecf->use != NGX_CONF_UNSET_UINT) {
1001         return "is duplicate";
1002     }
1003
1004     value = cf->args->elts;
1005
1006     if (cf->cycle->old_cycle->conf_ctx) {
1007         old_ecf = ngx_event_get_conf(cf->cycle->old_cycle->conf_ctx,
1008                                      ngx_event_core_module);
```

```
    } else {
        old_ecf = NULL;
    }


    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_EVENT_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;
        if (module->name->len == value[1].len) {
            if (ngx_strcmp(module->name->data, value[1].data) == 0) {
                ecf->use = ngx_modules[m]->ctx_index;
                ecf->name = module->name->data;

                if (ngx_process == NGX_PROCESS_SINGLE
                    && old_ecf
                    && old_ecf->use != ecf->use)
                {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "when the server runs without a master process "
                               "the \"%V\" event type must be the same as "
                               "in previous configuration - \"%s\" "
                               "and it cannot be changed on the fly, "
                               "to change it you need to stop server "
                               "and start it again",
                               &value[1], old_ecf->name);

                    return NGX_CONF_ERROR;
                }

                return NGX_CONF_OK;
            }
        }
    }

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "invalid event type \"%V\"", &value[1]);

    return NGX_CONF_ERROR;
}


static char *
ngx_event_debug_connection(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
#if (NGX_DEBUG)
    ngx_event_conf_t  *ecf = conf;

    ngx_int_t             rc;
    ngx_str_t            *value;
    ngx_url_t             u;
    ngx_cidr_t            c, *cidr;
    ngx_uint_t            i;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
#endif

    value = cf->args->elts;

#if (NGX_HAVE_UNIX_DOMAIN)

    if (ngx_strcmp(value[1].data, "unix:") == 0) {
        cidr = ngx_array_push(&ecf->debug_connection);
        if (cidr == NULL) {
            return NGX_CONF_ERROR;
        }

        cidr->family = AF_UNIX;
        return NGX_CONF_OK;
    }

#endif
```

```
    rc = ngx_ptocidr(&value[1], &c);

    if (rc != NGX_ERROR) {
        if (rc == NGX_DONE) {
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                               "low address bits of %V are meaningless",
                               &value[1]);
        }

        cidr = ngx_array_push(&ecf->debug_connection);
        if (cidr == NULL) {
            return NGX_CONF_ERROR;
        }

        *cidr = c;

        return NGX_CONF_OK;
    }

    ngx_memzero(&u, sizeof(ngx_url_t));
    u.host = value[1];

    if (ngx_inet_resolve_host(cf->pool, &u) != NGX_OK) {
        if (u.err) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "%s in debug_connection \"%V\"",
                               u.err, &u.host);
        }

        return NGX_CONF_ERROR;
    }

    cidr = ngx_array_push_n(&ecf->debug_connection, u.naddrs);
    if (cidr == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memzero(cidr, u.naddrs * sizeof(ngx_cidr_t));

    for (i = 0; i < u.naddrs; i++) {
        cidr[i].family = u.addrs[i].sockaddr->sa_family;

        switch (cidr[i].family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            sin6 = (struct sockaddr_in6 *) u.addrs[i].sockaddr;
            cidr[i].u.in6.addr = sin6->sin6_addr;
            ngx_memset(cidr[i].u.in6.mask.s6_addr, 0xff, 16);
            break;
#endif

        default: /* AF_INET */
            sin = (struct sockaddr_in *) u.addrs[i].sockaddr;
            cidr[i].u.in.addr = sin->sin_addr.s_addr;
            cidr[i].u.in.mask = 0xffffffff;
            break;
        }
    }

#else

    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                       "\"debug_connection\" is ignored, you need to rebuild "
                       "nginx using --with-debug option to enable it");

#endif

    return NGX_CONF_OK;
}


static void *
ngx_event_core_create_conf(ngx_cycle_t *cycle)
{
    ngx_event_conf_t  *ecf;
```

```
1161
1162        ecf = ngx_palloc(cycle->pool, sizeof(ngx_event_conf_t));
1163        if (ecf == NULL) {
1164            return NULL;
1165        }
1166
1167        ecf->connections = NGX_CONF_UNSET_UINT;
1168        ecf->use = NGX_CONF_UNSET_UINT;
1169        ecf->multi_accept = NGX_CONF_UNSET;
1170        ecf->accept_mutex = NGX_CONF_UNSET;
1171        ecf->accept_mutex_delay = NGX_CONF_UNSET_MSEC;
1172        ecf->name = (void *) NGX_CONF_UNSET;
1173
1174    #if (NGX_DEBUG)
1175
1176        if (ngx_array_init(&ecf->debug_connection, cycle->pool, 4,
1177                           sizeof(ngx_cidr_t)) == NGX_ERROR)
1178        {
1179            return NULL;
1180        }
1181
1182    #endif
1183
1184        return ecf;
1185    }
1186
1187
1188    static char *
1189    ngx_event_core_init_conf(ngx_cycle_t *cycle, void *conf)
1190    {
1191        ngx_event_conf_t  *ecf = conf;
1192
1193    #if (NGX_HAVE_EPOLL) && !(NGX_TEST_BUILD_EPOLL)
1194        int                  fd;
1195    #endif
1196    #if (NGX_HAVE_RTSIG)
1197        ngx_uint_t           rtsig;
1198        ngx_core_conf_t     *ccf;
1199    #endif
1200        ngx_int_t            i;
1201        ngx_module_t        *module;
1202        ngx_event_module_t  *event_module;
1203
1204        module = NULL;
1205
1206    #if (NGX_HAVE_EPOLL) && !(NGX_TEST_BUILD_EPOLL)
1207
1208        fd = epoll_create(100);
1209
1210        if (fd != -1) {
1211            (void) close(fd);
1212            module = &ngx_epoll_module;
1213
1214        } else if (ngx_errno != NGX_ENOSYS) {
1215            module = &ngx_epoll_module;
1216        }
1217
1218    #endif
1219
1220    #if (NGX_HAVE_RTSIG)
1221
1222        if (module == NULL) {
1223            module = &ngx_rtsig_module;
1224            rtsig = 1;
1225
1226        } else {
1227            rtsig = 0;
1228        }
1229
1230    #endif
1231
1232    #if (NGX_HAVE_DEVPOLL)
1233
1234        module = &ngx_devpoll_module;
1235
1236    #endif
```

```
1237
1238  #if (NGX_HAVE_KQUEUE)
1239
1240      module = &ngx_kqueue_module;
1241
1242  #endif
1243
1244  #if (NGX_HAVE_SELECT)
1245
1246      if (module == NULL) {
1247          module = &ngx_select_module;
1248      }
1249
1250  #endif
1251
1252      if (module == NULL) {
1253          for (i = 0; ngx_modules[i]; i++) {
1254
1255              if (ngx_modules[i]->type != NGX_EVENT_MODULE) {
1256                  continue;
1257              }
1258
1259              event_module = ngx_modules[i]->ctx;
1260
1261              if (ngx_strcmp(event_module->name->data, event_core_name.data) == 0)
1262              {
1263                  continue;
1264              }
1265
1266              module = ngx_modules[i];
1267              break;
1268          }
1269      }
1270
1271      if (module == NULL) {
1272          ngx_log_error(NGX_LOG_EMERG, cycle->log, 0, "no events module found");
1273          return NGX_CONF_ERROR;
1274      }
1275
1276      ngx_conf_init_uint_value(ecf->connections, DEFAULT_CONNECTIONS);
1277      cycle->connection_n = ecf->connections;
1278
1279      ngx_conf_init_uint_value(ecf->use, module->ctx_index);
1280
1281      event_module = module->ctx;
1282      ngx_conf_init_ptr_value(ecf->name, event_module->name->data);
1283
1284      ngx_conf_init_value(ecf->multi_accept, 0);
1285      ngx_conf_init_value(ecf->accept_mutex, 1);
1286      ngx_conf_init_msec_value(ecf->accept_mutex_delay, 500);
1287
1288
1289  #if (NGX_HAVE_RTSIG)
1290
1291      if (!rtsig) {
1292          return NGX_CONF_OK;
1293      }
1294
1295      if (ecf->accept_mutex) {
1296          return NGX_CONF_OK;
1297      }
1298
1299      ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
1300
1301      if (ccf->worker_processes == 0) {
1302          return NGX_CONF_OK;
1303      }
1304
1305      ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
1306                    "the \"rtsig\" method requires \"accept_mutex\" to be on");
1307
1308      return NGX_CONF_ERROR;
1309
1310  #else
1311
1312      return NGX_CONF_OK;
```

```
1313
1314 #endif
1315 }
```

One Level Up Top Level

src/event/modules/ngx_eventport_module.c - nginx-1.7.10

Global variables defined

- ep
- event_list
- event_timer
- eventport_name
- nevents
- ngx_eventport_commands
- ngx_eventport_module
- ngx_eventport_module_ctx

Data types defined

- clockid_t
- itimerspec
- itimerspec_t
- ngx_eventport_conf_t
- port_event_t
- port_notify
- port_notify_t
- timer_t

Functions defined

- ngx_eventport_add_event
- ngx_eventport_create_conf
- ngx_eventport_del_event
- ngx_eventport_done
- ngx_eventport_init
- ngx_eventport_init_conf
- ngx_eventport_process_events
- port_associate
- port_create
- port_dissociate
- port_getn

- timer_create
- timer_delete
- timer_settime

Macros defined

- CLOCK_REALTIME
- ETIME
- PORT_SOURCE_AIO
- PORT_SOURCE_ALERT
- PORT_SOURCE_FD
- PORT_SOURCE_MQ
- PORT_SOURCE_TIMER
- PORT_SOURCE_USER
- SIGEV_PORT
- uint_t
- ushort_t

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #if (NGX_TEST_BUILD_EVENTPORT)
14
15  #define ushort_t  u_short
16  #define uint_t    u_int
17
18  #ifndef CLOCK_REALTIME
19  #define CLOCK_REALTIME          0
20  typedef int     clockid_t;
21  typedef void *  timer_t;
22  #endif
23
24  /* Solaris declarations */
25
26  #define PORT_SOURCE_AIO         1
27  #define PORT_SOURCE_TIMER       2
28  #define PORT_SOURCE_USER        3
29  #define PORT_SOURCE_FD          4
30  #define PORT_SOURCE_ALERT       5
31  #define PORT_SOURCE_MQ          6
32
33  #ifndef ETIME
34  #define ETIME                   64
35  #endif
36
37  #define SIGEV_PORT              4
```

```
38
39  typedef struct {
40      int         portev_events;  /* event data is source specific */
41      ushort_t    portev_source;  /* event source */
42      ushort_t    portev_pad;     /* port internal use */
43      uintptr_t   portev_object;  /* source specific object */
44      void       *portev_user;    /* user cookie */
45  } port_event_t;
46
47  typedef struct  port_notify {
48      int         portnfy_port;   /* bind request(s) to port */
49      void       *portnfy_user;   /* user defined */
50  } port_notify_t;
51
52  #if (__FreeBSD_version < 700005)
53
54  typedef struct itimerspec {     /* definition per POSIX.4 */
55      struct timespec it_interval;/* timer period */
56      struct timespec it_value;   /* timer expiration */
57  } itimerspec_t;
58
59  #endif
60
61  int port_create(void);
62
63  int port_create(void)
64  {
65      return -1;
66  }
67
68
69  int port_associate(int port, int source, uintptr_t object, int events,
70      void *user);
71
72  int port_associate(int port, int source, uintptr_t object, int events,
73      void *user)
74  {
75      return -1;
76  }
77
78
79  int port_dissociate(int port, int source, uintptr_t object);
80
81  int port_dissociate(int port, int source, uintptr_t object)
82  {
83      return -1;
84  }
85
86
87  int port_getn(int port, port_event_t list[], uint_t max, uint_t *nget,
88      struct timespec *timeout);
89
90  int port_getn(int port, port_event_t list[], uint_t max, uint_t *nget,
91      struct timespec *timeout)
92  {
93      return -1;
94  }
95
96
97  int timer_create(clockid_t clock_id, struct sigevent *evp, timer_t *timerid);
98
99  int timer_create(clockid_t clock_id, struct sigevent *evp, timer_t *timerid)
100  {
101      return -1;
102  }
103
104
105  int timer_settime(timer_t timerid, int flags, const struct itimerspec *value,
106      struct itimerspec *ovalue);
107
108  int timer_settime(timer_t timerid, int flags, const struct itimerspec *value,
109      struct itimerspec *ovalue)
110  {
111      return -1;
112  }
113
```

```
114
115 int timer_delete(timer_t timerid);
116
117 int timer_delete(timer_t timerid)
118 {
119     return -1;
120 }
121
122 #endif
123
124
125 typedef struct {
126     ngx_uint_t  events;
127 } ngx_eventport_conf_t;
128
129
130 static ngx_int_t ngx_eventport_init(ngx_cycle_t *cycle, ngx_msec_t timer);
131 static void ngx_eventport_done(ngx_cycle_t *cycle);
132 static ngx_int_t ngx_eventport_add_event(ngx_event_t *ev, ngx_int_t event,
133     ngx_uint_t flags);
134 static ngx_int_t ngx_eventport_del_event(ngx_event_t *ev, ngx_int_t event,
135     ngx_uint_t flags);
136 static ngx_int_t ngx_eventport_process_events(ngx_cycle_t *cycle,
137     ngx_msec_t timer, ngx_uint_t flags);
138
139 static void *ngx_eventport_create_conf(ngx_cycle_t *cycle);
140 static char *ngx_eventport_init_conf(ngx_cycle_t *cycle, void *conf);
141
142 static int            ep = -1;
143 static port_event_t  *event_list;
144 static ngx_uint_t     nevents;
145 static timer_t        event_timer = (timer_t) -1;
146
147 static ngx_str_t      eventport_name = ngx_string("eventport");
148
149
150 static ngx_command_t  ngx_eventport_commands[] = {
151
152     { ngx_string("eventport_events"),
153       NGX_EVENT_CONF|NGX_CONF_TAKE1,
154       ngx_conf_set_num_slot,
155       0,
156       offsetof(ngx_eventport_conf_t, events),
157       NULL },
158
159       ngx_null_command
160 };
161
162
163 ngx_event_module_t  ngx_eventport_module_ctx = {
164     &eventport_name,
165     ngx_eventport_create_conf,             /* create configuration */
166     ngx_eventport_init_conf,               /* init configuration */
167
168     {
169         ngx_eventport_add_event,           /* add an event */
170         ngx_eventport_del_event,           /* delete an event */
171         ngx_eventport_add_event,           /* enable an event */
172         ngx_eventport_del_event,           /* disable an event */
173         NULL,                              /* add an connection */
174         NULL,                              /* delete an connection */
175         NULL,                              /* process the changes */
176         ngx_eventport_process_events,      /* process the events */
177         ngx_eventport_init,                /* init the events */
178         ngx_eventport_done,                /* done the events */
179     }
180
181 };
182
183 ngx_module_t  ngx_eventport_module = {
184     NGX_MODULE_V1,
185     &ngx_eventport_module_ctx,             /* module context */
186     ngx_eventport_commands,                /* module directives */
187     NGX_EVENT_MODULE,                      /* module type */
188     NULL,                                  /* init master */
189     NULL,                                  /* init module */
```

```
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_eventport_init(ngx_cycle_t *cycle, ngx_msec_t timer)
{
    port_notify_t          pn;
    struct itimerspec      its;
    struct sigevent        sev;
    ngx_eventport_conf_t  *epcf;

    epcf = ngx_event_get_conf(cycle->conf_ctx, ngx_eventport_module);

    if (ep == -1) {
        ep = port_create();

        if (ep == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "port_create() failed");
            return NGX_ERROR;
        }
    }

    if (nevents < epcf->events) {
        if (event_list) {
            ngx_free(event_list);
        }

        event_list = ngx_alloc(sizeof(port_event_t) * epcf->events,
                               cycle->log);
        if (event_list == NULL) {
            return NGX_ERROR;
        }
    }

    ngx_event_flags = NGX_USE_EVENTPORT_EVENT;

    if (timer) {
        ngx_memzero(&pn, sizeof(port_notify_t));
        pn.portnfy_port = ep;

        ngx_memzero(&sev, sizeof(struct sigevent));
        sev.sigev_notify = SIGEV_PORT;
#if !(NGX_TEST_BUILD_EVENTPORT)
        sev.sigev_value.sival_ptr = &pn;
#endif

        if (timer_create(CLOCK_REALTIME, &sev, &event_timer) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "timer_create() failed");
            return NGX_ERROR;
        }

        its.it_interval.tv_sec = timer / 1000;
        its.it_interval.tv_nsec = (timer % 1000) * 1000000;
        its.it_value.tv_sec = timer / 1000;
        its.it_value.tv_nsec = (timer % 1000) * 1000000;

        if (timer_settime(event_timer, 0, &its, NULL) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "timer_settime() failed");
            return NGX_ERROR;
        }

        ngx_event_flags |= NGX_USE_TIMER_EVENT;
    }

    nevents = epcf->events;

    ngx_io = ngx_os_io;
```

```
266
267     ngx_event_actions = ngx_eventport_module_ctx.actions;
268
269     return NGX_OK;
270 }
271
272
273 static void
274 ngx_eventport_done(ngx_cycle_t *cycle)
275 {
276     if (event_timer != (timer_t) -1) {
277         if (timer_delete(event_timer) == -1) {
278             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
279                           "timer_delete() failed");
280         }
281
282         event_timer = (timer_t) -1;
283     }
284
285     if (close(ep) == -1) {
286         ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
287                       "close() event port failed");
288     }
289
290     ep = -1;
291
292     ngx_free(event_list);
293
294     event_list = NULL;
295     nevents = 0;
296 }
297
298
299 static ngx_int_t
300 ngx_eventport_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
301 {
302     ngx_int_t          events, prev;
303     ngx_event_t       *e;
304     ngx_connection_t  *c;
305
306     c = ev->data;
307
308     events = event;
309
310     if (event == NGX_READ_EVENT) {
311         e = c->write;
312         prev = POLLOUT;
313 #if (NGX_READ_EVENT != POLLIN)
314         events = POLLIN;
315 #endif
316
317     } else {
318         e = c->read;
319         prev = POLLIN;
320 #if (NGX_WRITE_EVENT != POLLOUT)
321         events = POLLOUT;
322 #endif
323     }
324
325     if (e->oneshot) {
326         events |= prev;
327     }
328
329     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
330                    "eventport add event: fd:%d ev:%04Xi", c->fd, events);
331
332     if (port_associate(ep, PORT_SOURCE_FD, c->fd, events,
333                        (void *) ((uintptr_t) ev | ev->instance))
334         == -1)
335     {
336         ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
337                       "port_associate() failed");
338         return NGX_ERROR;
339     }
340
341     ev->active = 1;
```

```
    ev->oneshot = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_eventport_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_event_t       *e;
    ngx_connection_t  *c;

    /*
     * when the file descriptor is closed, the event port automatically
     * dissociates it from the port, so we do not need to dissociate explicitly
     * the event before the closing the file descriptor
     */

    if (flags & NGX_CLOSE_EVENT) {
        ev->active = 0;
        ev->oneshot = 0;
        return NGX_OK;
    }

    c = ev->data;

    if (event == NGX_READ_EVENT) {
        e = c->write;
        event = POLLOUT;

    } else {
        e = c->read;
        event = POLLIN;
    }

    if (e->oneshot) {
        ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                       "eventport change event: fd:%d ev:%04Xi", c->fd, event);

        if (port_associate(ep, PORT_SOURCE_FD, c->fd, event,
                           (void *) ((uintptr_t) ev | ev->instance))
            == -1)
        {
            ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
                          "port_associate() failed");
            return NGX_ERROR;
        }

    } else {
        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                       "eventport del event: fd:%d", c->fd);

        if (port_dissociate(ep, PORT_SOURCE_FD, c->fd) == -1) {
            ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
                          "port_dissociate() failed");
            return NGX_ERROR;
        }
    }

    ev->active = 0;
    ev->oneshot = 0;

    return NGX_OK;
}


ngx_int_t
ngx_eventport_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
    ngx_uint_t flags)
{
    int                n, revents;
    u_int              events;
    ngx_err_t          err;
    ngx_int_t          instance;
    ngx_uint_t         i, level;
    ngx_event_t       *ev, *rev, *wev;
```

```
418     ngx_queue_t        *queue;
419     ngx_connection_t   *c;
420     struct timespec     ts, *tp;
421
422     if (timer == NGX_TIMER_INFINITE) {
423         tp = NULL;
424
425     } else {
426         ts.tv_sec = timer / 1000;
427         ts.tv_nsec = (timer % 1000) * 1000000;
428         tp = &ts;
429     }
430
431     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
432                    "eventport timer: %M", timer);
433
434     events = 1;
435
436     n = port_getn(ep, event_list, (u_int) nevents, &events, tp);
437
438     err = ngx_errno;
439
440     if (flags & NGX_UPDATE_TIME) {
441         ngx_time_update();
442     }
443
444     if (n == -1) {
445         if (err == ETIME) {
446             if (timer != NGX_TIMER_INFINITE) {
447                 return NGX_OK;
448             }
449
450             ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
451                           "port_getn() returned no events without timeout");
452             return NGX_ERROR;
453         }
454
455         level = (err == NGX_EINTR) ? NGX_LOG_INFO : NGX_LOG_ALERT;
456         ngx_log_error(level, cycle->log, err, "port_getn() failed");
457         return NGX_ERROR;
458     }
459
460     if (events == 0) {
461         if (timer != NGX_TIMER_INFINITE) {
462             return NGX_OK;
463         }
464
465         ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
466                       "port_getn() returned no events without timeout");
467         return NGX_ERROR;
468     }
469
470     for (i = 0; i < events; i++) {
471
472         if (event_list[i].portev_source == PORT_SOURCE_TIMER) {
473             ngx_time_update();
474             continue;
475         }
476
477         ev = event_list[i].portev_user;
478
479         switch (event_list[i].portev_source) {
480
481         case PORT_SOURCE_FD:
482
483             instance = (uintptr_t) ev & 1;
484             ev = (ngx_event_t *) ((uintptr_t) ev & (uintptr_t) ~1);
485
486             if (ev->closed || ev->instance != instance) {
487
488                 /*
489                  * the stale event from a file descriptor
490                  * that was just closed in this iteration
491                  */
492
493                 ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
```

```
                        "eventport: stale event %p", ev);
            continue;
        }

        revents = event_list[i].portev_events;

        ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "eventport: fd:%d, ev:%04Xd",
                       event_list[i].portev_object, revents);

        if (revents & (POLLERR|POLLHUP|POLLNVAL)) {
            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "port_getn() error fd:%d ev:%04Xd",
                           event_list[i].portev_object, revents);
        }

        if (revents & ~(POLLIN|POLLOUT|POLLERR|POLLHUP|POLLNVAL)) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                          "strange port_getn() events fd:%d ev:%04Xd",
                          event_list[i].portev_object, revents);
        }

        if ((revents & (POLLERR|POLLHUP|POLLNVAL))
             && (revents & (POLLIN|POLLOUT)) == 0)
        {
            /*
             * if the error events were returned without POLLIN or POLLOUT,
             * then add these flags to handle the events at least in one
             * active handler
             */

            revents |= POLLIN|POLLOUT;
        }

        c = ev->data;
        rev = c->read;
        wev = c->write;

        rev->active = 0;
        wev->active = 0;

        if (revents & POLLIN) {
            rev->ready = 1;

            if (flags & NGX_POST_EVENTS) {
                queue = rev->accept ? &ngx_posted_accept_events
                                    : &ngx_posted_events;

                ngx_post_event(rev, queue);

            } else {
                rev->handler(rev);

                if (ev->closed || ev->instance != instance) {
                    continue;
                }
            }

            if (rev->accept) {
                if (ngx_use_accept_mutex) {
                    ngx_accept_events = 1;
                    continue;
                }

                if (port_associate(ep, PORT_SOURCE_FD, c->fd, POLLIN,
                                   (void *) ((uintptr_t) ev | ev->instance))
                    == -1)
                {
                    ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
                                  "port_associate() failed");
                    return NGX_ERROR;
                }
            }
        }

        if (revents & POLLOUT) {
```

```
570                 wev->ready = 1;
571
572                 if (flags & NGX_POST_EVENTS) {
573                     ngx_post_event(wev, &ngx_posted_events);
574
575                 } else {
576                     wev->handler(wev);
577                 }
578             }
579
580             continue;
581
582         default:
583             ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
584                           "unexpected even_port object %d",
585                           event_list[i].portev_object);
586             continue;
587         }
588     }
589
590     return NGX_OK;
591 }
592
593
594 static void *
595 ngx_eventport_create_conf(ngx_cycle_t *cycle)
596 {
597     ngx_eventport_conf_t  *epcf;
598
599     epcf = ngx_palloc(cycle->pool, sizeof(ngx_eventport_conf_t));
600     if (epcf == NULL) {
601         return NULL;
602     }
603
604     epcf->events = NGX_CONF_UNSET;
605
606     return epcf;
607 }
608
609
610 static char *
611 ngx_eventport_init_conf(ngx_cycle_t *cycle, void *conf)
612 {
613     ngx_eventport_conf_t *epcf = conf;
614
615     ngx_conf_init_uint_value(epcf->events, 32);
616
617     return NGX_CONF_OK;
618 }
```

One Level Up Top Level

src/os/unix/ngx_time.h - nginx-1.7.10

Data types defined

- ngx_msec_int_t
- ngx_msec_t
- ngx_tm_t

Macros defined

- _NGX_TIME_H_INCLUDED_
- ngx_gettimeofday
- ngx_msleep
- ngx_sleep
- ngx_timezone
- ngx_timezone
- ngx_tm_gmtoff
- ngx_tm_hour
- ngx_tm_hour_t
- ngx_tm_isdst
- ngx_tm_mday
- ngx_tm_mday_t
- ngx_tm_min
- ngx_tm_min_t
- ngx_tm_mon
- ngx_tm_mon_t
- ngx_tm_sec
- ngx_tm_sec_t
- ngx_tm_wday
- ngx_tm_wday_t
- ngx_tm_year
- ngx_tm_year_t
- ngx_tm_zone

Source code

```
1
2 /*
```

```
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#ifndef _NGX_TIME_H_INCLUDED_
#define _NGX_TIME_H_INCLUDED_


#include <ngx_config.h>
#include <ngx_core.h>


typedef ngx_rbtree_key_t      ngx_msec_t;
typedef ngx_rbtree_key_int_t  ngx_msec_int_t;

typedef struct tm             ngx_tm_t;

#define ngx_tm_sec            tm_sec
#define ngx_tm_min            tm_min
#define ngx_tm_hour           tm_hour
#define ngx_tm_mday           tm_mday
#define ngx_tm_mon            tm_mon
#define ngx_tm_year           tm_year
#define ngx_tm_wday           tm_wday
#define ngx_tm_isdst          tm_isdst

#define ngx_tm_sec_t          int
#define ngx_tm_min_t          int
#define ngx_tm_hour_t         int
#define ngx_tm_mday_t         int
#define ngx_tm_mon_t          int
#define ngx_tm_year_t         int
#define ngx_tm_wday_t         int


#if (NGX_HAVE_GMTOFF)
#define ngx_tm_gmtoff         tm_gmtoff
#define ngx_tm_zone           tm_zone
#endif


#if (NGX_SOLARIS)

#define ngx_timezone(isdst) (- (isdst ? altzone : timezone) / 60)

#else

#define ngx_timezone(isdst) (- (isdst ? timezone + 3600 : timezone) / 60)

#endif


void ngx_timezone_update(void);
void ngx_localtime(time_t s, ngx_tm_t *tm);
void ngx_libc_localtime(time_t s, struct tm *tm);
void ngx_libc_gmtime(time_t s, struct tm *tm);

#define ngx_gettimeofday(tp)  (void) gettimeofday(tp, NULL);
#define ngx_msleep(ms)        (void) usleep(ms * 1000)
#define ngx_sleep(s)          (void) sleep(s)


#endif /* _NGX_TIME_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_rbtree.h - nginx-1.7.10

Data types defined

- ngx_rbtree_insert_pt
- ngx_rbtree_key_int_t
- ngx_rbtree_key_t
- ngx_rbtree_node_s
- ngx_rbtree_node_t
- ngx_rbtree_s
- ngx_rbtree_t

Functions defined

- ngx_rbtree_min

Macros defined

- _NGX_RBTREE_H_INCLUDED_
- ngx_rbt_black
- ngx_rbt_copy_color
- ngx_rbt_is_black
- ngx_rbt_is_red
- ngx_rbt_red
- ngx_rbtree_init
- ngx_rbtree_sentinel_init

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_RBTREE_H_INCLUDED_
9  #define _NGX_RBTREE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef ngx_uint_t  ngx_rbtree_key_t;
17  typedef ngx_int_t   ngx_rbtree_key_int_t;
18
19
20  typedef struct ngx_rbtree_node_s  ngx_rbtree_node_t;
21
```

```
22  struct ngx_rbtree_node_s {
23      ngx_rbtree_key_t       key;
24      ngx_rbtree_node_t     *left;
25      ngx_rbtree_node_t     *right;
26      ngx_rbtree_node_t     *parent;
27      u_char                 color;
28      u_char                 data;
29  };
30
31
32  typedef struct ngx_rbtree_s  ngx_rbtree_t;
33
34  typedef void (*ngx_rbtree_insert_pt) (ngx_rbtree_node_t *root,
35      ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
36
37  struct ngx_rbtree_s {
38      ngx_rbtree_node_t     *root;
39      ngx_rbtree_node_t     *sentinel;
40      ngx_rbtree_insert_pt   insert;
41  };
42
43
44  #define ngx_rbtree_init(tree, s, i)                                           \
45      ngx_rbtree_sentinel_init(s);                                              \
46      (tree)->root = s;                                                         \
47      (tree)->sentinel = s;                                                     \
48      (tree)->insert = i
49
50
51  void ngx_rbtree_insert(ngx_rbtree_t *tree, ngx_rbtree_node_t *node);
52  void ngx_rbtree_delete(ngx_rbtree_t *tree, ngx_rbtree_node_t *node);
53  void ngx_rbtree_insert_value(ngx_rbtree_node_t *root, ngx_rbtree_node_t *node,
54      ngx_rbtree_node_t *sentinel);
55  void ngx_rbtree_insert_timer_value(ngx_rbtree_node_t *root,
56      ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
57
58
59  #define ngx_rbt_red(node)               ((node)->color = 1)
60  #define ngx_rbt_black(node)             ((node)->color = 0)
61  #define ngx_rbt_is_red(node)            ((node)->color)
62  #define ngx_rbt_is_black(node)          (!ngx_rbt_is_red(node))
63  #define ngx_rbt_copy_color(n1, n2)      (n1->color = n2->color)
64
65
66  /* a sentinel must be black */
67
68  #define ngx_rbtree_sentinel_init(node)  ngx_rbt_black(node)
69
70
71  static ngx_inline ngx_rbtree_node_t *
72  ngx_rbtree_min(ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
73  {
74      while (node->left != sentinel) {
75          node = node->left;
76      }
77
78      return node;
79  }
80
81
82  #endif /* _NGX_RBTREE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_rbtree.c - nginx-1.7.10

Functions defined

- ngx_rbtree_delete
- ngx_rbtree_insert
- ngx_rbtree_insert_timer_value
- ngx_rbtree_insert_value
- ngx_rbtree_left_rotate
- ngx_rbtree_right_rotate

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13  * The red-black tree code is based on the algorithm described in
14  * the "Introduction to Algorithms" by Cormen, Leiserson and Rivest.
15  */
16
17
18  static ngx_inline void ngx_rbtree_left_rotate(ngx_rbtree_node_t **root,
19      ngx_rbtree_node_t *sentinel, ngx_rbtree_node_t *node);
20  static ngx_inline void ngx_rbtree_right_rotate(ngx_rbtree_node_t **root,
21      ngx_rbtree_node_t *sentinel, ngx_rbtree_node_t *node);
22
23
24  void
25  ngx_rbtree_insert(ngx_rbtree_t *tree, ngx_rbtree_node_t *node)
26  {
27      ngx_rbtree_node_t  **root, *temp, *sentinel;
28
29      /* a binary tree insert */
30
31      root = (ngx_rbtree_node_t **) &tree->root;
32      sentinel = tree->sentinel;
33
34      if (*root == sentinel) {
35          node->parent = NULL;
36          node->left = sentinel;
37          node->right = sentinel;
38          ngx_rbt_black(node);
39          *root = node;
40
41          return;
42      }
43
44      tree->insert(*root, node, sentinel);
45
46      /* re-balance tree */
47
48      while (node != *root && ngx_rbt_is_red(node->parent)) {
49
50          if (node->parent == node->parent->parent->left) {
51              temp = node->parent->parent->right;
```

```
52
53              if (ngx_rbt_is_red(temp)) {
54                  ngx_rbt_black(node->parent);
55                  ngx_rbt_black(temp);
56                  ngx_rbt_red(node->parent->parent);
57                  node = node->parent->parent;
58
59              } else {
60                  if (node == node->parent->right) {
61                      node = node->parent;
62                      ngx_rbtree_left_rotate(root, sentinel, node);
63                  }
64
65                  ngx_rbt_black(node->parent);
66                  ngx_rbt_red(node->parent->parent);
67                  ngx_rbtree_right_rotate(root, sentinel, node->parent->parent);
68              }
69
70          } else {
71              temp = node->parent->parent->left;
72
73              if (ngx_rbt_is_red(temp)) {
74                  ngx_rbt_black(node->parent);
75                  ngx_rbt_black(temp);
76                  ngx_rbt_red(node->parent->parent);
77                  node = node->parent->parent;
78
79              } else {
80                  if (node == node->parent->left) {
81                      node = node->parent;
82                      ngx_rbtree_right_rotate(root, sentinel, node);
83                  }
84
85                  ngx_rbt_black(node->parent);
86                  ngx_rbt_red(node->parent->parent);
87                  ngx_rbtree_left_rotate(root, sentinel, node->parent->parent);
88              }
89          }
90      }
91
92      ngx_rbt_black(*root);
93  }
94
95
96  void
97  ngx_rbtree_insert_value(ngx_rbtree_node_t *temp, ngx_rbtree_node_t *node,
98      ngx_rbtree_node_t *sentinel)
99  {
100      ngx_rbtree_node_t  **p;
101
102      for ( ;; ) {
103
104          p = (node->key < temp->key) ? &temp->left : &temp->right;
105
106          if (*p == sentinel) {
107              break;
108          }
109
110          temp = *p;
111      }
112
113      *p = node;
114      node->parent = temp;
115      node->left = sentinel;
116      node->right = sentinel;
117      ngx_rbt_red(node);
118  }
119
120
121  void
122  ngx_rbtree_insert_timer_value(ngx_rbtree_node_t *temp, ngx_rbtree_node_t *node,
123      ngx_rbtree_node_t *sentinel)
124  {
125      ngx_rbtree_node_t  **p;
126
127      for ( ;; ) {
```

```
128
129         /*
130          * Timer values
131          * 1) are spread in small range, usually several minutes,
132          * 2) and overflow each 49 days, if milliseconds are stored in 32 bits.
133          * The comparison takes into account that overflow.
134          */
135
136         /*  node->key < temp->key */
137
138         p = ((ngx_rbtree_key_int_t) (node->key - temp->key) < 0)
139             ? &temp->left : &temp->right;
140
141         if (*p == sentinel) {
142             break;
143         }
144
145         temp = *p;
146     }
147
148     *p = node;
149     node->parent = temp;
150     node->left = sentinel;
151     node->right = sentinel;
152     ngx_rbt_red(node);
153 }
154
155
156 void
157 ngx_rbtree_delete(ngx_rbtree_t *tree, ngx_rbtree_node_t *node)
158 {
159     ngx_uint_t           red;
160     ngx_rbtree_node_t  **root, *sentinel, *subst, *temp, *w;
161
162     /* a binary tree delete */
163
164     root = (ngx_rbtree_node_t **) &tree->root;
165     sentinel = tree->sentinel;
166
167     if (node->left == sentinel) {
168         temp = node->right;
169         subst = node;
170
171     } else if (node->right == sentinel) {
172         temp = node->left;
173         subst = node;
174
175     } else {
176         subst = ngx_rbtree_min(node->right, sentinel);
177
178         if (subst->left != sentinel) {
179             temp = subst->left;
180         } else {
181             temp = subst->right;
182         }
183     }
184
185     if (subst == *root) {
186         *root = temp;
187         ngx_rbt_black(temp);
188
189         /* DEBUG stuff */
190         node->left = NULL;
191         node->right = NULL;
192         node->parent = NULL;
193         node->key = 0;
194
195         return;
196     }
197
198     red = ngx_rbt_is_red(subst);
199
200     if (subst == subst->parent->left) {
201         subst->parent->left = temp;
202
203     } else {
```

```
204         subst->parent->right = temp;
205     }
206
207     if (subst == node) {
208
209         temp->parent = subst->parent;
210
211     } else {
212
213         if (subst->parent == node) {
214             temp->parent = subst;
215
216         } else {
217             temp->parent = subst->parent;
218         }
219
220         subst->left = node->left;
221         subst->right = node->right;
222         subst->parent = node->parent;
223         ngx_rbt_copy_color(subst, node);
224
225         if (node == *root) {
226             *root = subst;
227
228         } else {
229             if (node == node->parent->left) {
230                 node->parent->left = subst;
231             } else {
232                 node->parent->right = subst;
233             }
234         }
235
236         if (subst->left != sentinel) {
237             subst->left->parent = subst;
238         }
239
240         if (subst->right != sentinel) {
241             subst->right->parent = subst;
242         }
243     }
244
245     /* DEBUG stuff */
246     node->left = NULL;
247     node->right = NULL;
248     node->parent = NULL;
249     node->key = 0;
250
251     if (red) {
252         return;
253     }
254
255     /* a delete fixup */
256
257     while (temp != *root && ngx_rbt_is_black(temp)) {
258
259         if (temp == temp->parent->left) {
260             w = temp->parent->right;
261
262             if (ngx_rbt_is_red(w)) {
263                 ngx_rbt_black(w);
264                 ngx_rbt_red(temp->parent);
265                 ngx_rbtree_left_rotate(root, sentinel, temp->parent);
266                 w = temp->parent->right;
267             }
268
269             if (ngx_rbt_is_black(w->left) && ngx_rbt_is_black(w->right)) {
270                 ngx_rbt_red(w);
271                 temp = temp->parent;
272
273             } else {
274                 if (ngx_rbt_is_black(w->right)) {
275                     ngx_rbt_black(w->left);
276                     ngx_rbt_red(w);
277                     ngx_rbtree_right_rotate(root, sentinel, w);
278                     w = temp->parent->right;
279                 }
```

```
                ngx_rbt_copy_color(w, temp->parent);
                ngx_rbt_black(temp->parent);
                ngx_rbt_black(w->right);
                ngx_rbtree_left_rotate(root, sentinel, temp->parent);
                temp = *root;
            }

        } else {
            w = temp->parent->left;

            if (ngx_rbt_is_red(w)) {
                ngx_rbt_black(w);
                ngx_rbt_red(temp->parent);
                ngx_rbtree_right_rotate(root, sentinel, temp->parent);
                w = temp->parent->left;
            }

            if (ngx_rbt_is_black(w->left) && ngx_rbt_is_black(w->right)) {
                ngx_rbt_red(w);
                temp = temp->parent;

            } else {
                if (ngx_rbt_is_black(w->left)) {
                    ngx_rbt_black(w->right);
                    ngx_rbt_red(w);
                    ngx_rbtree_left_rotate(root, sentinel, w);
                    w = temp->parent->left;
                }

                ngx_rbt_copy_color(w, temp->parent);
                ngx_rbt_black(temp->parent);
                ngx_rbt_black(w->left);
                ngx_rbtree_right_rotate(root, sentinel, temp->parent);
                temp = *root;
            }
        }
    }

    ngx_rbt_black(temp);
}


static ngx_inline void
ngx_rbtree_left_rotate(ngx_rbtree_node_t **root, ngx_rbtree_node_t *sentinel,
    ngx_rbtree_node_t *node)
{
    ngx_rbtree_node_t  *temp;

    temp = node->right;
    node->right = temp->left;

    if (temp->left != sentinel) {
        temp->left->parent = node;
    }

    temp->parent = node->parent;

    if (node == *root) {
        *root = temp;

    } else if (node == node->parent->left) {
        node->parent->left = temp;

    } else {
        node->parent->right = temp;
    }

    temp->left = node;
    node->parent = temp;
}


static ngx_inline void
ngx_rbtree_right_rotate(ngx_rbtree_node_t **root, ngx_rbtree_node_t *sentinel,
    ngx_rbtree_node_t *node)
```

```
{
    ngx_rbtree_node_t  *temp;

    temp = node->left;
    node->left = temp->right;

    if (temp->right != sentinel) {
        temp->right->parent = node;
    }

    temp->parent = node->parent;

    if (node == *root) {
        *root = temp;

    } else if (node == node->parent->right) {
        node->parent->right = temp;

    } else {
        node->parent->left = temp;
    }

    temp->right = node;
    node->parent = temp;
}
```

One Level Up Top Level

src/event/modules/ngx_rtsig_module.c - nginx-1.7.10

Global variables defined

- ngx_linux_rtsig_max
- ngx_overflow_threshold_bounds
- ngx_rtsig_commands
- ngx_rtsig_module
- ngx_rtsig_module_ctx
- overflow
- overflow_current
- overflow_list
- rtsig_name
- set

Data types defined

- ngx_rtsig_conf_t

Functions defined

- ngx_check_ngx_overflow_threshold_bounds
- ngx_rtsig_add_connection
- ngx_rtsig_create_conf
- ngx_rtsig_del_connection
- ngx_rtsig_done
- ngx_rtsig_init
- ngx_rtsig_init_conf
- ngx_rtsig_process_events
- ngx_rtsig_process_overflow
- sigtimedwait

Macros defined

- F_SETSIG
- KERN_RTSIGMAX
- KERN_RTSIGNR
- SIGRTMIN

- SIGRTMIN
- si_fd
- si_fd
- si_fd

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #if (NGX_TEST_BUILD_RTSIG)
14
15  #if (NGX_DARWIN)
16
17  #define SIGRTMIN       33
18  #define si_fd          __pad[0]
19
20  #else
21
22  #ifdef  SIGRTMIN
23  #define si_fd          _reason.__spare__.__spare2__[0]
24  #else
25  #define SIGRTMIN       33
26  #define si_fd          __spare__[0]
27  #endif
28
29  #endif
30
31  #define F_SETSIG       10
32  #define KERN_RTSIGNR   30
33  #define KERN_RTSIGMAX  31
34
35  int sigtimedwait(const sigset_t *set, siginfo_t *info,
36                   const struct timespec *timeout);
37
38  int sigtimedwait(const sigset_t *set, siginfo_t *info,
39                   const struct timespec *timeout)
40  {
41      return -1;
42  }
43
44  int ngx_linux_rtsig_max;
45
46  #endif
47
48
49  typedef struct {
50      ngx_uint_t  signo;
51      ngx_uint_t  overflow_events;
52      ngx_uint_t  overflow_test;
53      ngx_uint_t  overflow_threshold;
54  } ngx_rtsig_conf_t;
55
56
57  extern ngx_event_module_t  ngx_poll_module_ctx;
58
59  static ngx_int_t ngx_rtsig_init(ngx_cycle_t *cycle, ngx_msec_t timer);
60  static void ngx_rtsig_done(ngx_cycle_t *cycle);
61  static ngx_int_t ngx_rtsig_add_connection(ngx_connection_t *c);
62  static ngx_int_t ngx_rtsig_del_connection(ngx_connection_t *c,
63      ngx_uint_t flags);
```

```
64  static ngx_int_t ngx_rtsig_process_events(ngx_cycle_t *cycle,
65      ngx_msec_t timer, ngx_uint_t flags);
66  static ngx_int_t ngx_rtsig_process_overflow(ngx_cycle_t *cycle,
67      ngx_msec_t timer, ngx_uint_t flags);
68
69  static void *ngx_rtsig_create_conf(ngx_cycle_t *cycle);
70  static char *ngx_rtsig_init_conf(ngx_cycle_t *cycle, void *conf);
71  static char *ngx_check_ngx_overflow_threshold_bounds(ngx_conf_t *cf,
72      void *post, void *data);
73
74
75  static sigset_t        set;
76  static ngx_uint_t      overflow, overflow_current;
77  static struct pollfd  *overflow_list;
78
79
80  static ngx_str_t      rtsig_name = ngx_string("rtsig");
81
82  static ngx_conf_num_bounds_t  ngx_overflow_threshold_bounds = {
83      ngx_check_ngx_overflow_threshold_bounds, 2, 10
84  };
85
86
87  static ngx_command_t  ngx_rtsig_commands[] = {
88
89      { ngx_string("rtsig_signo"),
90        NGX_EVENT_CONF|NGX_CONF_TAKE1,
91        ngx_conf_set_num_slot,
92        0,
93        offsetof(ngx_rtsig_conf_t, signo),
94        NULL },
95
96      { ngx_string("rtsig_overflow_events"),
97        NGX_EVENT_CONF|NGX_CONF_TAKE1,
98        ngx_conf_set_num_slot,
99        0,
100        offsetof(ngx_rtsig_conf_t, overflow_events),
101        NULL },
102
103      { ngx_string("rtsig_overflow_test"),
104        NGX_EVENT_CONF|NGX_CONF_TAKE1,
105        ngx_conf_set_num_slot,
106        0,
107        offsetof(ngx_rtsig_conf_t, overflow_test),
108        NULL },
109
110      { ngx_string("rtsig_overflow_threshold"),
111        NGX_EVENT_CONF|NGX_CONF_TAKE1,
112        ngx_conf_set_num_slot,
113        0,
114        offsetof(ngx_rtsig_conf_t, overflow_threshold),
115        &ngx_overflow_threshold_bounds },
116
117        ngx_null_command
118  };
119
120
121  ngx_event_module_t  ngx_rtsig_module_ctx = {
122      &rtsig_name,
123      ngx_rtsig_create_conf,               /* create configuration */
124      ngx_rtsig_init_conf,                 /* init configuration */
125
126      {
127          NULL,                            /* add an event */
128          NULL,                            /* delete an event */
129          NULL,                            /* enable an event */
130          NULL,                            /* disable an event */
131          ngx_rtsig_add_connection,        /* add an connection */
132          ngx_rtsig_del_connection,        /* delete an connection */
133          NULL,                            /* process the changes */
134          ngx_rtsig_process_events,        /* process the events */
135          ngx_rtsig_init,                  /* init the events */
136          ngx_rtsig_done,                  /* done the events */
137      }
138
139  };
```

```
140
141 ngx_module_t  ngx_rtsig_module = {
142     NGX_MODULE_V1,
143     &ngx_rtsig_module_ctx,               /* module context */
144     ngx_rtsig_commands,                  /* module directives */
145     NGX_EVENT_MODULE,                    /* module type */
146     NULL,                                /* init master */
147     NULL,                                /* init module */
148     NULL,                                /* init process */
149     NULL,                                /* init thread */
150     NULL,                                /* exit thread */
151     NULL,                                /* exit process */
152     NULL,                                /* exit master */
153     NGX_MODULE_V1_PADDING
154 };
155
156
157 static ngx_int_t
158 ngx_rtsig_init(ngx_cycle_t *cycle, ngx_msec_t timer)
159 {
160     ngx_rtsig_conf_t  *rtscf;
161
162     rtscf = ngx_event_get_conf(cycle->conf_ctx, ngx_rtsig_module);
163
164     sigemptyset(&set);
165     sigaddset(&set, (int) rtscf->signo);
166     sigaddset(&set, (int) rtscf->signo + 1);
167     sigaddset(&set, SIGIO);
168     sigaddset(&set, SIGALRM);
169
170     if (sigprocmask(SIG_BLOCK, &set, NULL) == -1) {
171         ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
172                       "sigprocmask() failed");
173         return NGX_ERROR;
174     }
175
176     if (overflow_list) {
177         ngx_free(overflow_list);
178     }
179
180     overflow_list = ngx_alloc(sizeof(struct pollfd) * rtscf->overflow_events,
181                               cycle->log);
182     if (overflow_list == NULL) {
183         return NGX_ERROR;
184     }
185
186     ngx_io = ngx_os_io;
187
188     ngx_event_actions = ngx_rtsig_module_ctx.actions;
189
190     ngx_event_flags = NGX_USE_RTSIG_EVENT
191                       |NGX_USE_GREEDY_EVENT
192                       |NGX_USE_FD_EVENT;
193
194     return NGX_OK;
195 }
196
197
198 static void
199 ngx_rtsig_done(ngx_cycle_t *cycle)
200 {
201     ngx_free(overflow_list);
202
203     overflow_list = NULL;
204 }
205
206
207 static ngx_int_t
208 ngx_rtsig_add_connection(ngx_connection_t *c)
209 {
210     ngx_uint_t         signo;
211     ngx_rtsig_conf_t  *rtscf;
212
213     if (c->read->accept && c->read->disabled) {
214
215         ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
```

```
                       "rtsig enable connection: fd:%d", c->fd);

        if (fcntl(c->fd, F_SETOWN, ngx_pid) == -1) {
            ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                          "fcntl(F_SETOWN) failed");
            return NGX_ERROR;
        }

        c->read->active = 1;
        c->read->disabled = 0;
    }

    rtscf = ngx_event_get_conf(ngx_cycle->conf_ctx, ngx_rtsig_module);

    signo = rtscf->signo + c->read->instance;

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "rtsig add connection: fd:%d signo:%ui", c->fd, signo);

    if (fcntl(c->fd, F_SETFL, O_RDWR|O_NONBLOCK|O_ASYNC) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      "fcntl(O_RDWR|O_NONBLOCK|O_ASYNC) failed");
        return NGX_ERROR;
    }

    if (fcntl(c->fd, F_SETSIG, (int) signo) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      "fcntl(F_SETSIG) failed");
        return NGX_ERROR;
    }

    if (fcntl(c->fd, F_SETOWN, ngx_pid) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      "fcntl(F_SETOWN) failed");
        return NGX_ERROR;
    }

#if (NGX_HAVE_ONESIGFD)
    if (fcntl(c->fd, F_SETAUXFL, O_ONESIGFD) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      "fcntl(F_SETAUXFL) failed");
        return NGX_ERROR;
    }
#endif

    c->read->active = 1;
    c->write->active = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_rtsig_del_connection(ngx_connection_t *c, ngx_uint_t flags)
{
    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "rtsig del connection: fd:%d", c->fd);

    if ((flags & NGX_DISABLE_EVENT) && c->read->accept) {

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
                       "rtsig disable connection: fd:%d", c->fd);

        c->read->active = 0;
        c->read->disabled = 1;
        return NGX_OK;
    }

    if (flags & NGX_CLOSE_EVENT) {
        c->read->active = 0;
        c->write->active = 0;
        return NGX_OK;
    }

    if (fcntl(c->fd, F_SETFL, O_RDWR|O_NONBLOCK) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
```

```
                      "fcntl(O_RDWR|O_NONBLOCK) failed");
        return NGX_ERROR;
    }

    c->read->active = 0;
    c->write->active = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_rtsig_process_events(ngx_cycle_t *cycle, ngx_msec_t timer, ngx_uint_t flags)
{
    int                 signo;
    ngx_int_t           instance;
    ngx_err_t           err;
    siginfo_t           si;
    ngx_event_t        *rev, *wev;
    ngx_queue_t        *queue;
    struct timespec     ts, *tp;
    struct sigaction    sa;
    ngx_connection_t   *c;
    ngx_rtsig_conf_t   *rtscf;

    if (timer == NGX_TIMER_INFINITE) {
        tp = NULL;

    } else {
        ts.tv_sec = timer / 1000;
        ts.tv_nsec = (timer % 1000) * 1000000;
        tp = &ts;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "rtsig timer: %M", timer);

    /* Linux's sigwaitinfo() is sigtimedwait() with the NULL timeout pointer */

    signo = sigtimedwait(&set, &si, tp);

    if (signo == -1) {
        err = ngx_errno;

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, err,
                       "rtsig signo:%d", signo);

        if (flags & NGX_UPDATE_TIME) {
            ngx_time_update();
        }

        if (err == NGX_EAGAIN) {

            /* timeout */

            if (timer != NGX_TIMER_INFINITE) {
                return NGX_AGAIN;
            }

            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          "sigtimedwait() returned EAGAIN without timeout");
            return NGX_ERROR;
        }

        ngx_log_error((err == NGX_EINTR) ? NGX_LOG_INFO : NGX_LOG_ALERT,
                      cycle->log, err, "sigtimedwait() failed");
        return NGX_ERROR;
    }

    ngx_log_debug3(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "rtsig signo:%d fd:%d band:%04Xd",
                   signo, si.si_fd, si.si_band);

    if (flags & NGX_UPDATE_TIME) {
        ngx_time_update();
    }
```

```
368
369     rtscf = ngx_event_get_conf(ngx_cycle->conf_ctx, ngx_rtsig_module);
370
371     if (signo == (int) rtscf->signo || signo == (int) rtscf->signo + 1) {
372
373         if (overflow && (ngx_uint_t) si.si_fd > overflow_current) {
374             return NGX_OK;
375         }
376
377         c = ngx_cycle->files[si.si_fd];
378
379         if (c == NULL) {
380
381             /* the stale event */
382
383             return NGX_OK;
384         }
385
386         instance = signo - (int) rtscf->signo;
387
388         rev = c->read;
389
390         if (rev->instance != instance) {
391
392             /*
393              * the stale event from a file descriptor
394              * that was just closed in this iteration
395              */
396
397             ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
398                            "rtsig: stale event %p", c);
399
400             return NGX_OK;
401         }
402
403         if ((si.si_band & (POLLIN|POLLHUP|POLLERR)) && rev->active) {
404
405             rev->ready = 1;
406
407             if (flags & NGX_POST_EVENTS) {
408                 queue = rev->accept ? &ngx_posted_accept_events
409                                     : &ngx_posted_events;
410
411                 ngx_post_event(rev, queue);
412
413             } else {
414                 rev->handler(rev);
415             }
416         }
417
418         wev = c->write;
419
420         if ((si.si_band & (POLLOUT|POLLHUP|POLLERR)) && wev->active) {
421
422             wev->ready = 1;
423
424             if (flags & NGX_POST_EVENTS) {
425                 ngx_post_event(wev, &ngx_posted_events);
426
427             } else {
428                 wev->handler(wev);
429             }
430         }
431
432         return NGX_OK;
433
434     } else if (signo == SIGALRM) {
435
436         ngx_time_update();
437
438         return NGX_OK;
439
440     } else if (signo == SIGIO) {
441
442         ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
443                       "rt signal queue overflowed");
```

```
444
445         /* flush the RT signal queue */
446
447         ngx_memzero(&sa, sizeof(struct sigaction));
448         sa.sa_handler = SIG_DFL;
449         sigemptyset(&sa.sa_mask);
450
451         if (sigaction(rtscf->signo, &sa, NULL) == -1) {
452             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
453                           "sigaction(%d, SIG_DFL) failed", rtscf->signo);
454         }
455
456         if (sigaction(rtscf->signo + 1, &sa, NULL) == -1) {
457             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
458                           "sigaction(%d, SIG_DFL) failed", rtscf->signo + 1);
459         }
460
461         overflow = 1;
462         overflow_current = 0;
463         ngx_event_actions.process_events = ngx_rtsig_process_overflow;
464
465         return NGX_ERROR;
466
467     }
468
469     ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
470                   "sigtimedwait() returned unexpected signal: %d", signo);
471
472     return NGX_ERROR;
473 }
474
475
476 static ngx_int_t
477 ngx_rtsig_process_overflow(ngx_cycle_t *cycle, ngx_msec_t timer,
478     ngx_uint_t flags)
479 {
480     int                name[2], rtsig_max, rtsig_nr, events, ready;
481     size_t             len;
482     ngx_err_t          err;
483     ngx_uint_t         tested, n, i;
484     ngx_event_t       *rev, *wev;
485     ngx_queue_t       *queue;
486     ngx_connection_t  *c;
487     ngx_rtsig_conf_t  *rtscf;
488
489     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
490                    "rtsig process overflow");
491
492     rtscf = ngx_event_get_conf(ngx_cycle->conf_ctx, ngx_rtsig_module);
493
494     tested = 0;
495
496     for ( ;; ) {
497
498         n = 0;
499         while (n < rtscf->overflow_events) {
500
501             if (overflow_current == cycle->connection_n) {
502                 break;
503             }
504
505             c = cycle->files[overflow_current++];
506
507             if (c == NULL || c->fd == -1) {
508                 continue;
509             }
510
511             events = 0;
512
513             if (c->read->active && c->read->handler) {
514                 events |= POLLIN;
515             }
516
517             if (c->write->active && c->write->handler) {
518                 events |= POLLOUT;
519             }
```

```
            if (events == 0) {
                continue;
            }

            overflow_list[n].fd = c->fd;
            overflow_list[n].events = events;
            overflow_list[n].revents = 0;
            n++;
        }

        if (n == 0) {
            break;
        }

        for ( ;; ) {
            ready = poll(overflow_list, n, 0);

            ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "rtsig overflow poll:%d", ready);

            if (ready == -1) {
                err = ngx_errno;
                ngx_log_error((err == NGX_EINTR) ? NGX_LOG_INFO : NGX_LOG_ALERT,
                              cycle->log, 0,
                              "poll() failed while the overflow recover");

                if (err == NGX_EINTR) {
                    continue;
                }
            }

            break;
        }

        if (ready <= 0) {
            continue;
        }

        for (i = 0; i < n; i++) {
            c = cycle->files[overflow_list[i].fd];

            if (c == NULL) {
                continue;
            }

            rev = c->read;

            if (rev->active
                && !rev->closed
                && rev->handler
                && (overflow_list[i].revents
                                        & (POLLIN|POLLERR|POLLHUP|POLLNVAL)))
            {
                tested++;

                rev->ready = 1;

                if (flags & NGX_POST_EVENTS) {
                    queue = rev->accept ? &ngx_posted_accept_events
                                        : &ngx_posted_events;

                    ngx_post_event(rev, queue);

                } else {
                    rev->handler(rev);
                }
            }

            wev = c->write;

            if (wev->active
                && !wev->closed
                && wev->handler
                && (overflow_list[i].revents
                                        & (POLLOUT|POLLERR|POLLHUP|POLLNVAL)))
```

```
596             {
597                 tested++;
598
599                 wev->ready = 1;
600
601                 if (flags & NGX_POST_EVENTS) {
602                     ngx_post_event(wev, &ngx_posted_events);
603
604                 } else {
605                     wev->handler(wev);
606                 }
607             }
608         }
609
610         if (tested >= rtscf->overflow_test) {
611
612             if (ngx_linux_rtsig_max) {
613
614                 /*
615                  * Check the current rt queue length to prevent
616                  * the new overflow.
617                  *
618                  * learn the "/proc/sys/kernel/rtsig-max" value because
619                  * it can be changed since the last checking
620                  */
621
622                 name[0] = CTL_KERN;
623                 name[1] = KERN_RTSIGMAX;
624                 len = sizeof(rtsig_max);
625
626                 if (sysctl(name, 2, &rtsig_max, &len, NULL, 0) == -1) {
627                     ngx_log_error(NGX_LOG_ALERT, cycle->log, errno,
628                                   "sysctl(KERN_RTSIGMAX) failed");
629                     return NGX_ERROR;
630                 }
631
632                 /* name[0] = CTL_KERN; */
633                 name[1] = KERN_RTSIGNR;
634                 len = sizeof(rtsig_nr);
635
636                 if (sysctl(name, 2, &rtsig_nr, &len, NULL, 0) == -1) {
637                     ngx_log_error(NGX_LOG_ALERT, cycle->log, errno,
638                                   "sysctl(KERN_RTSIGNR) failed");
639                     return NGX_ERROR;
640                 }
641
642                 /*
643                  * drain the rt signal queue if the /"proc/sys/kernel/rtsig-nr"
644                  * is bigger than
645                  *    "/proc/sys/kernel/rtsig-max" / "rtsig_overflow_threshold"
646                  */
647
648                 if (rtsig_max / (int) rtscf->overflow_threshold < rtsig_nr) {
649                     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
650                                    "rtsig queue state: %d/%d",
651                                    rtsig_nr, rtsig_max);
652                     while (ngx_rtsig_process_events(cycle, 0, flags) == NGX_OK)
653                     {
654                         /* void */
655                     }
656                 }
657
658             } else {
659
660                 /*
661                  * Linux has not KERN_RTSIGMAX since 2.6.6-mm2
662                  * so drain the rt signal queue unconditionally
663                  */
664
665                 while (ngx_rtsig_process_events(cycle, 0, flags) == NGX_OK) {
666                     /* void */
667                 }
668             }
669
670             tested = 0;
671         }
```

```
672     }
673
674     if (flags & NGX_UPDATE_TIME) {
675         ngx_time_update();
676     }
677
678     ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
679                   "rt signal queue overflow recovered");
680
681     overflow = 0;
682     ngx_event_actions.process_events = ngx_rtsig_process_events;
683
684     return NGX_OK;
685 }
686
687
688 static void *
689 ngx_rtsig_create_conf(ngx_cycle_t *cycle)
690 {
691     ngx_rtsig_conf_t  *rtscf;
692
693     rtscf = ngx_palloc(cycle->pool, sizeof(ngx_rtsig_conf_t));
694     if (rtscf == NULL) {
695         return NULL;
696     }
697
698     rtscf->signo = NGX_CONF_UNSET;
699     rtscf->overflow_events = NGX_CONF_UNSET;
700     rtscf->overflow_test = NGX_CONF_UNSET;
701     rtscf->overflow_threshold = NGX_CONF_UNSET;
702
703     return rtscf;
704 }
705
706
707 static char *
708 ngx_rtsig_init_conf(ngx_cycle_t *cycle, void *conf)
709 {
710     ngx_rtsig_conf_t  *rtscf = conf;
711
712     /* LinuxThreads use the first 3 RT signals */
713     ngx_conf_init_uint_value(rtscf->signo, SIGRTMIN + 10);
714
715     ngx_conf_init_uint_value(rtscf->overflow_events, 16);
716     ngx_conf_init_uint_value(rtscf->overflow_test, 32);
717     ngx_conf_init_uint_value(rtscf->overflow_threshold, 10);
718
719     return NGX_CONF_OK;
720 }
721
722
723 static char *
724 ngx_check_ngx_overflow_threshold_bounds(ngx_conf_t *cf, void *post, void *data)
725 {
726     if (ngx_linux_rtsig_max) {
727         return ngx_conf_check_num_bounds(cf, post, data);
728     }
729
730     ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
731                        "\"rtsig_overflow_threshold\" is not supported "
732                        "since Linux 2.6.6-mm2, ignored");
733
734     return NGX_CONF_OK;
735 }
```

One Level Up Top Level

src/event/ngx_event.h - nginx-1.7.10

Data types defined

- ngx_event_actions_t
- ngx_event_aio_s
- ngx_event_conf_t
- ngx_event_module_t
- ngx_event_mutex_t
- ngx_event_ovlp_t
- ngx_event_s

Macros defined

- EPOLLRDHUP
- NGX_CLEAR_EVENT
- NGX_CLEAR_EVENT
- NGX_CLEAR_EVENT
- NGX_CLOSE_EVENT
- NGX_CLOSE_EVENT
- NGX_CLOSE_EVENT
- NGX_DISABLE_EVENT
- NGX_DISABLE_EVENT
- NGX_DISABLE_EVENT
- NGX_EVENT_CONF
- NGX_EVENT_MODULE
- NGX_FLUSH_EVENT
- NGX_FLUSH_EVENT
- NGX_FLUSH_EVENT
- NGX_INVALID_INDEX
- NGX_IOCP_ACCEPT
- NGX_IOCP_CONNECT
- NGX_IOCP_IO
- NGX_LEVEL_EVENT
- NGX_LEVEL_EVENT

- NGX_LEVEL_EVENT
- NGX_LEVEL_EVENT
- NGX_LEVEL_EVENT
- NGX_LOWAT_EVENT
- NGX_LOWAT_EVENT
- NGX_LOWAT_EVENT
- NGX_ONESHOT_EVENT
- NGX_ONESHOT_EVENT
- NGX_ONESHOT_EVENT
- NGX_ONESHOT_EVENT
- NGX_ONESHOT_EVENT
- NGX_POST_EVENTS
- NGX_READ_EVENT
- NGX_READ_EVENT
- NGX_READ_EVENT
- NGX_READ_EVENT
- NGX_READ_EVENT
- NGX_UPDATE_TIME
- NGX_USE_AIO_EVENT
- NGX_USE_CLEAR_EVENT
- NGX_USE_EPOLL_EVENT
- NGX_USE_EVENTPORT_EVENT
- NGX_USE_FD_EVENT
- NGX_USE_GREEDY_EVENT
- NGX_USE_IOCP_EVENT
- NGX_USE_KQUEUE_EVENT
- NGX_USE_LEVEL_EVENT
- NGX_USE_LOWAT_EVENT
- NGX_USE_ONESHOT_EVENT
- NGX_USE_RTSIG_EVENT
- NGX_USE_TIMER_EVENT
- NGX_USE_VNODE_EVENT
- NGX_VNODE_EVENT

- NGX_VNODE_EVENT
- NGX_VNODE_EVENT
- NGX_WRITE_EVENT
- NGX_WRITE_EVENT
- NGX_WRITE_EVENT
- NGX_WRITE_EVENT
- NGX_WRITE_EVENT
- _NGX_EVENT_H_INCLUDED_
- ngx_add_conn
- ngx_add_event
- ngx_add_timer
- ngx_del_conn
- ngx_del_event
- ngx_del_timer
- ngx_done_events
- ngx_event_get_conf
- ngx_event_ident
- ngx_process_changes
- ngx_process_events
- ngx_recv
- ngx_recv_chain
- ngx_send
- ngx_send_chain
- ngx_udp_recv

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_EVENT_H_INCLUDED_
9  #define _NGX_EVENT_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #define NGX_INVALID_INDEX  0xd0d0d0d0
17
18
```

```
#if (NGX_HAVE_IOCP)

typedef struct {
    WSAOVERLAPPED    ovlp;
    ngx_event_t     *event;
    int              error;
} ngx_event_ovlp_t;

#endif


typedef struct {
    ngx_uint_t       lock;

    ngx_event_t     *events;
    ngx_event_t     *last;
} ngx_event_mutex_t;


struct ngx_event_s {
    void            *data;

    unsigned         write:1;

    unsigned         accept:1;

    /* used to detect the stale events in kqueue, rtsig, and epoll */
    unsigned         instance:1;

    /*
     * the event was passed or would be passed to a kernel;
     * in aio mode - operation was posted.
     */
    unsigned         active:1;

    unsigned         disabled:1;

    /* the ready event; in aio mode 0 means that no operation can be posted */
    unsigned         ready:1;

    unsigned         oneshot:1;

    /* aio operation is complete */
    unsigned         complete:1;

    unsigned         eof:1;
    unsigned         error:1;

    unsigned         timedout:1;
    unsigned         timer_set:1;

    unsigned         delayed:1;

    unsigned         deferred_accept:1;

    /* the pending eof reported by kqueue, epoll or in aio chain operation */
    unsigned         pending_eof:1;

    unsigned         posted:1;

#if (NGX_WIN32)
    /* setsockopt(SO_UPDATE_ACCEPT_CONTEXT) was successful */
    unsigned         accept_context_updated:1;
#endif

#if (NGX_HAVE_KQUEUE)
    unsigned         kq_vnode:1;

    /* the pending errno reported by kqueue */
    int              kq_errno;
#endif

    /*
     * kqueue only:
     *   accept:     number of sockets that wait to be accepted
     *   read:       bytes to read when event is ready
```

```
95      *               or lowat when event is set with NGX_LOWAT_EVENT flag
96      *   write:      available space in buffer when event is ready
97      *               or lowat when event is set with NGX_LOWAT_EVENT flag
98      *
99      * iocp: TODO
100      *
101      * otherwise:
102      *   accept:     1 if accept many, 0 otherwise
103      */
104
105 #if (NGX_HAVE_KQUEUE) || (NGX_HAVE_IOCP)
106     int              available;
107 #else
108     unsigned         available:1;
109 #endif
110
111     ngx_event_handler_pt  handler;
112
113
114 #if (NGX_HAVE_AIO)
115
116 #if (NGX_HAVE_IOCP)
117     ngx_event_ovlp_t ovlp;
118 #else
119     struct aiocb     aiocb;
120 #endif
121
122 #endif
123
124     ngx_uint_t       index;
125
126     ngx_log_t       *log;
127
128     ngx_rbtree_node_t   timer;
129
130     /* the posted queue */
131     ngx_queue_t      queue;
132
133     unsigned         closed:1;
134
135     /* to test on worker exit */
136     unsigned         channel:1;
137     unsigned         resolver:1;
138
139     unsigned         cancelable:1;
140
141
142 #if 0
143
144     /* the threads support */
145
146     /*
147      * the event thread context, we store it here
148      * if $(CC) does not understand __thread declaration
149      * and pthread_getspecific() is too costly
150      */
151
152     void            *thr_ctx;
153
154 #if (NGX_EVENT_T_PADDING)
155
156     /* event should not cross cache line in SMP */
157
158     uint32_t         padding[NGX_EVENT_T_PADDING];
159 #endif
160 #endif
161 };
162
163
164 #if (NGX_HAVE_FILE_AIO)
165
166 struct ngx_event_aio_s {
167     void                      *data;
168     ngx_event_handler_pt       handler;
169     ngx_file_t                *file;
170
```

```
    ngx_fd_t                   fd;

#if (NGX_HAVE_EVENTFD)
    int64_t                    res;
#if (NGX_TEST_BUILD_EPOLL)
    ngx_err_t                  err;
    size_t                     nbytes;
#endif
#else
    ngx_err_t                  err;
    size_t                     nbytes;
#endif

#if (NGX_HAVE_AIO_SENDFILE)
    off_t                      last_offset;
#endif

    ngx_aiocb_t                aiocb;
    ngx_event_t                event;
};

#endif


typedef struct {
    ngx_int_t  (*add)(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags);
    ngx_int_t  (*del)(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags);

    ngx_int_t  (*enable)(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags);
    ngx_int_t  (*disable)(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags);

    ngx_int_t  (*add_conn)(ngx_connection_t *c);
    ngx_int_t  (*del_conn)(ngx_connection_t *c, ngx_uint_t flags);

    ngx_int_t  (*process_changes)(ngx_cycle_t *cycle, ngx_uint_t nowait);
    ngx_int_t  (*process_events)(ngx_cycle_t *cycle, ngx_msec_t timer,
                ngx_uint_t flags);

    ngx_int_t  (*init)(ngx_cycle_t *cycle, ngx_msec_t timer);
    void       (*done)(ngx_cycle_t *cycle);
} ngx_event_actions_t;


extern ngx_event_actions_t   ngx_event_actions;


/*
 * The event filter requires to read/write the whole data:
 * select, poll, /dev/poll, kqueue, epoll.
 */
#define NGX_USE_LEVEL_EVENT      0x00000001

/*
 * The event filter is deleted after a notification without an additional
 * syscall: kqueue, epoll.
 */
#define NGX_USE_ONESHOT_EVENT    0x00000002

/*
 * The event filter notifies only the changes and an initial level:
 * kqueue, epoll.
 */
#define NGX_USE_CLEAR_EVENT      0x00000004

/*
 * The event filter has kqueue features: the eof flag, errno,
 * available data, etc.
 */
#define NGX_USE_KQUEUE_EVENT     0x00000008

/*
 * The event filter supports low water mark: kqueue's NOTE_LOWAT.
 * kqueue in FreeBSD 4.1-4.2 has no NOTE_LOWAT so we need a separate flag.
 */
#define NGX_USE_LOWAT_EVENT      0x00000010

```

```
/*
* The event filter requires to do i/o operation until EAGAIN: epoll, rtsig.
*/
#define NGX_USE_GREEDY_EVENT     0x00000020

/*
* The event filter is epoll.
*/
#define NGX_USE_EPOLL_EVENT      0x00000040

/*
* No need to add or delete the event filters: rtsig.
*/
#define NGX_USE_RTSIG_EVENT      0x00000080

/*
* No need to add or delete the event filters: overlapped, aio_read,
* aioread, io_submit.
*/
#define NGX_USE_AIO_EVENT        0x00000100

/*
* Need to add socket or handle only once: i/o completion port.
* It also requires NGX_HAVE_AIO and NGX_USE_AIO_EVENT to be set.
*/
#define NGX_USE_IOCP_EVENT       0x00000200

/*
* The event filter has no opaque data and requires file descriptors table:
* poll, /dev/poll, rtsig.
*/
#define NGX_USE_FD_EVENT         0x00000400

/*
* The event module handles periodic or absolute timer event by itself:
* kqueue in FreeBSD 4.4, NetBSD 2.0, and MacOSX 10.4, Solaris 10's event ports.
*/
#define NGX_USE_TIMER_EVENT      0x00000800

/*
* All event filters on file descriptor are deleted after a notification:
* Solaris 10's event ports.
*/
#define NGX_USE_EVENTPORT_EVENT  0x00001000

/*
* The event filter support vnode notifications: kqueue.
*/
#define NGX_USE_VNODE_EVENT      0x00002000


/*
* The event filter is deleted just before the closing file.
* Has no meaning for select and poll.
* kqueue, epoll, rtsig, eventport:  allows to avoid explicit delete,
*                                   because filter automatically is deleted
*                                   on file close,
*
* /dev/poll:                        we need to flush POLLREMOVE event
*                                   before closing file.
*/
#define NGX_CLOSE_EVENT    1

/*
* disable temporarily event filter, this may avoid locks
* in kernel malloc()/free(): kqueue.
*/
#define NGX_DISABLE_EVENT  2

/*
* event must be passed to kernel right now, do not wait until batch processing.
*/
#define NGX_FLUSH_EVENT    4


/* these flags have a meaning only for kqueue */
```

```
#define NGX_LOWAT_EVENT    0
#define NGX_VNODE_EVENT    0


#if (NGX_HAVE_EPOLL) && !(NGX_HAVE_EPOLLRDHUP)
#define EPOLLRDHUP         0
#endif


#if (NGX_HAVE_KQUEUE)

#define NGX_READ_EVENT     EVFILT_READ
#define NGX_WRITE_EVENT    EVFILT_WRITE

#undef  NGX_VNODE_EVENT
#define NGX_VNODE_EVENT    EVFILT_VNODE

/*
 * NGX_CLOSE_EVENT, NGX_LOWAT_EVENT, and NGX_FLUSH_EVENT are the module flags
 * and they must not go into a kernel so we need to choose the value
 * that must not interfere with any existent and future kqueue flags.
 * kqueue has such values - EV_FLAG1, EV_EOF, and EV_ERROR:
 * they are reserved and cleared on a kernel entrance.
 */
#undef  NGX_CLOSE_EVENT
#define NGX_CLOSE_EVENT    EV_EOF

#undef  NGX_LOWAT_EVENT
#define NGX_LOWAT_EVENT    EV_FLAG1

#undef  NGX_FLUSH_EVENT
#define NGX_FLUSH_EVENT    EV_ERROR

#define NGX_LEVEL_EVENT    0
#define NGX_ONESHOT_EVENT  EV_ONESHOT
#define NGX_CLEAR_EVENT    EV_CLEAR

#undef  NGX_DISABLE_EVENT
#define NGX_DISABLE_EVENT  EV_DISABLE


#elif (NGX_HAVE_DEVPOLL || NGX_HAVE_EVENTPORT)

#define NGX_READ_EVENT     POLLIN
#define NGX_WRITE_EVENT    POLLOUT

#define NGX_LEVEL_EVENT    0
#define NGX_ONESHOT_EVENT  1


#elif (NGX_HAVE_EPOLL)

#define NGX_READ_EVENT     (EPOLLIN|EPOLLRDHUP)
#define NGX_WRITE_EVENT    EPOLLOUT

#define NGX_LEVEL_EVENT    0
#define NGX_CLEAR_EVENT    EPOLLET
#define NGX_ONESHOT_EVENT  0x70000000
#if 0
#define NGX_ONESHOT_EVENT  EPOLLONESHOT
#endif


#elif (NGX_HAVE_POLL)

#define NGX_READ_EVENT     POLLIN
#define NGX_WRITE_EVENT    POLLOUT

#define NGX_LEVEL_EVENT    0
#define NGX_ONESHOT_EVENT  1


#else /* select */

#define NGX_READ_EVENT     0
#define NGX_WRITE_EVENT    1
```

```
399
400 #define NGX_LEVEL_EVENT    0
401 #define NGX_ONESHOT_EVENT  1
402
403 #endif /* NGX_HAVE_KQUEUE */
404
405
406 #if (NGX_HAVE_IOCP)
407 #define NGX_IOCP_ACCEPT      0
408 #define NGX_IOCP_IO          1
409 #define NGX_IOCP_CONNECT     2
410 #endif
411
412
413 #ifndef NGX_CLEAR_EVENT
414 #define NGX_CLEAR_EVENT    0    /* dummy declaration */
415 #endif
416
417
418 #define ngx_process_changes  ngx_event_actions.process_changes
419 #define ngx_process_events   ngx_event_actions.process_events
420 #define ngx_done_events      ngx_event_actions.done
421
422 #define ngx_add_event        ngx_event_actions.add
423 #define ngx_del_event        ngx_event_actions.del
424 #define ngx_add_conn         ngx_event_actions.add_conn
425 #define ngx_del_conn         ngx_event_actions.del_conn
426
427 #define ngx_add_timer        ngx_event_add_timer
428 #define ngx_del_timer        ngx_event_del_timer
429
430
431 extern ngx_os_io_t  ngx_io;
432
433 #define ngx_recv             ngx_io.recv
434 #define ngx_recv_chain       ngx_io.recv_chain
435 #define ngx_udp_recv         ngx_io.udp_recv
436 #define ngx_send             ngx_io.send
437 #define ngx_send_chain       ngx_io.send_chain
438
439
440 #define NGX_EVENT_MODULE      0x544E5645  /* "EVNT" */
441 #define NGX_EVENT_CONF        0x02000000
442
443
444 typedef struct {
445     ngx_uint_t    connections;
446     ngx_uint_t    use;
447
448     ngx_flag_t    multi_accept;
449     ngx_flag_t    accept_mutex;
450
451     ngx_msec_t    accept_mutex_delay;
452
453     u_char       *name;
454
455 #if (NGX_DEBUG)
456     ngx_array_t   debug_connection;
457 #endif
458 } ngx_event_conf_t;
459
460
461 typedef struct {
462     ngx_str_t             *name;
463
464     void                *(*create_conf)(ngx_cycle_t *cycle);
465     char                 (*init_conf)(ngx_cycle_t *cycle, void *conf);
466
467     ngx_event_actions_t     actions;
468 } ngx_event_module_t;
469
470
471 extern ngx_atomic_t          *ngx_connection_counter;
472
473 extern ngx_atomic_t          *ngx_accept_mutex_ptr;
474 extern ngx_shmtx_t            ngx_accept_mutex;
```

```
475 extern ngx_uint_t             ngx_use_accept_mutex;
476 extern ngx_uint_t             ngx_accept_events;
477 extern ngx_uint_t             ngx_accept_mutex_held;
478 extern ngx_msec_t             ngx_accept_mutex_delay;
479 extern ngx_int_t              ngx_accept_disabled;
480
481
482 #if (NGX_STAT_STUB)
483
484 extern ngx_atomic_t  *ngx_stat_accepted;
485 extern ngx_atomic_t  *ngx_stat_handled;
486 extern ngx_atomic_t  *ngx_stat_requests;
487 extern ngx_atomic_t  *ngx_stat_active;
488 extern ngx_atomic_t  *ngx_stat_reading;
489 extern ngx_atomic_t  *ngx_stat_writing;
490 extern ngx_atomic_t  *ngx_stat_waiting;
491
492 #endif
493
494
495 #define NGX_UPDATE_TIME         1
496 #define NGX_POST_EVENTS         2
497
498
499 extern sig_atomic_t           ngx_event_timer_alarm;
500 extern ngx_uint_t             ngx_event_flags;
501 extern ngx_module_t           ngx_events_module;
502 extern ngx_module_t           ngx_event_core_module;
503
504
505 #define ngx_event_get_conf(conf_ctx, module)                                 \
506              (*(ngx_get_conf(conf_ctx, ngx_events_module))) [module.ctx_index];
507
508
509
510 void ngx_event_accept(ngx_event_t *ev);
511 ngx_int_t ngx_trylock_accept_mutex(ngx_cycle_t *cycle);
512 u_char *ngx_accept_log_error(ngx_log_t *log, u_char *buf, size_t len);
513
514
515 void ngx_process_events_and_timers(ngx_cycle_t *cycle);
516 ngx_int_t ngx_handle_read_event(ngx_event_t *rev, ngx_uint_t flags);
517 ngx_int_t ngx_handle_write_event(ngx_event_t *wev, size_t lowat);
518
519
520 #if (NGX_WIN32)
521 void ngx_event_acceptex(ngx_event_t *ev);
522 ngx_int_t ngx_event_post_acceptex(ngx_listening_t *ls, ngx_uint_t n);
523 u_char *ngx_acceptex_log_error(ngx_log_t *log, u_char *buf, size_t len);
524 #endif
525
526
527 ngx_int_t ngx_send_lowat(ngx_connection_t *c, size_t lowat);
528
529
530 /* used in ngx_log_debugX() */
531 #define ngx_event_ident(p)  ((ngx_connection_t *) (p))->fd
532
533
534 #include <ngx_event_timer.h>
535 #include <ngx_event_posted.h>
536 #include <ngx_event_busy_lock.h>
537
538 #if (NGX_WIN32)
539 #include <ngx_iocp_module.h>
540 #endif
541
542
543 #endif /* _NGX_EVENT_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_queue.h - nginx-1.7.10

Data types defined

- ngx_queue_s
- ngx_queue_t

Macros defined

- _NGX_QUEUE_H_INCLUDED_
- ngx_queue_add
- ngx_queue_data
- ngx_queue_empty
- ngx_queue_head
- ngx_queue_init
- ngx_queue_insert_after
- ngx_queue_insert_head
- ngx_queue_insert_tail
- ngx_queue_last
- ngx_queue_next
- ngx_queue_prev
- ngx_queue_remove
- ngx_queue_remove
- ngx_queue_sentinel
- ngx_queue_split

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  #ifndef _NGX_QUEUE_H_INCLUDED_
13  #define _NGX_QUEUE_H_INCLUDED_
14
15
16  typedef struct ngx_queue_s  ngx_queue_t;
17
18  struct ngx_queue_s {
19      ngx_queue_t  *prev;
20      ngx_queue_t  *next;
```

```
};


#define ngx_queue_init(q)                                                     \
    (q)->prev = q;                                                            \
    (q)->next = q


#define ngx_queue_empty(h)                                                    \
    (h == (h)->prev)


#define ngx_queue_insert_head(h, x)                                           \
    (x)->next = (h)->next;                                                    \
    (x)->next->prev = x;                                                      \
    (x)->prev = h;                                                            \
    (h)->next = x


#define ngx_queue_insert_after   ngx_queue_insert_head


#define ngx_queue_insert_tail(h, x)                                           \
    (x)->prev = (h)->prev;                                                    \
    (x)->prev->next = x;                                                      \
    (x)->next = h;                                                            \
    (h)->prev = x


#define ngx_queue_head(h)                                                     \
    (h)->next


#define ngx_queue_last(h)                                                     \
    (h)->prev


#define ngx_queue_sentinel(h)                                                 \
    (h)


#define ngx_queue_next(q)                                                     \
    (q)->next


#define ngx_queue_prev(q)                                                     \
    (q)->prev


#if (NGX_DEBUG)

#define ngx_queue_remove(x)                                                   \
    (x)->next->prev = (x)->prev;                                              \
    (x)->prev->next = (x)->next;                                              \
    (x)->prev = NULL;                                                         \
    (x)->next = NULL

#else

#define ngx_queue_remove(x)                                                   \
    (x)->next->prev = (x)->prev;                                              \
    (x)->prev->next = (x)->next

#endif


#define ngx_queue_split(h, q, n)                                              \
    (n)->prev = (h)->prev;                                                    \
    (n)->prev->next = n;                                                      \
    (n)->next = q;                                                            \
    (h)->prev = (q)->prev;                                                    \
    (h)->prev->next = h;                                                      \
    (q)->prev = n;


#define ngx_queue_add(h, n)                                                   \
```

```
97      (h)->prev->next = (n)->next;                                         \
98      (n)->next->prev = (h)->prev;                                         \
99      (h)->prev = (n)->prev;                                               \
100      (h)->prev->next = h;
101
102
103  #define ngx_queue_data(q, type, link)                                     \
104      (type *) ((u_char *) q - offsetof(type, link))
105
106
107  ngx_queue_t *ngx_queue_middle(ngx_queue_t *queue);
108  void ngx_queue_sort(ngx_queue_t *queue,
109      ngx_int_t (*cmp)(const ngx_queue_t *, const ngx_queue_t *));
110
111
112  #endif /* _NGX_QUEUE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_queue.c - nginx-1.7.10

Functions defined

- ngx_queue_middle
- ngx_queue_sort

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13  * find the middle queue element if the queue has odd number of elements
14  * or the first element of the queue's second part otherwise
15  */
16
17  ngx_queue_t *
18  ngx_queue_middle(ngx_queue_t *queue)
19  {
20      ngx_queue_t  *middle, *next;
21
22      middle = ngx_queue_head(queue);
23
24      if (middle == ngx_queue_last(queue)) {
25          return middle;
26      }
27
28      next = ngx_queue_head(queue);
29
30      for ( ;; ) {
31          middle = ngx_queue_next(middle);
32
33          next = ngx_queue_next(next);
34
35          if (next == ngx_queue_last(queue)) {
36              return middle;
37          }
38
39          next = ngx_queue_next(next);
40
41          if (next == ngx_queue_last(queue)) {
42              return middle;
43          }
44      }
45  }
46
47
48  /* the stable insertion sort */
49
50  void
51  ngx_queue_sort(ngx_queue_t *queue,
52      ngx_int_t (*cmp)(const ngx_queue_t *, const ngx_queue_t *))
53  {
54      ngx_queue_t  *q, *prev, *next;
55
56      q = ngx_queue_head(queue);
57
58      if (q == ngx_queue_last(queue)) {
59          return;
60      }
```

```
61
62      for (q = ngx_queue_next(q); q != ngx_queue_sentinel(queue); q = next) {
63
64          prev = ngx_queue_prev(q);
65          next = ngx_queue_next(q);
66
67          ngx_queue_remove(q);
68
69          do {
70              if (cmp(prev, q) <= 0) {
71                  break;
72              }
73
74              prev = ngx_queue_prev(prev);
75
76          } while (prev != ngx_queue_sentinel(queue));
77
78          ngx_queue_insert_after(prev, q);
79      }
80  }
```

One Level Up Top Level

src/os/unix/ngx_files.h - nginx-1.7.10

Data types defined

- ngx_dir_t
- ngx_fd_t
- ngx_file_info_t
- ngx_file_mapping_t
- ngx_file_uniq_t
- ngx_glob_t

Functions defined

- ngx_de_info
- ngx_write_fd

Macros defined

- NGX_AT_FDCWD
- NGX_DIR_MASK_LEN
- NGX_FILE_APPEND
- NGX_FILE_CREATE_OR_OPEN
- NGX_FILE_DEFAULT_ACCESS
- NGX_FILE_DIRECTORY
- NGX_FILE_DIRECTORY
- NGX_FILE_ERROR
- NGX_FILE_NOFOLLOW
- NGX_FILE_NONBLOCK
- NGX_FILE_OPEN
- NGX_FILE_OWNER_ACCESS
- NGX_FILE_RDONLY
- NGX_FILE_RDWR
- NGX_FILE_SEARCH
- NGX_FILE_SEARCH
- NGX_FILE_SEARCH
- NGX_FILE_SEARCH
- NGX_FILE_TRUNCATE

- NGX_FILE_WRONLY
- NGX_HAVE_CASELESS_FILESYSTEM
- NGX_HAVE_MAX_PATH
- NGX_HAVE_READ_AHEAD
- NGX_HAVE_READ_AHEAD
- NGX_INVALID_FILE
- NGX_LINEFEED
- NGX_LINEFEED_SIZE
- NGX_MAX_PATH
- NGX_MAX_PATH
- _NGX_FILES_H_INCLUDED_
- ngx_change_file_access
- ngx_change_file_access_n
- ngx_close_dir
- ngx_close_dir_n
- ngx_close_file
- ngx_close_file_n
- ngx_create_dir
- ngx_create_dir_n
- ngx_de_access
- ngx_de_fs_size
- ngx_de_info_n
- ngx_de_is_dir
- ngx_de_is_dir
- ngx_de_is_file
- ngx_de_is_file
- ngx_de_is_link
- ngx_de_is_link
- ngx_de_link_info
- ngx_de_link_info_n
- ngx_de_mtime
- ngx_de_name
- ngx_de_namelen

- ngx_de_namelen
- ngx_de_size
- ngx_delete_dir
- ngx_delete_dir_n
- ngx_delete_file
- ngx_delete_file_n
- ngx_dir_access
- ngx_directio_off_n
- ngx_directio_on
- ngx_directio_on
- ngx_directio_on
- ngx_directio_on_n
- ngx_directio_on_n
- ngx_directio_on_n
- ngx_directio_on_n
- ngx_fd_info
- ngx_fd_info_n
- ngx_file_access
- ngx_file_at_info
- ngx_file_at_info_n
- ngx_file_fs_size
- ngx_file_info
- ngx_file_info_n
- ngx_file_mtime
- ngx_file_size
- ngx_file_uniq
- ngx_getcwd
- ngx_getcwd_n
- ngx_is_dir
- ngx_is_exec
- ngx_is_file
- ngx_is_link
- ngx_linefeed

- ngx_link_info
- ngx_link_info_n
- ngx_lock_fd_n
- ngx_open_dir_n
- ngx_open_file
- ngx_open_file
- ngx_open_file_n
- ngx_open_glob_n
- ngx_open_tempfile_n
- ngx_openat_file
- ngx_openat_file_n
- ngx_path_separator
- ngx_read_ahead
- ngx_read_ahead
- ngx_read_ahead_n
- ngx_read_ahead_n
- ngx_read_ahead_n
- ngx_read_dir_n
- ngx_read_fd
- ngx_read_fd_n
- ngx_read_file_n
- ngx_read_file_n
- ngx_realpath
- ngx_realpath_n
- ngx_rename_file
- ngx_rename_file_n
- ngx_set_file_time_n
- ngx_set_stderr
- ngx_set_stderr_n
- ngx_stderr
- ngx_trylock_fd_n
- ngx_unlock_fd_n
- ngx_write_console

- ngx_write_fd_n

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_FILES_H_INCLUDED_
9  #define _NGX_FILES_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef int                      ngx_fd_t;
17  typedef struct stat              ngx_file_info_t;
18  typedef ino_t                    ngx_file_uniq_t;
19
20
21  typedef struct {
22      u_char                      *name;
23      size_t                       size;
24      void                        *addr;
25      ngx_fd_t                     fd;
26      ngx_log_t                   *log;
27  } ngx_file_mapping_t;
28
29
30  typedef struct {
31      DIR                         *dir;
32      struct dirent               *de;
33      struct stat                  info;
34
35      unsigned                     type:8;
36      unsigned                     valid_info:1;
37  } ngx_dir_t;
38
39
40  typedef struct {
41      size_t                       n;
42      glob_t                       pglob;
43      u_char                      *pattern;
44      ngx_log_t                   *log;
45      ngx_uint_t                   test;
46  } ngx_glob_t;
47
48
49  #define NGX_INVALID_FILE         -1
50  #define NGX_FILE_ERROR           -1
51
52
53
54  #ifdef __CYGWIN__
55
56  #ifndef NGX_HAVE_CASELESS_FILESYSTEM
57  #define NGX_HAVE_CASELESS_FILESYSTEM  1
58  #endif
59
60  #define ngx_open_file(name, mode, create, access)                           \
61      open((const char *) name, mode|create|O_BINARY, access)
62
63  #else
64
65  #define ngx_open_file(name, mode, create, access)                           \
66      open((const char *) name, mode|create, access)
67
68  #endif
69
70  #define ngx_open_file_n          "open()"
```

```
71
72  #define NGX_FILE_RDONLY          O_RDONLY
73  #define NGX_FILE_WRONLY          O_WRONLY
74  #define NGX_FILE_RDWR            O_RDWR
75  #define NGX_FILE_CREATE_OR_OPEN  O_CREAT
76  #define NGX_FILE_OPEN            0
77  #define NGX_FILE_TRUNCATE        (O_CREAT|O_TRUNC)
78  #define NGX_FILE_APPEND          (O_WRONLY|O_APPEND)
79  #define NGX_FILE_NONBLOCK        O_NONBLOCK
80
81  #if (NGX_HAVE_OPENAT)
82  #define NGX_FILE_NOFOLLOW        O_NOFOLLOW
83
84  #if defined(O_DIRECTORY)
85  #define NGX_FILE_DIRECTORY       O_DIRECTORY
86  #else
87  #define NGX_FILE_DIRECTORY       0
88  #endif
89
90  #if defined(O_SEARCH)
91  #define NGX_FILE_SEARCH          (O_SEARCH|NGX_FILE_DIRECTORY)
92
93  #elif defined(O_EXEC)
94  #define NGX_FILE_SEARCH          (O_EXEC|NGX_FILE_DIRECTORY)
95
96  #elif (NGX_HAVE_O_PATH)
97  #define NGX_FILE_SEARCH          (O_PATH|O_RDONLY|NGX_FILE_DIRECTORY)
98
99  #else
100  #define NGX_FILE_SEARCH          (O_RDONLY|NGX_FILE_DIRECTORY)
101  #endif
102
103  #endif /* NGX_HAVE_OPENAT */
104
105  #define NGX_FILE_DEFAULT_ACCESS  0644
106  #define NGX_FILE_OWNER_ACCESS    0600
107
108
109  #define ngx_close_file           close
110  #define ngx_close_file_n         "close()"
111
112
113  #define ngx_delete_file(name)    unlink((const char *) name)
114  #define ngx_delete_file_n        "unlink()"
115
116
117  ngx_fd_t ngx_open_tempfile(u_char *name, ngx_uint_t persistent,
118      ngx_uint_t access);
119  #define ngx_open_tempfile_n      "open()"
120
121
122  ssize_t ngx_read_file(ngx_file_t *file, u_char *buf, size_t size, off_t offset);
123  #if (NGX_HAVE_PREAD)
124  #define ngx_read_file_n          "pread()"
125  #else
126  #define ngx_read_file_n          "read()"
127  #endif
128
129  ssize_t ngx_write_file(ngx_file_t *file, u_char *buf, size_t size,
130      off_t offset);
131
132  ssize_t ngx_write_chain_to_file(ngx_file_t *file, ngx_chain_t *ce,
133      off_t offset, ngx_pool_t *pool);
134
135
136  #define ngx_read_fd              read
137  #define ngx_read_fd_n            "read()"
138
139  /*
140  * we use inlined function instead of simple #define
141  * because glibc 2.3 sets warn_unused_result attribute for write()
142  * and in this case gcc 4.3 ignores (void) cast
143  */
144  static ngx_inline ssize_t
145  ngx_write_fd(ngx_fd_t fd, void *buf, size_t n)
146  {
```

```
    return write(fd, buf, n);
}

#define ngx_write_fd_n           "write()"


#define ngx_write_console        ngx_write_fd


#define ngx_linefeed(p)          *p++ = LF;
#define NGX_LINEFEED_SIZE        1
#define NGX_LINEFEED             "\x0a"


#define ngx_rename_file(o, n)    rename((const char *) o, (const char *) n)
#define ngx_rename_file_n        "rename()"


#define ngx_change_file_access(n, a) chmod((const char *) n, a)
#define ngx_change_file_access_n "chmod()"


ngx_int_t ngx_set_file_time(u_char *name, ngx_fd_t fd, time_t s);
#define ngx_set_file_time_n      "utimes()"


#define ngx_file_info(file, sb)  stat((const char *) file, sb)
#define ngx_file_info_n          "stat()"

#define ngx_fd_info(fd, sb)      fstat(fd, sb)
#define ngx_fd_info_n            "fstat()"

#define ngx_link_info(file, sb)  lstat((const char *) file, sb)
#define ngx_link_info_n          "lstat()"

#define ngx_is_dir(sb)           (S_ISDIR((sb)->st_mode))
#define ngx_is_file(sb)          (S_ISREG((sb)->st_mode))
#define ngx_is_link(sb)          (S_ISLNK((sb)->st_mode))
#define ngx_is_exec(sb)          (((sb)->st_mode & S_IXUSR) == S_IXUSR)
#define ngx_file_access(sb)      ((sb)->st_mode & 0777)
#define ngx_file_size(sb)        (sb)->st_size
#define ngx_file_fs_size(sb)     ngx_max((sb)->st_size, (sb)->st_blocks * 512)
#define ngx_file_mtime(sb)       (sb)->st_mtime
#define ngx_file_uniq(sb)        (sb)->st_ino


ngx_int_t ngx_create_file_mapping(ngx_file_mapping_t *fm);
void ngx_close_file_mapping(ngx_file_mapping_t *fm);


#define ngx_realpath(p, r)       (u_char *) realpath((char *) p, (char *) r)
#define ngx_realpath_n           "realpath()"
#define ngx_getcwd(buf, size)    (getcwd((char *) buf, size) != NULL)
#define ngx_getcwd_n             "getcwd()"
#define ngx_path_separator(c)    ((c) == '/')


#if defined(PATH_MAX)

#define NGX_HAVE_MAX_PATH        1
#define NGX_MAX_PATH             PATH_MAX

#else

#define NGX_MAX_PATH             4096

#endif


#define NGX_DIR_MASK_LEN         0


ngx_int_t ngx_open_dir(ngx_str_t *name, ngx_dir_t *dir);
#define ngx_open_dir_n           "opendir()"
```

```
#define ngx_close_dir(d)         closedir((d)->dir)
#define ngx_close_dir_n          "closedir()"


ngx_int_t ngx_read_dir(ngx_dir_t *dir);
#define ngx_read_dir_n           "readdir()"


#define ngx_create_dir(name, access) mkdir((const char *) name, access)
#define ngx_create_dir_n         "mkdir()"


#define ngx_delete_dir(name)     rmdir((const char *) name)
#define ngx_delete_dir_n         "rmdir()"


#define ngx_dir_access(a)        (a | (a & 0444) >> 2)


#define ngx_de_name(dir)         ((u_char *) (dir)->de->d_name)
#if (NGX_HAVE_D_NAMLEN)
#define ngx_de_namelen(dir)      (dir)->de->d_namlen
#else
#define ngx_de_namelen(dir)      ngx_strlen((dir)->de->d_name)
#endif

static ngx_inline ngx_int_t
ngx_de_info(u_char *name, ngx_dir_t *dir)
{
    dir->type = 0;
    return stat((const char *) name, &dir->info);
}

#define ngx_de_info_n            "stat()"
#define ngx_de_link_info(name, dir)  lstat((const char *) name, &(dir)->info)
#define ngx_de_link_info_n       "lstat()"

#if (NGX_HAVE_D_TYPE)

/*
* some file systems (e.g. XFS on Linux and CD9660 on FreeBSD)
* do not set dirent.d_type
*/

#define ngx_de_is_dir(dir)                                                   \
    (((dir)->type) ? ((dir)->type == DT_DIR) : (S_ISDIR((dir)->info.st_mode)))
#define ngx_de_is_file(dir)                                                  \
    (((dir)->type) ? ((dir)->type == DT_REG) : (S_ISREG((dir)->info.st_mode)))
#define ngx_de_is_link(dir)                                                  \
    (((dir)->type) ? ((dir)->type == DT_LNK) : (S_ISLNK((dir)->info.st_mode)))

#else

#define ngx_de_is_dir(dir)       (S_ISDIR((dir)->info.st_mode))
#define ngx_de_is_file(dir)      (S_ISREG((dir)->info.st_mode))
#define ngx_de_is_link(dir)      (S_ISLNK((dir)->info.st_mode))

#endif

#define ngx_de_access(dir)       (((dir)->info.st_mode) & 0777)
#define ngx_de_size(dir)         (dir)->info.st_size
#define ngx_de_fs_size(dir)                                                  \
    ngx_max((dir)->info.st_size, (dir)->info.st_blocks * 512)
#define ngx_de_mtime(dir)        (dir)->info.st_mtime


ngx_int_t ngx_open_glob(ngx_glob_t *gl);
#define ngx_open_glob_n          "glob()"
ngx_int_t ngx_read_glob(ngx_glob_t *gl, ngx_str_t *name);
void ngx_close_glob(ngx_glob_t *gl);


ngx_err_t ngx_trylock_fd(ngx_fd_t fd);
ngx_err_t ngx_lock_fd(ngx_fd_t fd);
ngx_err_t ngx_unlock_fd(ngx_fd_t fd);

```

```
#define ngx_trylock_fd_n          "fcntl(F_SETLK, F_WRLCK)"
#define ngx_lock_fd_n             "fcntl(F_SETLKW, F_WRLCK)"
#define ngx_unlock_fd_n           "fcntl(F_SETLK, F_UNLCK)"


#if (NGX_HAVE_F_READAHEAD)

#define NGX_HAVE_READ_AHEAD       1

#define ngx_read_ahead(fd, n)     fcntl(fd, F_READAHEAD, (int) n)
#define ngx_read_ahead_n          "fcntl(fd, F_READAHEAD)"

#elif (NGX_HAVE_POSIX_FADVISE)

#define NGX_HAVE_READ_AHEAD       1

ngx_int_t ngx_read_ahead(ngx_fd_t fd, size_t n);
#define ngx_read_ahead_n          "posix_fadvise(POSIX_FADV_SEQUENTIAL)"

#else

#define ngx_read_ahead(fd, n)     0
#define ngx_read_ahead_n          "ngx_read_ahead_n"

#endif


#if (NGX_HAVE_O_DIRECT)

ngx_int_t ngx_directio_on(ngx_fd_t fd);
#define ngx_directio_on_n         "fcntl(O_DIRECT)"

ngx_int_t ngx_directio_off(ngx_fd_t fd);
#define ngx_directio_off_n        "fcntl(!O_DIRECT)"

#elif (NGX_HAVE_F_NOCACHE)

#define ngx_directio_on(fd)       fcntl(fd, F_NOCACHE, 1)
#define ngx_directio_on_n         "fcntl(F_NOCACHE, 1)"

#elif (NGX_HAVE_DIRECTIO)

#define ngx_directio_on(fd)       directio(fd, DIRECTIO_ON)
#define ngx_directio_on_n         "directio(DIRECTIO_ON)"

#else

#define ngx_directio_on(fd)       0
#define ngx_directio_on_n         "ngx_directio_on_n"

#endif

size_t ngx_fs_bsize(u_char *name);


#if (NGX_HAVE_OPENAT)

#define ngx_openat_file(fd, name, mode, create, access)                    \
    openat(fd, (const char *) name, mode|create, access)

#define ngx_openat_file_n         "openat()"

#define ngx_file_at_info(fd, name, sb, flag)                               \
    fstatat(fd, (const char *) name, sb, flag)

#define ngx_file_at_info_n        "fstatat()"

#define NGX_AT_FDCWD              (ngx_fd_t) AT_FDCWD

#endif


#define ngx_stderr                STDERR_FILENO
#define ngx_set_stderr(fd)        dup2(fd, STDERR_FILENO)
#define ngx_set_stderr_n          "dup2(STDERR_FILENO)"

```

```
375
376 #if (NGX_HAVE_FILE_AIO)
377
378 ssize_t ngx_file_aio_read(ngx_file_t *file, u_char *buf, size_t size,
379     off_t offset, ngx_pool_t *pool);
380
381 extern ngx_uint_t  ngx_file_aio;
382
383 #endif
384
385
386 #endif /* _NGX_FILES_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_files.c - nginx-1.7.10

Global variables defined

- ngx_file_aio

Functions defined

- ngx_close_file_mapping
- ngx_close_glob
- ngx_create_file_mapping
- ngx_directio_off
- ngx_directio_on
- ngx_fs_bsize
- ngx_fs_bsize
- ngx_fs_bsize
- ngx_lock_fd
- ngx_open_dir
- ngx_open_glob
- ngx_open_tempfile
- ngx_read_ahead
- ngx_read_dir
- ngx_read_file
- ngx_read_glob
- ngx_set_file_time
- ngx_trylock_fd
- ngx_unlock_fd
- ngx_write_chain_to_file
- ngx_write_file

Macros defined

- NGX_IOVS

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
```

```c


#include <ngx_config.h>
#include <ngx_core.h>


#if (NGX_HAVE_FILE_AIO)

ngx_uint_t  ngx_file_aio = 1;

#endif


ssize_t
ngx_read_file(ngx_file_t *file, u_char *buf, size_t size, off_t offset)
{
    ssize_t  n;

    ngx_log_debug4(NGX_LOG_DEBUG_CORE, file->log, 0,
                   "read: %d, %p, %uz, %O", file->fd, buf, size, offset);

#if (NGX_HAVE_PREAD)

    n = pread(file->fd, buf, size, offset);

    if (n == -1) {
        ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                      "pread() \"%s\" failed", file->name.data);
        return NGX_ERROR;
    }

#else

    if (file->sys_offset != offset) {
        if (lseek(file->fd, offset, SEEK_SET) == -1) {
            ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                          "lseek() \"%s\" failed", file->name.data);
            return NGX_ERROR;
        }

        file->sys_offset = offset;
    }

    n = read(file->fd, buf, size);

    if (n == -1) {
        ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                      "read() \"%s\" failed", file->name.data);
        return NGX_ERROR;
    }

    file->sys_offset += n;

#endif

    file->offset += n;

    return n;
}


ssize_t
ngx_write_file(ngx_file_t *file, u_char *buf, size_t size, off_t offset)
{
    ssize_t  n, written;

    ngx_log_debug4(NGX_LOG_DEBUG_CORE, file->log, 0,
                   "write: %d, %p, %uz, %O", file->fd, buf, size, offset);

    written = 0;

#if (NGX_HAVE_PWRITE)

    for ( ;; ) {
        n = pwrite(file->fd, buf + written, size, offset);

```

```
        if (n == -1) {
            ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                          "pwrite() \"%s\" failed", file->name.data);
            return NGX_ERROR;
        }

        file->offset += n;
        written += n;

        if ((size_t) n == size) {
            return written;
        }

        offset += n;
        size -= n;
    }

#else

    if (file->sys_offset != offset) {
        if (lseek(file->fd, offset, SEEK_SET) == -1) {
            ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                          "lseek() \"%s\" failed", file->name.data);
            return NGX_ERROR;
        }

        file->sys_offset = offset;
    }

    for ( ;; ) {
        n = write(file->fd, buf + written, size);

        if (n == -1) {
            ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                          "write() \"%s\" failed", file->name.data);
            return NGX_ERROR;
        }

        file->offset += n;
        written += n;

        if ((size_t) n == size) {
            return written;
        }

        size -= n;
    }
#endif
}


ngx_fd_t
ngx_open_tempfile(u_char *name, ngx_uint_t persistent, ngx_uint_t access)
{
    ngx_fd_t  fd;

    fd = open((const char *) name, O_CREAT|O_EXCL|O_RDWR,
              access ? access : 0600);

    if (fd != -1 && !persistent) {
        (void) unlink((const char *) name);
    }

    return fd;
}


#define NGX_IOVS  8

ssize_t
ngx_write_chain_to_file(ngx_file_t *file, ngx_chain_t *cl, off_t offset,
    ngx_pool_t *pool)
{
    u_char        *prev;
    size_t         size;
    ssize_t        total, n;
```

```
    ngx_array_t    vec;
    struct iovec  *iov, iovs[NGX_IOVS];

    /* use pwrite() if there is the only buf in a chain */

    if (cl->next == NULL) {
        return ngx_write_file(file, cl->buf->pos,
                              (size_t) (cl->buf->last - cl->buf->pos),
                              offset);
    }

    total = 0;

    vec.elts = iovs;
    vec.size = sizeof(struct iovec);
    vec.nalloc = NGX_IOVS;
    vec.pool = pool;

    do {
        prev = NULL;
        iov = NULL;
        size = 0;

        vec.nelts = 0;

        /* create the iovec and coalesce the neighbouring bufs */

        while (cl && vec.nelts < IOV_MAX) {
            if (prev == cl->buf->pos) {
                iov->iov_len += cl->buf->last - cl->buf->pos;

            } else {
                iov = ngx_array_push(&vec);
                if (iov == NULL) {
                    return NGX_ERROR;
                }

                iov->iov_base = (void *) cl->buf->pos;
                iov->iov_len = cl->buf->last - cl->buf->pos;
            }

            size += cl->buf->last - cl->buf->pos;
            prev = cl->buf->last;
            cl = cl->next;
        }

        /* use pwrite() if there is the only iovec buffer */

        if (vec.nelts == 1) {
            iov = vec.elts;

            n = ngx_write_file(file, (u_char *) iov[0].iov_base,
                               iov[0].iov_len, offset);

            if (n == NGX_ERROR) {
                return n;
            }

            return total + n;
        }

        if (file->sys_offset != offset) {
            if (lseek(file->fd, offset, SEEK_SET) == -1) {
                ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                              "lseek() \"%s\" failed", file->name.data);
                return NGX_ERROR;
            }

            file->sys_offset = offset;
        }

        n = writev(file->fd, vec.elts, vec.nelts);

        if (n == -1) {
            ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                          "writev() \"%s\" failed", file->name.data);
```

```
            return NGX_ERROR;
        }

        if ((size_t) n != size) {
            ngx_log_error(NGX_LOG_CRIT, file->log, 0,
                          "writev() \"%s\" has written only %z of %uz",
                          file->name.data, n, size);
            return NGX_ERROR;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, file->log, 0,
                       "writev: %d, %z", file->fd, n);

        file->sys_offset += n;
        file->offset += n;
        offset += n;
        total += n;

    } while (cl);

    return total;
}


ngx_int_t
ngx_set_file_time(u_char *name, ngx_fd_t fd, time_t s)
{
    struct timeval  tv[2];

    tv[0].tv_sec = ngx_time();
    tv[0].tv_usec = 0;
    tv[1].tv_sec = s;
    tv[1].tv_usec = 0;

    if (utimes((char *) name, tv) != -1) {
        return NGX_OK;
    }

    return NGX_ERROR;
}


ngx_int_t
ngx_create_file_mapping(ngx_file_mapping_t *fm)
{
    fm->fd = ngx_open_file(fm->name, NGX_FILE_RDWR, NGX_FILE_TRUNCATE,
                           NGX_FILE_DEFAULT_ACCESS);
    if (fm->fd == NGX_INVALID_FILE) {
        ngx_log_error(NGX_LOG_CRIT, fm->log, ngx_errno,
                      ngx_open_file_n " \"%s\" failed", fm->name);
        return NGX_ERROR;
    }

    if (ftruncate(fm->fd, fm->size) == -1) {
        ngx_log_error(NGX_LOG_CRIT, fm->log, ngx_errno,
                      "ftruncate() \"%s\" failed", fm->name);
        goto failed;
    }

    fm->addr = mmap(NULL, fm->size, PROT_READ|PROT_WRITE, MAP_SHARED,
                    fm->fd, 0);
    if (fm->addr != MAP_FAILED) {
        return NGX_OK;
    }

    ngx_log_error(NGX_LOG_CRIT, fm->log, ngx_errno,
                  "mmap(%uz) \"%s\" failed", fm->size, fm->name);

failed:

    if (ngx_close_file(fm->fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, fm->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", fm->name);
    }

    return NGX_ERROR;
```

```
}


void
ngx_close_file_mapping(ngx_file_mapping_t *fm)
{
    if (munmap(fm->addr, fm->size) == -1) {
        ngx_log_error(NGX_LOG_CRIT, fm->log, ngx_errno,
                      "munmap(%uz) \"%s\" failed", fm->size, fm->name);
    }

    if (ngx_close_file(fm->fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, fm->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", fm->name);
    }
}


ngx_int_t
ngx_open_dir(ngx_str_t *name, ngx_dir_t *dir)
{
    dir->dir = opendir((const char *) name->data);

    if (dir->dir == NULL) {
        return NGX_ERROR;
    }

    dir->valid_info = 0;

    return NGX_OK;
}


ngx_int_t
ngx_read_dir(ngx_dir_t *dir)
{
    dir->de = readdir(dir->dir);

    if (dir->de) {
#if (NGX_HAVE_D_TYPE)
        dir->type = dir->de->d_type;
#else
        dir->type = 0;
#endif
        return NGX_OK;
    }

    return NGX_ERROR;
}


ngx_int_t
ngx_open_glob(ngx_glob_t *gl)
{
    int  n;

    n = glob((char *) gl->pattern, 0, NULL, &gl->pglob);

    if (n == 0) {
        return NGX_OK;
    }

#ifdef GLOB_NOMATCH

    if (n == GLOB_NOMATCH && gl->test) {
        return NGX_OK;
    }

#endif

    return NGX_ERROR;
}


ngx_int_t
ngx_read_glob(ngx_glob_t *gl, ngx_str_t *name)
```

```
{
    size_t  count;

#ifdef GLOB_NOMATCH
    count = (size_t) gl->pglob.gl_pathc;
#else
    count = (size_t) gl->pglob.gl_matchc;
#endif

    if (gl->n < count) {

        name->len = (size_t) ngx_strlen(gl->pglob.gl_pathv[gl->n]);
        name->data = (u_char *) gl->pglob.gl_pathv[gl->n];
        gl->n++;

        return NGX_OK;
    }

    return NGX_DONE;
}


void
ngx_close_glob(ngx_glob_t *gl)
{
    globfree(&gl->pglob);
}


ngx_err_t
ngx_trylock_fd(ngx_fd_t fd)
{
    struct flock  fl;

    ngx_memzero(&fl, sizeof(struct flock));
    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;

    if (fcntl(fd, F_SETLK, &fl) == -1) {
        return ngx_errno;
    }

    return 0;
}


ngx_err_t
ngx_lock_fd(ngx_fd_t fd)
{
    struct flock  fl;

    ngx_memzero(&fl, sizeof(struct flock));
    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;

    if (fcntl(fd, F_SETLKW, &fl) == -1) {
        return ngx_errno;
    }

    return 0;
}


ngx_err_t
ngx_unlock_fd(ngx_fd_t fd)
{
    struct flock  fl;

    ngx_memzero(&fl, sizeof(struct flock));
    fl.l_type = F_UNLCK;
    fl.l_whence = SEEK_SET;

    if (fcntl(fd, F_SETLK, &fl) == -1) {
        return  ngx_errno;
    }

```

```
    return 0;
}


#if (NGX_HAVE_POSIX_FADVISE) && !(NGX_HAVE_F_READAHEAD)

ngx_int_t
ngx_read_ahead(ngx_fd_t fd, size_t n)
{
    int  err;

    err = posix_fadvise(fd, 0, 0, POSIX_FADV_SEQUENTIAL);

    if (err == 0) {
        return 0;
    }

    ngx_set_errno(err);
    return NGX_FILE_ERROR;
}

#endif


#if (NGX_HAVE_O_DIRECT)

ngx_int_t
ngx_directio_on(ngx_fd_t fd)
{
    int  flags;

    flags = fcntl(fd, F_GETFL);

    if (flags == -1) {
        return NGX_FILE_ERROR;
    }

    return fcntl(fd, F_SETFL, flags | O_DIRECT);
}


ngx_int_t
ngx_directio_off(ngx_fd_t fd)
{
    int  flags;

    flags = fcntl(fd, F_GETFL);

    if (flags == -1) {
        return NGX_FILE_ERROR;
    }

    return fcntl(fd, F_SETFL, flags & ~O_DIRECT);
}

#endif


#if (NGX_HAVE_STATFS)

size_t
ngx_fs_bsize(u_char *name)
{
    struct statfs  fs;

    if (statfs((char *) name, &fs) == -1) {
        return 512;
    }

    if ((fs.f_bsize % 512) != 0) {
        return 512;
    }

    return (size_t) fs.f_bsize;
}

```

```
538 #elif (NGX_HAVE_STATVFS)
539
540 size_t
541 ngx_fs_bsize(u_char *name)
542 {
543     struct statvfs  fs;
544
545     if (statvfs((char *) name, &fs) == -1) {
546         return 512;
547     }
548
549     if ((fs.f_frsize % 512) != 0) {
550         return 512;
551     }
552
553     return (size_t) fs.f_frsize;
554 }
555
556 #else
557
558 size_t
559 ngx_fs_bsize(u_char *name)
560 {
561     return 512;
562 }
563
564 #endif
```

One Level Up Top Level

src/core/ngx_log.h - nginx-1.7.10

Data types defined

- ngx_log_handler_pt
- ngx_log_s
- ngx_log_writer_pt

Functions defined

- ngx_write_stderr

Macros defined

- NGX_HAVE_VARIADIC_MACROS
- NGX_HAVE_VARIADIC_MACROS
- NGX_HAVE_VARIADIC_MACROS
- NGX_LOG_ALERT
- NGX_LOG_CRIT
- NGX_LOG_DEBUG
- NGX_LOG_DEBUG_ALL
- NGX_LOG_DEBUG_ALLOC
- NGX_LOG_DEBUG_CONNECTION
- NGX_LOG_DEBUG_CORE
- NGX_LOG_DEBUG_EVENT
- NGX_LOG_DEBUG_FIRST
- NGX_LOG_DEBUG_HTTP
- NGX_LOG_DEBUG_LAST
- NGX_LOG_DEBUG_MAIL
- NGX_LOG_DEBUG_MUTEX
- NGX_LOG_DEBUG_MYSQL
- NGX_LOG_EMERG
- NGX_LOG_ERR
- NGX_LOG_INFO
- NGX_LOG_NOTICE
- NGX_LOG_STDERR
- NGX_LOG_WARN

- NGX_MAX_ERROR_STR
- _NGX_LOG_H_INCLUDED_
- ngx_log_debug
- ngx_log_debug
- ngx_log_debug0
- ngx_log_debug0
- ngx_log_debug0
- ngx_log_debug1
- ngx_log_debug1
- ngx_log_debug1
- ngx_log_debug2
- ngx_log_debug2
- ngx_log_debug2
- ngx_log_debug3
- ngx_log_debug3
- ngx_log_debug3
- ngx_log_debug4
- ngx_log_debug4
- ngx_log_debug4
- ngx_log_debug5
- ngx_log_debug5
- ngx_log_debug5
- ngx_log_debug6
- ngx_log_debug6
- ngx_log_debug6
- ngx_log_debug7
- ngx_log_debug7
- ngx_log_debug7
- ngx_log_debug8
- ngx_log_debug8
- ngx_log_debug8
- ngx_log_error
- ngx_log_error

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_LOG_H_INCLUDED_
9  #define _NGX_LOG_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #define NGX_LOG_STDERR            0
17  #define NGX_LOG_EMERG             1
18  #define NGX_LOG_ALERT             2
19  #define NGX_LOG_CRIT              3
20  #define NGX_LOG_ERR               4
21  #define NGX_LOG_WARN              5
22  #define NGX_LOG_NOTICE            6
23  #define NGX_LOG_INFO              7
24  #define NGX_LOG_DEBUG             8
25
26  #define NGX_LOG_DEBUG_CORE        0x010
27  #define NGX_LOG_DEBUG_ALLOC       0x020
28  #define NGX_LOG_DEBUG_MUTEX       0x040
29  #define NGX_LOG_DEBUG_EVENT       0x080
30  #define NGX_LOG_DEBUG_HTTP        0x100
31  #define NGX_LOG_DEBUG_MAIL        0x200
32  #define NGX_LOG_DEBUG_MYSQL       0x400
33
34  /*
35   * do not forget to update debug_levels[] in src/core/ngx_log.c
36   * after the adding a new debug level
37   */
38
39  #define NGX_LOG_DEBUG_FIRST       NGX_LOG_DEBUG_CORE
40  #define NGX_LOG_DEBUG_LAST        NGX_LOG_DEBUG_MYSQL
41  #define NGX_LOG_DEBUG_CONNECTION  0x80000000
42  #define NGX_LOG_DEBUG_ALL         0x7ffffff0
43
44
45  typedef u_char *(*ngx_log_handler_pt) (ngx_log_t *log, u_char *buf, size_t len);
46  typedef void (*ngx_log_writer_pt) (ngx_log_t *log, ngx_uint_t level,
47      u_char *buf, size_t len);
48
49
50  struct ngx_log_s {
51      ngx_uint_t           log_level;
52      ngx_open_file_t     *file;
53
54      ngx_atomic_uint_t    connection;
55
56      time_t               disk_full_time;
57
58      ngx_log_handler_pt   handler;
59      void                *data;
60
61      ngx_log_writer_pt    writer;
62      void                *wdata;
63
64      /*
65       * we declare "action" as "char *" because the actions are usually
66       * the static strings and in the "u_char *" case we have to override
67       * their types all the time
68       */
69
70      char                *action;
71
72      ngx_log_t           *next;
73  };
```

```
74
75
76  #define NGX_MAX_ERROR_STR   2048
77
78
79  /*********************************/
80
81  #if (NGX_HAVE_C99_VARIADIC_MACROS)
82
83  #define NGX_HAVE_VARIADIC_MACROS  1
84
85  #define ngx_log_error(level, log, ...)                                       \
86      if ((log)->log_level >= level) ngx_log_error_core(level, log, __VA_ARGS__)
87
88  void ngx_log_error_core(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
89      const char *fmt, ...);
90
91  #define ngx_log_debug(level, log, ...)                                       \
92      if ((log)->log_level & level)                                            \
93          ngx_log_error_core(NGX_LOG_DEBUG, log, __VA_ARGS__)
94
95  /*********************************/
96
97  #elif (NGX_HAVE_GCC_VARIADIC_MACROS)
98
99  #define NGX_HAVE_VARIADIC_MACROS  1
100
101  #define ngx_log_error(level, log, args...)                                   \
102      if ((log)->log_level >= level) ngx_log_error_core(level, log, args)
103
104  void ngx_log_error_core(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
105      const char *fmt, ...);
106
107  #define ngx_log_debug(level, log, args...)                                   \
108      if ((log)->log_level & level)                                            \
109          ngx_log_error_core(NGX_LOG_DEBUG, log, args)
110
111  /*********************************/
112
113  #else /* NO VARIADIC MACROS */
114
115  #define NGX_HAVE_VARIADIC_MACROS  0
116
117  void ngx_cdecl ngx_log_error(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
118      const char *fmt, ...);
119  void ngx_log_error_core(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
120      const char *fmt, va_list args);
121  void ngx_cdecl ngx_log_debug_core(ngx_log_t *log, ngx_err_t err,
122      const char *fmt, ...);
123
124
125  #endif /* VARIADIC MACROS */
126
127
128  /*********************************/
129
130  #if (NGX_DEBUG)
131
132  #if (NGX_HAVE_VARIADIC_MACROS)
133
134  #define ngx_log_debug0(level, log, err, fmt)                                 \
135          ngx_log_debug(level, log, err, fmt)
136
137  #define ngx_log_debug1(level, log, err, fmt, arg1)                           \
138          ngx_log_debug(level, log, err, fmt, arg1)
139
140  #define ngx_log_debug2(level, log, err, fmt, arg1, arg2)                     \
141          ngx_log_debug(level, log, err, fmt, arg1, arg2)
142
143  #define ngx_log_debug3(level, log, err, fmt, arg1, arg2, arg3)               \
144          ngx_log_debug(level, log, err, fmt, arg1, arg2, arg3)
145
146  #define ngx_log_debug4(level, log, err, fmt, arg1, arg2, arg3, arg4)         \
147          ngx_log_debug(level, log, err, fmt, arg1, arg2, arg3, arg4)
148
149  #define ngx_log_debug5(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5)   \
```

```
        ngx_log_debug(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5)

#define ngx_log_debug6(level, log, err, fmt,                                 \
                       arg1, arg2, arg3, arg4, arg5, arg6)                   \
        ngx_log_debug(level, log, err, fmt,                                  \
                       arg1, arg2, arg3, arg4, arg5, arg6)

#define ngx_log_debug7(level, log, err, fmt,                                 \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7)             \
        ngx_log_debug(level, log, err, fmt,                                  \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7)

#define ngx_log_debug8(level, log, err, fmt,                                 \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8)       \
        ngx_log_debug(level, log, err, fmt,                                  \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8)


#else /* NO VARIADIC MACROS */

#define ngx_log_debug0(level, log, err, fmt)                                 \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt)

#define ngx_log_debug1(level, log, err, fmt, arg1)                           \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt, arg1)

#define ngx_log_debug2(level, log, err, fmt, arg1, arg2)                     \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt, arg1, arg2)

#define ngx_log_debug3(level, log, err, fmt, arg1, arg2, arg3)               \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt, arg1, arg2, arg3)

#define ngx_log_debug4(level, log, err, fmt, arg1, arg2, arg3, arg4)         \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt, arg1, arg2, arg3, arg4)

#define ngx_log_debug5(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5)   \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt, arg1, arg2, arg3, arg4, arg5)

#define ngx_log_debug6(level, log, err, fmt,                                 \
                       arg1, arg2, arg3, arg4, arg5, arg6)                   \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt, arg1, arg2, arg3, arg4, arg5, arg6)

#define ngx_log_debug7(level, log, err, fmt,                                 \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7)             \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt,                                    \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7)

#define ngx_log_debug8(level, log, err, fmt,                                 \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8)       \
    if ((log)->log_level & level)                                            \
        ngx_log_debug_core(log, err, fmt,                                    \
                       arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8)

#endif

#else /* NO NGX_DEBUG */

#define ngx_log_debug0(level, log, err, fmt)
#define ngx_log_debug1(level, log, err, fmt, arg1)
#define ngx_log_debug2(level, log, err, fmt, arg1, arg2)
#define ngx_log_debug3(level, log, err, fmt, arg1, arg2, arg3)
#define ngx_log_debug4(level, log, err, fmt, arg1, arg2, arg3, arg4)
#define ngx_log_debug5(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5)
#define ngx_log_debug6(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5, arg6)
#define ngx_log_debug7(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5,   \
                       arg6, arg7)
#define ngx_log_debug8(level, log, err, fmt, arg1, arg2, arg3, arg4, arg5,   \
                       arg6, arg7, arg8)
```

```
226
227 #endif
228
229 /*********************************/
230
231 ngx_log_t *ngx_log_init(u_char *prefix);
232 void ngx_cdecl ngx_log_abort(ngx_err_t err, const char *fmt, ...);
233 void ngx_cdecl ngx_log_stderr(ngx_err_t err, const char *fmt, ...);
234 u_char *ngx_log_errno(u_char *buf, u_char *last, ngx_err_t err);
235 ngx_int_t ngx_log_open_default(ngx_cycle_t *cycle);
236 ngx_int_t ngx_log_redirect_stderr(ngx_cycle_t *cycle);
237 ngx_log_t *ngx_log_get_file_log(ngx_log_t *head);
238 char *ngx_log_set_log(ngx_conf_t *cf, ngx_log_t **head);
239
240
241 /*
242  * ngx_write_stderr() cannot be implemented as macro, since
243  * MSVC does not allow to use #ifdef inside macro parameters.
244  *
245  * ngx_write_fd() is used instead of ngx_write_console(), since
246  * CharToOemBuff() inside ngx_write_console() cannot be used with
247  * read only buffer as destination and CharToOemBuff() is not needed
248  * for ngx_write_stderr() anyway.
249  */
250 static ngx_inline void
251 ngx_write_stderr(char *text)
252 {
253     (void) ngx_write_fd(ngx_stderr, text, ngx_strlen(text));
254 }
255
256
257 extern ngx_module_t  ngx_errlog_module;
258 extern ngx_uint_t    ngx_use_stderr;
259
260
261 #endif /* _NGX_LOG_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_log.c - nginx-1.7.10

Global variables defined

- debug_levels
- err_levels
- ngx_errlog_commands
- ngx_errlog_module
- ngx_errlog_module_ctx
- ngx_log
- ngx_log_file
- ngx_use_stderr

Functions defined

- ngx_error_log
- ngx_log_abort
- ngx_log_debug_core
- ngx_log_errno
- ngx_log_error
- ngx_log_error_core
- ngx_log_get_file_log
- ngx_log_init
- ngx_log_insert
- ngx_log_open_default
- ngx_log_redirect_stderr
- ngx_log_set_levels
- ngx_log_set_log
- ngx_log_stderr

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
```

```
12 static char *ngx_error_log(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
13 static char *ngx_log_set_levels(ngx_conf_t *cf, ngx_log_t *log);
14 static void ngx_log_insert(ngx_log_t *log, ngx_log_t *new_log);
15
16
17 static ngx_command_t  ngx_errlog_commands[] = {
18
19     {ngx_string("error_log"),
20      NGX_MAIN_CONF|NGX_CONF_1MORE,
21      ngx_error_log,
22      0,
23      0,
24      NULL},
25
26     ngx_null_command
27 };
28
29
30 static ngx_core_module_t  ngx_errlog_module_ctx = {
31     ngx_string("errlog"),
32     NULL,
33     NULL
34 };
35
36
37 ngx_module_t  ngx_errlog_module = {
38     NGX_MODULE_V1,
39     &ngx_errlog_module_ctx,                /* module context */
40     ngx_errlog_commands,                   /* module directives */
41     NGX_CORE_MODULE,                       /* module type */
42     NULL,                                  /* init master */
43     NULL,                                  /* init module */
44     NULL,                                  /* init process */
45     NULL,                                  /* init thread */
46     NULL,                                  /* exit thread */
47     NULL,                                  /* exit process */
48     NULL,                                  /* exit master */
49     NGX_MODULE_V1_PADDING
50 };
51
52
53 static ngx_log_t        ngx_log;
54 static ngx_open_file_t  ngx_log_file;
55 ngx_uint_t              ngx_use_stderr = 1;
56
57
58 static ngx_str_t err_levels[] = {
59     ngx_null_string,
60     ngx_string("emerg"),
61     ngx_string("alert"),
62     ngx_string("crit"),
63     ngx_string("error"),
64     ngx_string("warn"),
65     ngx_string("notice"),
66     ngx_string("info"),
67     ngx_string("debug")
68 };
69
70 static const char *debug_levels[] = {
71     "debug_core", "debug_alloc", "debug_mutex", "debug_event",
72     "debug_http", "debug_mail", "debug_mysql"
73 };
74
75
76 #if (NGX_HAVE_VARIADIC_MACROS)
77
78 void
79 ngx_log_error_core(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
80     const char *fmt, ...)
81
82 #else
83
84 void
85 ngx_log_error_core(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
86     const char *fmt, va_list args)
87
```

```
#endif
{
#if (NGX_HAVE_VARIADIC_MACROS)
    va_list      args;
#endif
    u_char      *p, *last, *msg;
    ssize_t      n;
    ngx_uint_t   wrote_stderr, debug_connection;
    u_char       errstr[NGX_MAX_ERROR_STR];

    last = errstr + NGX_MAX_ERROR_STR;

     ngx_memcpy(errstr, ngx_cached_err_log_time.data,
                ngx_cached_err_log_time.len);

     p = errstr + ngx_cached_err_log_time.len;

     p = ngx_slprintf(p, last, " [%V] ", &err_levels[level]);

     /* pid#tid */
     p = ngx_slprintf(p, last, "%P#" NGX_TID_T_FMT ": ",
                      ngx_log_pid, ngx_log_tid);

     if (log->connection) {
         p = ngx_slprintf(p, last, "*%uA ", log->connection);
     }

     msg = p;

#if (NGX_HAVE_VARIADIC_MACROS)

     va_start(args, fmt);
     p = ngx_vslprintf(p, last, fmt, args);
     va_end(args);

#else

     p = ngx_vslprintf(p, last, fmt, args);

#endif

     if (err) {
         p = ngx_log_errno(p, last, err);
     }

     if (level != NGX_LOG_DEBUG && log->handler) {
         p = log->handler(log, p, last - p);
     }

     if (p > last - NGX_LINEFEED_SIZE) {
         p = last - NGX_LINEFEED_SIZE;
     }

     ngx_linefeed(p);

     wrote_stderr = 0;
     debug_connection = (log->log_level & NGX_LOG_DEBUG_CONNECTION) != 0;

     while (log) {

         if (log->log_level < level && !debug_connection) {
             break;
         }

         if (log->writer) {
             log->writer(log, level, errstr, p - errstr);
             goto next;
         }

         if (ngx_time() == log->disk_full_time) {

             /*
              * on FreeBSD writing to a full filesystem with enabled softupdates
              * may block process for much longer time than writing to non-full
              * filesystem, so we skip writing to a log for one second
              */
```

```
            goto next;
        }

        n = ngx_write_fd(log->file->fd, errstr, p - errstr);

        if (n == -1 && ngx_errno == NGX_ENOSPC) {
            log->disk_full_time = ngx_time();
        }

        if (log->file->fd == ngx_stderr) {
            wrote_stderr = 1;
        }

    next:

        log = log->next;
    }

    if (!ngx_use_stderr
        || level > NGX_LOG_WARN
        || wrote_stderr)
    {
        return;
    }

    msg -= (7 + err_levels[level].len + 3);

    (void) ngx_sprintf(msg, "nginx: [%V] ", &err_levels[level]);

    (void) ngx_write_console(ngx_stderr, msg, p - msg);
}


#if !(NGX_HAVE_VARIADIC_MACROS)

void ngx_cdecl
ngx_log_error(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
    const char *fmt, ...)
{
    va_list  args;

    if (log->log_level >= level) {
        va_start(args, fmt);
        ngx_log_error_core(level, log, err, fmt, args);
        va_end(args);
    }
}


void ngx_cdecl
ngx_log_debug_core(ngx_log_t *log, ngx_err_t err, const char *fmt, ...)
{
    va_list  args;

    va_start(args, fmt);
    ngx_log_error_core(NGX_LOG_DEBUG, log, err, fmt, args);
    va_end(args);
}

#endif


void ngx_cdecl
ngx_log_abort(ngx_err_t err, const char *fmt, ...)
{
    u_char   *p;
    va_list   args;
    u_char    errstr[NGX_MAX_CONF_ERRSTR];

    va_start(args, fmt);
    p = ngx_vsnprintf(errstr, sizeof(errstr) - 1, fmt, args);
    va_end(args);

    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, err,
                  "%*s", p - errstr, errstr);
```

```
}


void ngx_cdecl
ngx_log_stderr(ngx_err_t err, const char *fmt, ...)
{
    u_char   *p, *last;
    va_list   args;
    u_char    errstr[NGX_MAX_ERROR_STR];

    last = errstr + NGX_MAX_ERROR_STR;
    p = errstr + 7;

    ngx_memcpy(errstr, "nginx: ", 7);

    va_start(args, fmt);
    p = ngx_vslprintf(p, last, fmt, args);
    va_end(args);

    if (err) {
        p = ngx_log_errno(p, last, err);
    }

    if (p > last - NGX_LINEFEED_SIZE) {
        p = last - NGX_LINEFEED_SIZE;
    }

    ngx_linefeed(p);

    (void) ngx_write_console(ngx_stderr, errstr, p - errstr);
}


u_char *
ngx_log_errno(u_char *buf, u_char *last, ngx_err_t err)
{
    if (buf > last - 50) {

        /* leave a space for an error code */

        buf = last - 50;
        *buf++ = '.';
        *buf++ = '.';
        *buf++ = '.';
    }

#if (NGX_WIN32)
    buf = ngx_slprintf(buf, last, ((unsigned) err < 0x80000000)
                                       ? " (%d: " : " (%Xd: ", err);
#else
    buf = ngx_slprintf(buf, last, " (%d: ", err);
#endif

    buf = ngx_strerror(err, buf, last - buf);

    if (buf < last) {
        *buf++ = ')';
    }

    return buf;
}


ngx_log_t *
ngx_log_init(u_char *prefix)
{
    u_char  *p, *name;
    size_t   nlen, plen;

    ngx_log.file = &ngx_log_file;
    ngx_log.log_level = NGX_LOG_NOTICE;

    name = (u_char *) NGX_ERROR_LOG_PATH;

    /*
     * we use ngx_strlen() here since BCC warns about
```

```
316      * condition is always false and unreachable code
317      */
318
319     nlen = ngx_strlen(name);
320
321     if (nlen == 0) {
322         ngx_log_file.fd = ngx_stderr;
323         return &ngx_log;
324     }
325
326     p = NULL;
327
328 #if (NGX_WIN32)
329     if (name[1] != ':') {
330 #else
331     if (name[0] != '/') {
332 #endif
333
334         if (prefix) {
335             plen = ngx_strlen(prefix);
336
337         } else {
338 #ifdef NGX_PREFIX
339             prefix = (u_char *) NGX_PREFIX;
340             plen = ngx_strlen(prefix);
341 #else
342             plen = 0;
343 #endif
344         }
345
346         if (plen) {
347             name = malloc(plen + nlen + 2);
348             if (name == NULL) {
349                 return NULL;
350             }
351
352             p = ngx_cpymem(name, prefix, plen);
353
354             if (!ngx_path_separator(*(p - 1))) {
355                 *p++ = '/';
356             }
357
358             ngx_cpystrn(p, (u_char *) NGX_ERROR_LOG_PATH, nlen + 1);
359
360             p = name;
361         }
362     }
363
364     ngx_log_file.fd = ngx_open_file(name, NGX_FILE_APPEND,
365                                     NGX_FILE_CREATE_OR_OPEN,
366                                     NGX_FILE_DEFAULT_ACCESS);
367
368     if (ngx_log_file.fd == NGX_INVALID_FILE) {
369         ngx_log_stderr(ngx_errno,
370                        "[alert] could not open error log file: "
371                        ngx_open_file_n " \"%s\" failed", name);
372 #if (NGX_WIN32)
373         ngx_event_log(ngx_errno,
374                       "could not open error log file: "
375                       ngx_open_file_n " \"%s\" failed", name);
376 #endif
377
378         ngx_log_file.fd = ngx_stderr;
379     }
380
381     if (p) {
382         ngx_free(p);
383     }
384
385     return &ngx_log;
386 }
387
388
389 ngx_int_t
390 ngx_log_open_default(ngx_cycle_t *cycle)
391 {
```

```
    ngx_log_t         *log;
    static ngx_str_t   error_log = ngx_string(NGX_ERROR_LOG_PATH);

    if (ngx_log_get_file_log(&cycle->new_log) != NULL) {
        return NGX_OK;
    }

    if (cycle->new_log.log_level != 0) {
        /* there are some error logs, but no files */

        log = ngx_pcalloc(cycle->pool, sizeof(ngx_log_t));
        if (log == NULL) {
            return NGX_ERROR;
        }

    } else {
        /* no error logs at all */
        log = &cycle->new_log;
    }

    log->log_level = NGX_LOG_ERR;

    log->file = ngx_conf_open_file(cycle, &error_log);
    if (log->file == NULL) {
        return NGX_ERROR;
    }

    if (log != &cycle->new_log) {
        ngx_log_insert(&cycle->new_log, log);
    }

    return NGX_OK;
}


ngx_int_t
ngx_log_redirect_stderr(ngx_cycle_t *cycle)
{
    ngx_fd_t  fd;

    if (cycle->log_use_stderr) {
        return NGX_OK;
    }

    /* file log always exists when we are called */
    fd = ngx_log_get_file_log(cycle->log)->file->fd;

    if (fd != ngx_stderr) {
        if (ngx_set_stderr(fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          ngx_set_stderr_n " failed");

            return NGX_ERROR;
        }
    }

    return NGX_OK;
}


ngx_log_t *
ngx_log_get_file_log(ngx_log_t *head)
{
    ngx_log_t  *log;

    for (log = head; log; log = log->next) {
        if (log->file != NULL) {
            return log;
        }
    }

    return NULL;
}


static char *
```

```
ngx_log_set_levels(ngx_conf_t *cf, ngx_log_t *log)
{
    ngx_uint_t   i, n, d, found;
    ngx_str_t   *value;

    if (cf->args->nelts == 2) {
        log->log_level = NGX_LOG_ERR;
        return NGX_CONF_OK;
    }

    value = cf->args->elts;

    for (i = 2; i < cf->args->nelts; i++) {
        found = 0;

        for (n = 1; n <= NGX_LOG_DEBUG; n++) {
            if (ngx_strcmp(value[i].data, err_levels[n].data) == 0) {

                if (log->log_level != 0) {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "duplicate log level \"%V\"",
                                       &value[i]);
                    return NGX_CONF_ERROR;
                }

                log->log_level = n;
                found = 1;
                break;
            }
        }

        for (n = 0, d = NGX_LOG_DEBUG_FIRST; d <= NGX_LOG_DEBUG_LAST; d <<= 1) {
            if (ngx_strcmp(value[i].data, debug_levels[n++]) == 0) {
                if (log->log_level & ~NGX_LOG_DEBUG_ALL) {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "invalid log level \"%V\"",
                                       &value[i]);
                    return NGX_CONF_ERROR;
                }

                log->log_level |= d;
                found = 1;
                break;
            }
        }


        if (!found) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid log level \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }
    }

    if (log->log_level == NGX_LOG_DEBUG) {
        log->log_level = NGX_LOG_DEBUG_ALL;
    }

    return NGX_CONF_OK;
}


static char *
ngx_error_log(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_log_t  *dummy;

    dummy = &cf->cycle->new_log;

    return ngx_log_set_log(cf, &dummy);
}


char *
ngx_log_set_log(ngx_conf_t *cf, ngx_log_t **head)
{
```

```
    ngx_log_t          *new_log;
    ngx_str_t          *value, name;
    ngx_syslog_peer_t  *peer;

    if (*head != NULL && (*head)->log_level == 0) {
        new_log = *head;

    } else {

        new_log = ngx_pcalloc(cf->pool, sizeof(ngx_log_t));
        if (new_log == NULL) {
            return NGX_CONF_ERROR;
        }

        if (*head == NULL) {
            *head = new_log;
        }
    }

    value = cf->args->elts;

    if (ngx_strcmp(value[1].data, "stderr") == 0) {
        ngx_str_null(&name);
        cf->cycle->log_use_stderr = 1;

        new_log->file = ngx_conf_open_file(cf->cycle, &name);
        if (new_log->file == NULL) {
            return NGX_CONF_ERROR;
        }


     } else if (ngx_strncmp(value[1].data, "syslog:", 7) == 0) {
        peer = ngx_pcalloc(cf->pool, sizeof(ngx_syslog_peer_t));
        if (peer == NULL) {
            return NGX_CONF_ERROR;
        }

        if (ngx_syslog_process_conf(cf, peer) != NGX_CONF_OK) {
            return NGX_CONF_ERROR;
        }

        new_log->writer = ngx_syslog_writer;
        new_log->wdata = peer;

    } else {
        new_log->file = ngx_conf_open_file(cf->cycle, &value[1]);
        if (new_log->file == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    if (ngx_log_set_levels(cf, new_log) != NGX_CONF_OK) {
        return NGX_CONF_ERROR;
    }

    if (*head != new_log) {
        ngx_log_insert(*head, new_log);
    }

    return NGX_CONF_OK;
}


static void
ngx_log_insert(ngx_log_t *log, ngx_log_t *new_log)
{
    ngx_log_t  tmp;

    if (new_log->log_level > log->log_level) {

        /*
         * list head address is permanent, insert new log after
         * head and swap its contents with head
         */

        tmp = *log;
```

```
620         *log = *new_log;
621         *new_log = tmp;
622 
623         log->next = new_log;
624         return;
625     }
626 
627     while (log->next) {
628         if (new_log->log_level > log->next->log_level) {
629             new_log->next = log->next;
630             log->next = new_log;
631             return;
632         }
633 
634         log = log->next;
635     }
636 
637     log->next = new_log;
638 }
```

One Level Up Top Level

src/core/ngx_string.h - nginx-1.7.10

Data types defined

- ngx_keyval_t
- ngx_str_node_t
- ngx_str_t
- ngx_variable_value_t

Functions defined

- ngx_copy
- ngx_strlchr

Macros defined

- NGX_ESCAPE_ARGS
- NGX_ESCAPE_HTML
- NGX_ESCAPE_MAIL_AUTH
- NGX_ESCAPE_MEMCACHED
- NGX_ESCAPE_REFRESH
- NGX_ESCAPE_URI
- NGX_ESCAPE_URI_COMPONENT
- NGX_UNESCAPE_REDIRECT
- NGX_UNESCAPE_URI
- _NGX_STRING_H_INCLUDED_
- ngx_base64_decoded_length
- ngx_base64_encoded_length
- ngx_copy
- ngx_cpymem
- ngx_cpymem
- ngx_memcmp
- ngx_memcpy
- ngx_memmove
- ngx_memset
- ngx_memzero
- ngx_movemem

- ngx_null_string
- ngx_qsort
- ngx_str_null
- ngx_str_set
- ngx_strchr
- ngx_strcmp
- ngx_string
- ngx_strlen
- ngx_strncmp
- ngx_strstr
- ngx_tolower
- ngx_toupper
- ngx_value
- ngx_value_helper
- ngx_vsnprintf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_STRING_H_INCLUDED_
9  #define _NGX_STRING_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct {
17      size_t      len;
18      u_char     *data;
19  } ngx_str_t;
20
21
22  typedef struct {
23      ngx_str_t   key;
24      ngx_str_t   value;
25  } ngx_keyval_t;
26
27
28  typedef struct {
29      unsigned    len:28;
30
31      unsigned    valid:1;
32      unsigned    no_cacheable:1;
33      unsigned    not_found:1;
34      unsigned    escape:1;
35
36      u_char     *data;
37  } ngx_variable_value_t;
38
```

```
39
40  #define ngx_string(str)     { sizeof(str) - 1, (u_char *) str }
41  #define ngx_null_string     { 0, NULL }
42  #define ngx_str_set(str, text)                                               \
43      (str)->len = sizeof(text) - 1; (str)->data = (u_char *) text
44  #define ngx_str_null(str)   (str)->len = 0; (str)->data = NULL
45
46
47  #define ngx_tolower(c)      (u_char) ((c >= 'A' && c <= 'Z') ? (c | 0x20) : c)
48  #define ngx_toupper(c)      (u_char) ((c >= 'a' && c <= 'z') ? (c & ~0x20) : c)
49
50  void ngx_strlow(u_char *dst, u_char *src, size_t n);
51
52
53  #define ngx_strncmp(s1, s2, n)  strncmp((const char *) s1, (const char *) s2, n)
54
55
56  /* msvc and icc7 compile strcmp() to inline loop */
57  #define ngx_strcmp(s1, s2)  strcmp((const char *) s1, (const char *) s2)
58
59
60  #define ngx_strstr(s1, s2)  strstr((const char *) s1, (const char *) s2)
61  #define ngx_strlen(s)       strlen((const char *) s)
62
63  #define ngx_strchr(s1, c)   strchr((const char *) s1, (int) c)
64
65  static ngx_inline u_char *
66  ngx_strlchr(u_char *p, u_char *last, u_char c)
67  {
68      while (p < last) {
69
70          if (*p == c) {
71              return p;
72          }
73
74          p++;
75      }
76
77      return NULL;
78  }
79
80
81  /*
82  * msvc and icc7 compile memset() to the inline "rep stos"
83  * while ZeroMemory() and bzero() are the calls.
84  * icc7 may also inline several mov's of a zeroed register for small blocks.
85  */
86  #define ngx_memzero(buf, n)       (void) memset(buf, 0, n)
87  #define ngx_memset(buf, c, n)     (void) memset(buf, c, n)
88
89
90  #if (NGX_MEMCPY_LIMIT)
91
92  void *ngx_memcpy(void *dst, const void *src, size_t n);
93  #define ngx_cpymem(dst, src, n)   (((u_char *) ngx_memcpy(dst, src, n)) + (n))
94
95  #else
96
97  /*
98  * gcc3, msvc, and icc7 compile memcpy() to the inline "rep movs".
99  * gcc3 compiles memcpy(d, s, 4) to the inline "mov"es.
100  * icc8 compile memcpy(d, s, 4) to the inline "mov"es or XMM moves.
101  */
102  #define ngx_memcpy(dst, src, n)   (void) memcpy(dst, src, n)
103  #define ngx_cpymem(dst, src, n)   (((u_char *) memcpy(dst, src, n)) + (n))
104
105  #endif
106
107
108  #if ( __INTEL_COMPILER >= 800 )
109
110  /*
111  * the simple inline cycle copies the variable length strings up to 16
112  * bytes faster than icc8 autodetecting _intel_fast_memcpy()
113  */
114
```

```
static ngx_inline u_char *
ngx_copy(u_char *dst, u_char *src, size_t len)
{
    if (len < 17) {

        while (len) {
            *dst++ = *src++;
            len--;
        }

        return dst;

    } else {
        return ngx_cpymem(dst, src, len);
    }
}

#else

#define ngx_copy                  ngx_cpymem

#endif


#define ngx_memmove(dst, src, n)   (void) memmove(dst, src, n)
#define ngx_movemem(dst, src, n)   (((u_char *) memmove(dst, src, n)) + (n))


/* msvc and icc7 compile memcmp() to the inline loop */
#define ngx_memcmp(s1, s2, n)  memcmp((const char *) s1, (const char *) s2, n)


u_char *ngx_cpystrn(u_char *dst, u_char *src, size_t n);
u_char *ngx_pstrdup(ngx_pool_t *pool, ngx_str_t *src);
u_char * ngx_cdecl ngx_sprintf(u_char *buf, const char *fmt, ...);
u_char * ngx_cdecl ngx_snprintf(u_char *buf, size_t max, const char *fmt, ...);
u_char * ngx_cdecl ngx_slprintf(u_char *buf, u_char *last, const char *fmt,
    ...);
u_char *ngx_vslprintf(u_char *buf, u_char *last, const char *fmt, va_list args);
#define ngx_vsnprintf(buf, max, fmt, args)                                   \
    ngx_vslprintf(buf, buf + (max), fmt, args)

ngx_int_t ngx_strcasecmp(u_char *s1, u_char *s2);
ngx_int_t ngx_strncasecmp(u_char *s1, u_char *s2, size_t n);

u_char *ngx_strnstr(u_char *s1, char *s2, size_t n);

u_char *ngx_strstrn(u_char *s1, char *s2, size_t n);
u_char *ngx_strcasestrn(u_char *s1, char *s2, size_t n);
u_char *ngx_strlcasestrn(u_char *s1, u_char *last, u_char *s2, size_t n);

ngx_int_t ngx_rstrncmp(u_char *s1, u_char *s2, size_t n);
ngx_int_t ngx_rstrncasecmp(u_char *s1, u_char *s2, size_t n);
ngx_int_t ngx_memn2cmp(u_char *s1, u_char *s2, size_t n1, size_t n2);
ngx_int_t ngx_dns_strcmp(u_char *s1, u_char *s2);
ngx_int_t ngx_filename_cmp(u_char *s1, u_char *s2, size_t n);

ngx_int_t ngx_atoi(u_char *line, size_t n);
ngx_int_t ngx_atofp(u_char *line, size_t n, size_t point);
ssize_t ngx_atosz(u_char *line, size_t n);
off_t ngx_atoof(u_char *line, size_t n);
time_t ngx_atotm(u_char *line, size_t n);
ngx_int_t ngx_hextoi(u_char *line, size_t n);

u_char *ngx_hex_dump(u_char *dst, u_char *src, size_t len);


#define ngx_base64_encoded_length(len)  (((len + 2) / 3) * 4)
#define ngx_base64_decoded_length(len)  (((len + 3) / 4) * 3)

void ngx_encode_base64(ngx_str_t *dst, ngx_str_t *src);
void ngx_encode_base64url(ngx_str_t *dst, ngx_str_t *src);
ngx_int_t ngx_decode_base64(ngx_str_t *dst, ngx_str_t *src);
ngx_int_t ngx_decode_base64url(ngx_str_t *dst, ngx_str_t *src);

uint32_t ngx_utf8_decode(u_char **p, size_t n);
```

```
size_t ngx_utf8_length(u_char *p, size_t n);
u_char *ngx_utf8_cpystrn(u_char *dst, u_char *src, size_t n, size_t len);


#define NGX_ESCAPE_URI            0
#define NGX_ESCAPE_ARGS           1
#define NGX_ESCAPE_URI_COMPONENT  2
#define NGX_ESCAPE_HTML           3
#define NGX_ESCAPE_REFRESH        4
#define NGX_ESCAPE_MEMCACHED      5
#define NGX_ESCAPE_MAIL_AUTH      6

#define NGX_UNESCAPE_URI       1
#define NGX_UNESCAPE_REDIRECT  2

uintptr_t ngx_escape_uri(u_char *dst, u_char *src, size_t size,
    ngx_uint_t type);
void ngx_unescape_uri(u_char **dst, u_char **src, size_t size, ngx_uint_t type);
uintptr_t ngx_escape_html(u_char *dst, u_char *src, size_t size);
uintptr_t ngx_escape_json(u_char *dst, u_char *src, size_t size);


typedef struct {
    ngx_rbtree_node_t         node;
    ngx_str_t                 str;
} ngx_str_node_t;


void ngx_str_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
ngx_str_node_t *ngx_str_rbtree_lookup(ngx_rbtree_t *rbtree, ngx_str_t *name,
    uint32_t hash);


void ngx_sort(void *base, size_t n, size_t size,
    ngx_int_t (*cmp)(const void *, const void *));
#define ngx_qsort             qsort


#define ngx_value_helper(n)   #n
#define ngx_value(n)          ngx_value_helper(n)


#endif /* _NGX_STRING_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_string.c - nginx-1.7.10

Functions defined

- ngx_atofp
- ngx_atoi
- ngx_atoof
- ngx_atosz
- ngx_atotm
- ngx_cpystrn
- ngx_decode_base64
- ngx_decode_base64_internal
- ngx_decode_base64url
- ngx_dns_strcmp
- ngx_encode_base64
- ngx_encode_base64_internal
- ngx_encode_base64url
- ngx_escape_html
- ngx_escape_json
- ngx_escape_uri
- ngx_filename_cmp
- ngx_hex_dump
- ngx_hextoi
- ngx_memcpy
- ngx_memn2cmp
- ngx_pstrdup
- ngx_rstrncasecmp
- ngx_rstrncmp
- ngx_slprintf
- ngx_snprintf
- ngx_sort
- ngx_sprintf
- ngx_sprintf_num
- ngx_str_rbtree_insert_value

- ngx_str_rbtree_lookup
- ngx_strcasecmp
- ngx_strcasestrn
- ngx_strlcasestrn
- ngx_strlow
- ngx_strncasecmp
- ngx_strnstr
- ngx_strstrn
- ngx_unescape_uri
- ngx_utf8_cpystrn
- ngx_utf8_decode
- ngx_utf8_length
- ngx_vslprintf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 static u_char *ngx_sprintf_num(u_char *buf, u_char *last, uint64_t ui64,
13     u_char zero, ngx_uint_t hexadecimal, ngx_uint_t width);
14 static void ngx_encode_base64_internal(ngx_str_t *dst, ngx_str_t *src,
15     const u_char *basis, ngx_uint_t padding);
16 static ngx_int_t ngx_decode_base64_internal(ngx_str_t *dst, ngx_str_t *src,
17     const u_char *basis);
18
19
20 void
21 ngx_strlow(u_char *dst, u_char *src, size_t n)
22 {
23     while (n) {
24         *dst = ngx_tolower(*src);
25         dst++;
26         src++;
27         n--;
28     }
29 }
30
31
32 u_char *
33 ngx_cpystrn(u_char *dst, u_char *src, size_t n)
34 {
35     if (n == 0) {
36         return dst;
37     }
38
39     while (--n) {
40         *dst = *src;
41
42         if (*dst == '\0') {
43             return dst;
```

```
        }

        dst++;
        src++;
    }

    *dst = '\0';

    return dst;
}


u_char *
ngx_pstrdup(ngx_pool_t *pool, ngx_str_t *src)
{
    u_char  *dst;

    dst = ngx_pnalloc(pool, src->len);
    if (dst == NULL) {
        return NULL;
    }

    ngx_memcpy(dst, src->data, src->len);

    return dst;
}


/*
* supported formats:
*    %[0][width][x][X]O        off_t
*    %[0][width]T              time_t
*    %[0][width][u][x|X]z      ssize_t/size_t
*    %[0][width][u][x|X]d      int/u_int
*    %[0][width][u][x|X]l      long
*    %[0][width|m][u][x|X]i    ngx_int_t/ngx_uint_t
*    %[0][width][u][x|X]D      int32_t/uint32_t
*    %[0][width][u][x|X]L      int64_t/uint64_t
*    %[0][width|m][u][x|X]A    ngx_atomic_int_t/ngx_atomic_uint_t
*    %[0][width][.width]f      double, max valid number fits to %18.15f
*    %P                        ngx_pid_t
*    %M                        ngx_msec_t
*    %r                        rlim_t
*    %p                        void *
*    %V                        ngx_str_t *
*    %v                        ngx_variable_value_t *
*    %s                        null-terminated string
*    %*s                       length and string
*    %Z                        '\0'
*    %N                        '\n'
*    %c                        char
*    %%                        %
*
*  reserved:
*    %t                        ptrdiff_t
*    %S                        null-terminated wchar string
 *    %C                        wchar
 */


 u_char * ngx_cdecl
 ngx_sprintf(u_char *buf, const char *fmt, ...)
 {
     u_char   *p;
     va_list   args;

     va_start(args, fmt);
     p = ngx_vslprintf(buf, (void *) -1, fmt, args);
     va_end(args);

     return p;
 }


 u_char * ngx_cdecl
 ngx_snprintf(u_char *buf, size_t max, const char *fmt, ...)
```

```
{
    u_char   *p;
    va_list   args;

    va_start(args, fmt);
    p = ngx_vslprintf(buf, buf + max, fmt, args);
    va_end(args);

    return p;
}


u_char * ngx_cdecl
ngx_slprintf(u_char *buf, u_char *last, const char *fmt, ...)
{
    u_char   *p;
    va_list   args;

    va_start(args, fmt);
    p = ngx_vslprintf(buf, last, fmt, args);
    va_end(args);

    return p;
}


u_char *
ngx_vslprintf(u_char *buf, u_char *last, const char *fmt, va_list args)
{
    u_char                *p, zero;
    int                    d;
    double                 f;
    size_t                 len, slen;
    int64_t                i64;
    uint64_t               ui64, frac;
    ngx_msec_t             ms;
    ngx_uint_t             width, sign, hex, max_width, frac_width, scale, n;
    ngx_str_t             *v;
    ngx_variable_value_t  *vv;

    while (*fmt && buf < last) {

        /*
         * "buf < last" means that we could copy at least one character:
         * the plain character, "%%", "%c", and minus without the checking
         */

        if (*fmt == '%') {

            i64 = 0;
            ui64 = 0;

            zero = (u_char) ((*++fmt == '0') ? '0' : ' ');
            width = 0;
            sign = 1;
            hex = 0;
            max_width = 0;
            frac_width = 0;
            slen = (size_t) -1;

            while (*fmt >= '0' && *fmt <= '9') {
                width = width * 10 + *fmt++ - '0';
            }


            for ( ;; ) {
                switch (*fmt) {

                case 'u':
                    sign = 0;
                    fmt++;
                    continue;

                case 'm':
                    max_width = 1;
                    fmt++;
```

```
                continue;

            case 'X':
                hex = 2;
                sign = 0;
                fmt++;
                continue;

            case 'x':
                hex = 1;
                sign = 0;
                fmt++;
                continue;

            case '.':
                fmt++;

                while (*fmt >= '0' && *fmt <= '9') {
                    frac_width = frac_width * 10 + *fmt++ - '0';
                }

                break;

            case '*':
                slen = va_arg(args, size_t);
                fmt++;
                continue;

            default:
                break;
            }

            break;
        }


        switch (*fmt) {

        case 'V':
            v = va_arg(args, ngx_str_t *);

            len = ngx_min(((size_t) (last - buf)), v->len);
            buf = ngx_cpymem(buf, v->data, len);
            fmt++;

            continue;

        case 'v':
            vv = va_arg(args, ngx_variable_value_t *);

            len = ngx_min(((size_t) (last - buf)), vv->len);
            buf = ngx_cpymem(buf, vv->data, len);
            fmt++;

            continue;

        case 's':
            p = va_arg(args, u_char *);

            if (slen == (size_t) -1) {
                while (*p && buf < last) {
                    *buf++ = *p++;
                }

            } else {
                len = ngx_min(((size_t) (last - buf)), slen);
                buf = ngx_cpymem(buf, p, len);
            }

            fmt++;

            continue;

        case 'O':
            i64 = (int64_t) va_arg(args, off_t);
            sign = 1;
```

```
            break;

        case 'P':
            i64 = (int64_t) va_arg(args, ngx_pid_t);
            sign = 1;
            break;

        case 'T':
            i64 = (int64_t) va_arg(args, time_t);
            sign = 1;
            break;

        case 'M':
            ms = (ngx_msec_t) va_arg(args, ngx_msec_t);
            if ((ngx_msec_int_t) ms == -1) {
                sign = 1;
                i64 = -1;
            } else {
                sign = 0;
                ui64 = (uint64_t) ms;
            }
            break;

        case 'z':
            if (sign) {
                i64 = (int64_t) va_arg(args, ssize_t);
            } else {
                ui64 = (uint64_t) va_arg(args, size_t);
            }
            break;

        case 'i':
            if (sign) {
                i64 = (int64_t) va_arg(args, ngx_int_t);
            } else {
                ui64 = (uint64_t) va_arg(args, ngx_uint_t);
            }

            if (max_width) {
                width = NGX_INT_T_LEN;
            }

            break;

        case 'd':
            if (sign) {
                i64 = (int64_t) va_arg(args, int);
            } else {
                ui64 = (uint64_t) va_arg(args, u_int);
            }
            break;

        case 'l':
            if (sign) {
                i64 = (int64_t) va_arg(args, long);
            } else {
                ui64 = (uint64_t) va_arg(args, u_long);
            }
            break;

        case 'D':
            if (sign) {
                i64 = (int64_t) va_arg(args, int32_t);
            } else {
                ui64 = (uint64_t) va_arg(args, uint32_t);
            }
            break;

        case 'L':
            if (sign) {
                i64 = va_arg(args, int64_t);
            } else {
                ui64 = va_arg(args, uint64_t);
            }
            break;

```

```
            case 'A':
                if (sign) {
                    i64 = (int64_t) va_arg(args, ngx_atomic_int_t);
                } else {
                    ui64 = (uint64_t) va_arg(args, ngx_atomic_uint_t);
                }

                if (max_width) {
                    width = NGX_ATOMIC_T_LEN;
                }

                break;

            case 'f':
                f = va_arg(args, double);

                if (f < 0) {
                    *buf++ = '-';
                    f = -f;
                }

                ui64 = (int64_t) f;
                frac = 0;

                if (frac_width) {

                    scale = 1;
                    for (n = frac_width; n; n--) {
                        scale *= 10;
                    }

                    frac = (uint64_t) ((f - (double) ui64) * scale + 0.5);

                    if (frac == scale) {
                        ui64++;
                        frac = 0;
                    }
                }

                buf = ngx_sprintf_num(buf, last, ui64, zero, 0, width);

                if (frac_width) {
                    if (buf < last) {
                        *buf++ = '.';
                    }

                    buf = ngx_sprintf_num(buf, last, frac, '0', 0, frac_width);
                }

                fmt++;

                continue;

#if !(NGX_WIN32)
            case 'r':
                i64 = (int64_t) va_arg(args, rlim_t);
                sign = 1;
                break;
#endif

            case 'p':
                ui64 = (uintptr_t) va_arg(args, void *);
                hex = 2;
                sign = 0;
                zero = '0';
                width = NGX_PTR_SIZE * 2;
                break;

            case 'c':
                d = va_arg(args, int);
                *buf++ = (u_char) (d & 0xff);
                fmt++;

                continue;

            case 'Z':
```

```
                *buf++ = '\0';
                fmt++;

                continue;

            case 'N':
#if (NGX_WIN32)
                *buf++ = CR;
                if (buf < last) {
                    *buf++ = LF;
                }
#else
                *buf++ = LF;
#endif
                fmt++;

                continue;

            case '%':
                *buf++ = '%';
                fmt++;

                continue;

            default:
                *buf++ = *fmt++;

                continue;
            }

            if (sign) {
                if (i64 < 0) {
                    *buf++ = '-';
                    ui64 = (uint64_t) -i64;

                } else {
                    ui64 = (uint64_t) i64;
                }
            }

            buf = ngx_sprintf_num(buf, last, ui64, zero, hex, width);

            fmt++;

        } else {
            *buf++ = *fmt++;
        }
    }

    return buf;
}


static u_char *
ngx_sprintf_num(u_char *buf, u_char *last, uint64_t ui64, u_char zero,
    ngx_uint_t hexadecimal, ngx_uint_t width)
{
    u_char         *p, temp[NGX_INT64_LEN + 1];
                    /*
                     * we need temp[NGX_INT64_LEN] only,
                     * but icc issues the warning
                     */
    size_t          len;
    uint32_t        ui32;
    static u_char   hex[] = "0123456789abcdef";
    static u_char   HEX[] = "0123456789ABCDEF";

    p = temp + NGX_INT64_LEN;

    if (hexadecimal == 0) {

        if (ui64 <= (uint64_t) NGX_MAX_UINT32_VALUE) {

            /*
             * To divide 64-bit numbers and to find remainders
             * on the x86 platform gcc and icc call the libc functions
```

```
             * [u]divdi3() and [u]moddi3(), they call another function
             * in its turn.  On FreeBSD it is the qdivrem() function,
             * its source code is about 170 lines of the code.
             * The glibc counterpart is about 150 lines of the code.
             *
             * For 32-bit numbers and some divisors gcc and icc use
             * a inlined multiplication and shifts.  For example,
             * unsigned "i32 / 10" is compiled to
             *
             *     (i32 * 0xCCCCCCCD) >> 35
             */

            ui32 = (uint32_t) ui64;

            do {
                *--p = (u_char) (ui32 % 10 + '0');
            } while (ui32 /= 10);

        } else {
            do {
                *--p = (u_char) (ui64 % 10 + '0');
            } while (ui64 /= 10);
        }

    } else if (hexadecimal == 1) {

        do {

            /* the "(uint32_t)" cast disables the BCC's warning */
            *--p = hex[(uint32_t) (ui64 & 0xf)];

        } while (ui64 >>= 4);

    } else { /* hexadecimal == 2 */

        do {

            /* the "(uint32_t)" cast disables the BCC's warning */
            *--p = HEX[(uint32_t) (ui64 & 0xf)];

        } while (ui64 >>= 4);
    }

    /* zero or space padding */

    len = (temp + NGX_INT64_LEN) - p;

    while (len++ < width && buf < last) {
        *buf++ = zero;
    }

    /* number safe copy */

    len = (temp + NGX_INT64_LEN) - p;

    if (buf + len > last) {
        len = last - buf;
    }

    return ngx_cpymem(buf, p, len);
}


/*
* We use ngx_strcasecmp()/ngx_strncasecmp() for 7-bit ASCII strings only,
* and implement our own ngx_strcasecmp()/ngx_strncasecmp()
* to avoid libc locale overhead.  Besides, we use the ngx_uint_t's
* instead of the u_char's, because they are slightly faster.
*/

ngx_int_t
ngx_strcasecmp(u_char *s1, u_char *s2)
{
    ngx_uint_t  c1, c2;

    for ( ;; ) {
```

```
        c1 = (ngx_uint_t) *s1++;
        c2 = (ngx_uint_t) *s2++;

        c1 = (c1 >= 'A' && c1 <= 'Z') ? (c1 | 0x20) : c1;
        c2 = (c2 >= 'A' && c2 <= 'Z') ? (c2 | 0x20) : c2;

        if (c1 == c2) {

            if (c1) {
                continue;
            }

            return 0;
        }

        return c1 - c2;
    }
}


ngx_int_t
ngx_strncasecmp(u_char *s1, u_char *s2, size_t n)
{
    ngx_uint_t  c1, c2;

    while (n) {
        c1 = (ngx_uint_t) *s1++;
        c2 = (ngx_uint_t) *s2++;

        c1 = (c1 >= 'A' && c1 <= 'Z') ? (c1 | 0x20) : c1;
        c2 = (c2 >= 'A' && c2 <= 'Z') ? (c2 | 0x20) : c2;

        if (c1 == c2) {

            if (c1) {
                n--;
                continue;
            }

            return 0;
        }

        return c1 - c2;
    }

    return 0;
}


u_char *
ngx_strnstr(u_char *s1, char *s2, size_t len)
{
    u_char  c1, c2;
    size_t  n;

    c2 = *(u_char *) s2++;

    n = ngx_strlen(s2);

    do {
        do {
            if (len-- == 0) {
                return NULL;
            }

            c1 = *s1++;

            if (c1 == 0) {
                return NULL;
            }

        } while (c1 != c2);

        if (n > len) {
            return NULL;
        }
```

```
652
653     } while (ngx_strncmp(s1, (u_char *) s2, n) != 0);
654
655     return --s1;
656 }
657
658
659 /*
660 * ngx_strstrn() and ngx_strcasestrn() are intended to search for static
661 * substring with known length in null-terminated string. The argument n
662 * must be length of the second substring - 1.
663 */
664
665 u_char *
666 ngx_strstrn(u_char *s1, char *s2, size_t n)
667 {
668     u_char  c1, c2;
669
670     c2 = *(u_char *) s2++;
671
672     do {
673         do {
674             c1 = *s1++;
675
676             if (c1 == 0) {
677                 return NULL;
678             }
679
680         } while (c1 != c2);
681
682     } while (ngx_strncmp(s1, (u_char *) s2, n) != 0);
683
684     return --s1;
685 }
686
687
688 u_char *
689 ngx_strcasestrn(u_char *s1, char *s2, size_t n)
690 {
691     ngx_uint_t  c1, c2;
692
693     c2 = (ngx_uint_t) *s2++;
694     c2 = (c2 >= 'A' && c2 <= 'Z') ? (c2 | 0x20) : c2;
695
696     do {
697         do {
698             c1 = (ngx_uint_t) *s1++;
699
700             if (c1 == 0) {
701                 return NULL;
702             }
703
704             c1 = (c1 >= 'A' && c1 <= 'Z') ? (c1 | 0x20) : c1;
705
706         } while (c1 != c2);
707
708     } while (ngx_strncasecmp(s1, (u_char *) s2, n) != 0);
709
710     return --s1;
711 }
712
713
714 /*
715 * ngx_strlcasestrn() is intended to search for static substring
716 * with known length in string until the argument last. The argument n
717 * must be length of the second substring - 1.
718 */
719
720 u_char *
721 ngx_strlcasestrn(u_char *s1, u_char *last, u_char *s2, size_t n)
722 {
723     ngx_uint_t  c1, c2;
724
725     c2 = (ngx_uint_t) *s2++;
726     c2 = (c2 >= 'A' && c2 <= 'Z') ? (c2 | 0x20) : c2;
727     last -= n;
```

```
728
729     do {
730         do {
731             if (s1 >= last) {
732                 return NULL;
733             }
734
735             c1 = (ngx_uint_t) *s1++;
736
737             c1 = (c1 >= 'A' && c1 <= 'Z') ? (c1 | 0x20) : c1;
738
739         } while (c1 != c2);
740
741     } while (ngx_strncasecmp(s1, s2, n) != 0);
742
743     return --s1;
744 }
745
746
747 ngx_int_t
748 ngx_rstrncmp(u_char *s1, u_char *s2, size_t n)
749 {
750     if (n == 0) {
751         return 0;
752     }
753
754     n--;
755
756     for ( ;; ) {
757         if (s1[n] != s2[n]) {
758             return s1[n] - s2[n];
759         }
760
761         if (n == 0) {
762             return 0;
763         }
764
765         n--;
766     }
767 }
768
769
770 ngx_int_t
771 ngx_rstrncasecmp(u_char *s1, u_char *s2, size_t n)
772 {
773     u_char  c1, c2;
774
775     if (n == 0) {
776         return 0;
777     }
778
779     n--;
780
781     for ( ;; ) {
782         c1 = s1[n];
783         if (c1 >= 'a' && c1 <= 'z') {
784             c1 -= 'a' - 'A';
785         }
786
787         c2 = s2[n];
788         if (c2 >= 'a' && c2 <= 'z') {
789             c2 -= 'a' - 'A';
790         }
791
792         if (c1 != c2) {
793             return c1 - c2;
794         }
795
796         if (n == 0) {
797             return 0;
798         }
799
800         n--;
801     }
802 }
803
```

```
804
805 ngx_int_t
806 ngx_memn2cmp(u_char *s1, u_char *s2, size_t n1, size_t n2)
807 {
808     size_t     n;
809     ngx_int_t  m, z;
810
811     if (n1 <= n2) {
812         n = n1;
813         z = -1;
814
815     } else {
816         n = n2;
817         z = 1;
818     }
819
820     m = ngx_memcmp(s1, s2, n);
821
822     if (m || n1 == n2) {
823         return m;
824     }
825
826     return z;
827 }
828
829
830 ngx_int_t
831 ngx_dns_strcmp(u_char *s1, u_char *s2)
832 {
833     ngx_uint_t  c1, c2;
834
835     for ( ;; ) {
836         c1 = (ngx_uint_t) *s1++;
837         c2 = (ngx_uint_t) *s2++;
838
839         c1 = (c1 >= 'A' && c1 <= 'Z') ? (c1 | 0x20) : c1;
840         c2 = (c2 >= 'A' && c2 <= 'Z') ? (c2 | 0x20) : c2;
841
842         if (c1 == c2) {
843
844             if (c1) {
845                 continue;
846             }
847
848             return 0;
849         }
850
851         /* in ASCII '.' > '-', but we need '.' to be the lowest character */
852
853         c1 = (c1 == '.') ? ' ' : c1;
854         c2 = (c2 == '.') ? ' ' : c2;
855
856         return c1 - c2;
857     }
858 }
859
860
861 ngx_int_t
862 ngx_filename_cmp(u_char *s1, u_char *s2, size_t n)
863 {
864     ngx_uint_t  c1, c2;
865
866     while (n) {
867         c1 = (ngx_uint_t) *s1++;
868         c2 = (ngx_uint_t) *s2++;
869
870 #if (NGX_HAVE_CASELESS_FILESYSTEM)
871         c1 = tolower(c1);
872         c2 = tolower(c2);
873 #endif
874
875         if (c1 == c2) {
876
877             if (c1) {
878                 n--;
879                 continue;
```

```
            }

            return 0;
        }

        /* we need '/' to be the lowest character */

        if (c1 == 0 || c2 == 0) {
            return c1 - c2;
        }

        c1 = (c1 == '/') ? 0 : c1;
        c2 = (c2 == '/') ? 0 : c2;

        return c1 - c2;
    }

    return 0;
}


ngx_int_t
ngx_atoi(u_char *line, size_t n)
{
    ngx_int_t  value;

    if (n == 0) {
        return NGX_ERROR;
    }

    for (value = 0; n--; line++) {
        if (*line < '0' || *line > '9') {
            return NGX_ERROR;
        }

        value = value * 10 + (*line - '0');
    }

    if (value < 0) {
        return NGX_ERROR;

    } else {
        return value;
    }
}


/* parse a fixed point number, e.g., ngx_atofp("10.5", 4, 2) returns 1050 */

ngx_int_t
ngx_atofp(u_char *line, size_t n, size_t point)
{
    ngx_int_t   value;
    ngx_uint_t  dot;

    if (n == 0) {
        return NGX_ERROR;
    }

    dot = 0;

    for (value = 0; n--; line++) {

        if (point == 0) {
            return NGX_ERROR;
        }

        if (*line == '.') {
            if (dot) {
                return NGX_ERROR;
            }

            dot = 1;
            continue;
        }

```

```
        if (*line < '0' || *line > '9') {
            return NGX_ERROR;
        }

        value = value * 10 + (*line - '0');
        point -= dot;
    }

    while (point--) {
        value = value * 10;
    }

    if (value < 0) {
        return NGX_ERROR;

    } else {
        return value;
    }
}


ssize_t
ngx_atosz(u_char *line, size_t n)
{
    ssize_t  value;

    if (n == 0) {
        return NGX_ERROR;
    }

    for (value = 0; n--; line++) {
        if (*line < '0' || *line > '9') {
            return NGX_ERROR;
        }

        value = value * 10 + (*line - '0');
    }

    if (value < 0) {
        return NGX_ERROR;

    } else {
        return value;
    }
}


off_t
ngx_atoof(u_char *line, size_t n)
{
    off_t  value;

    if (n == 0) {
        return NGX_ERROR;
    }

    for (value = 0; n--; line++) {
        if (*line < '0' || *line > '9') {
            return NGX_ERROR;
        }

        value = value * 10 + (*line - '0');
    }

    if (value < 0) {
        return NGX_ERROR;

    } else {
        return value;
    }
}


time_t
ngx_atotm(u_char *line, size_t n)
{
```

```
    time_t  value;

    if (n == 0) {
        return NGX_ERROR;
    }

    for (value = 0; n--; line++) {
        if (*line < '0' || *line > '9') {
            return NGX_ERROR;
        }

        value = value * 10 + (*line - '0');
    }

    if (value < 0) {
        return NGX_ERROR;

    } else {
        return value;
    }
}


ngx_int_t
ngx_hextoi(u_char *line, size_t n)
{
    u_char     c, ch;
    ngx_int_t  value;

    if (n == 0) {
        return NGX_ERROR;
    }

    for (value = 0; n--; line++) {
        ch = *line;

        if (ch >= '0' && ch <= '9') {
            value = value * 16 + (ch - '0');
            continue;
        }

        c = (u_char) (ch | 0x20);

        if (c >= 'a' && c <= 'f') {
            value = value * 16 + (c - 'a' + 10);
            continue;
        }

        return NGX_ERROR;
    }

    if (value < 0) {
        return NGX_ERROR;

    } else {
        return value;
    }
}


u_char *
ngx_hex_dump(u_char *dst, u_char *src, size_t len)
{
    static u_char  hex[] = "0123456789abcdef";

    while (len--) {
        *dst++ = hex[*src >> 4];
        *dst++ = hex[*src++ & 0xf];
    }

    return dst;
}


void
ngx_encode_base64(ngx_str_t *dst, ngx_str_t *src)
```

```
{
    static u_char   basis64[] =
            "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

    ngx_encode_base64_internal(dst, src, basis64, 1);
}


void
ngx_encode_base64url(ngx_str_t *dst, ngx_str_t *src)
{
    static u_char   basis64[] =
            "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_";

    ngx_encode_base64_internal(dst, src, basis64, 0);
}


static void
ngx_encode_base64_internal(ngx_str_t *dst, ngx_str_t *src, const u_char *basis,
    ngx_uint_t padding)
{
    u_char         *d, *s;
    size_t          len;

    len = src->len;
    s = src->data;
    d = dst->data;

    while (len > 2) {
        *d++ = basis[(s[0] >> 2) & 0x3f];
        *d++ = basis[((s[0] & 3) << 4) | (s[1] >> 4)];
        *d++ = basis[((s[1] & 0x0f) << 2) | (s[2] >> 6)];
        *d++ = basis[s[2] & 0x3f];

        s += 3;
        len -= 3;
    }

    if (len) {
        *d++ = basis[(s[0] >> 2) & 0x3f];

        if (len == 1) {
            *d++ = basis[(s[0] & 3) << 4];
            if (padding) {
                *d++ = '=';
            }

        } else {
            *d++ = basis[((s[0] & 3) << 4) | (s[1] >> 4)];
            *d++ = basis[(s[1] & 0x0f) << 2];
        }

        if (padding) {
            *d++ = '=';
        }
    }

    dst->len = d - dst->data;
}


ngx_int_t
ngx_decode_base64(ngx_str_t *dst, ngx_str_t *src)
{
    static u_char   basis64[] = {
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 62, 77, 77, 77, 63,
        52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 77, 77, 77, 77, 77, 77,
        77,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
        15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 77, 77, 77, 77, 77,
        77, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
        41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 77, 77, 77, 77, 77,

        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
```

```
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77
    };

    return ngx_decode_base64_internal(dst, src, basis64);
}


ngx_int_t
ngx_decode_base64url(ngx_str_t *dst, ngx_str_t *src)
{
    static u_char   basis64[] = {
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 62, 77, 77,
        52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 77, 77, 77, 77, 77, 77,
        77,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
        15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 77, 77, 77, 77, 63,
        77, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
        41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 77, 77, 77, 77, 77,

        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
        77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77
    };

    return ngx_decode_base64_internal(dst, src, basis64);
}


static ngx_int_t
ngx_decode_base64_internal(ngx_str_t *dst, ngx_str_t *src, const u_char *basis)
{
    size_t          len;
    u_char         *d, *s;

    for (len = 0; len < src->len; len++) {
        if (src->data[len] == '=') {
            break;
        }

        if (basis[src->data[len]] == 77) {
            return NGX_ERROR;
        }
    }

    if (len % 4 == 1) {
        return NGX_ERROR;
    }

    s = src->data;
    d = dst->data;

    while (len > 3) {
        *d++ = (u_char) (basis[s[0]] << 2 | basis[s[1]] >> 4);
        *d++ = (u_char) (basis[s[1]] << 4 | basis[s[2]] >> 2);
        *d++ = (u_char) (basis[s[2]] << 6 | basis[s[3]]);

        s += 4;
        len -= 4;
    }

    if (len > 1) {
        *d++ = (u_char) (basis[s[0]] << 2 | basis[s[1]] >> 4);
    }

```

```
    if (len > 2) {
        *d++ = (u_char) (basis[s[1]] << 4 | basis[s[2]] >> 2);
    }

    dst->len = d - dst->data;

    return NGX_OK;
}


/*
* ngx_utf8_decode() decodes two and more bytes UTF sequences only
* the return values:
*    0x80 - 0x10ffff         valid character
*    0x110000 - 0xfffffffd   invalid sequence
*    0xfffffffe              incomplete sequence
*    0xffffffff              error
*/

uint32_t
ngx_utf8_decode(u_char **p, size_t n)
{
    size_t    len;
    uint32_t  u, i, valid;

    u = **p;

    if (u >= 0xf0) {

        u &= 0x07;
        valid = 0xffff;
        len = 3;

    } else if (u >= 0xe0) {

        u &= 0x0f;
        valid = 0x7ff;
        len = 2;

    } else if (u >= 0xc2) {

        u &= 0x1f;
        valid = 0x7f;
        len = 1;

    } else {
        (*p)++;
        return 0xffffffff;
    }

    if (n - 1 < len) {
        return 0xfffffffe;
    }

    (*p)++;

    while (len) {
        i = *(*p)++;

        if (i < 0x80) {
            return 0xffffffff;
        }

        u = (u << 6) | (i & 0x3f);

        len--;
    }

    if (u > valid) {
        return u;
    }

    return 0xffffffff;
}


```

```c
size_t
ngx_utf8_length(u_char *p, size_t n)
{
    u_char  c, *last;
    size_t  len;

    last = p + n;

    for (len = 0; p < last; len++) {

        c = *p;

        if (c < 0x80) {
            p++;
            continue;
        }

        if (ngx_utf8_decode(&p, n) > 0x10ffff) {
            /* invalid UTF-8 */
            return n;
        }
    }

    return len;
}


u_char *
ngx_utf8_cpystrn(u_char *dst, u_char *src, size_t n, size_t len)
{
    u_char  c, *next;

    if (n == 0) {
        return dst;
    }

    while (--n) {

        c = *src;
        *dst = c;

        if (c < 0x80) {

            if (c != '\0') {
                dst++;
                src++;
                len--;

                continue;
            }

            return dst;
        }

        next = src;

        if (ngx_utf8_decode(&next, len) > 0x10ffff) {
            /* invalid UTF-8 */
            break;
        }

        while (src < next) {
            *dst++ = *src++;
            len--;
        }
    }

    *dst = '\0';

    return dst;
}


uintptr_t
ngx_escape_uri(u_char *dst, u_char *src, size_t size, ngx_uint_t type)
{
```

```
1412      ngx_uint_t      n;
1413      uint32_t       *escape;
1414      static u_char   hex[] = "0123456789ABCDEF";
1415
1416                      /* " ", "#", "%", "?", %00-%1F, %7F-%FF */
1417
1418      static uint32_t   uri[] = {
1419          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1420
1421                      /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
1422          0x80000029, /* 1000 0000 0000 0000  0000 0000 0010 1001 */
1423
1424                      /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
1425          0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
1426
1427                      /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
1428          0x80000000, /* 1000 0000 0000 0000  0000 0000 0000 0000 */
1429
1430          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1431          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1432          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1433          0xffffffff  /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1434      };
1435
1436                      /* " ", "#", "%", "&", "+", "?", %00-%1F, %7F-%FF */
1437
1438      static uint32_t   args[] = {
1439          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1440
1441                      /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
1442          0x88000869, /* 1000 1000 0000 0000  0000 1000 0110 1001 */
1443
1444                      /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
1445          0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
1446
1447                      /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
1448          0x80000000, /* 1000 0000 0000 0000  0000 0000 0000 0000 */
1449
1450          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1451          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1452          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1453          0xffffffff  /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1454      };
1455
1456                      /* not ALPHA, DIGIT, "-", ".", "_", "~" */
1457
1458      static uint32_t   uri_component[] = {
1459          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1460
1461                      /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
1462          0xfc009fff, /* 1111 1100 0000 0000  1001 1111 1111 1111 */
1463
1464                      /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
1465          0x78000001, /* 0111 1000 0000 0000  0000 0000 0000 0001 */
1466
1467                      /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
1468          0xb8000001, /* 1011 1000 0000 0000  0000 0000 0000 0001 */
1469
1470          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1471          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1472          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1473          0xffffffff  /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1474      };
1475
1476                      /* " ", "#", """, "%", "'", %00-%1F, %7F-%FF */
1477
1478      static uint32_t   html[] = {
1479          0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
1480
1481                      /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
1482          0x000000ad, /* 0000 0000 0000 0000  0000 0000 1010 1101 */
1483
1484                      /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
1485          0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
1486
1487                      /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
```

```
        0x80000000, /* 1000 0000 0000 0000  0000 0000 0000 0000 */

        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff  /* 1111 1111 1111 1111  1111 1111 1111 1111 */
    };

                    /* " ", """, "%", "'", %00-%1F, %7F-%FF */

    static uint32_t   refresh[] = {
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */

                    /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
        0x00000085, /* 0000 0000 0000 0000  0000 0000 1000 0101 */

                    /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */

                    /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
        0x80000000, /* 1000 0000 0000 0000  0000 0000 0000 0000 */

        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff  /* 1111 1111 1111 1111  1111 1111 1111 1111 */
    };

                    /* " ", "%", %00-%1F */

    static uint32_t   memcached[] = {
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */

                    /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
        0x00000021, /* 0000 0000 0000 0000  0000 0000 0010 0001 */

                    /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */

                    /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */

        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
        0x00000000, /* 0000 0000 0000 0000  0000 0000 0000 0000 */
    };

                    /* mail_auth is the same as memcached */

    static uint32_t  *map[] =
        { uri, args, uri_component, html, refresh, memcached, memcached };


    escape = map[type];

    if (dst == NULL) {

        /* find the number of the characters to be escaped */

        n = 0;

        while (size) {
            if (escape[*src >> 5] & (1 << (*src & 0x1f))) {
                n++;
            }
            src++;
            size--;
        }

        return (uintptr_t) n;
    }

    while (size) {
        if (escape[*src >> 5] & (1 << (*src & 0x1f))) {
            *dst++ = '%';
```

```
            *dst++ = hex[*src >> 4];
            *dst++ = hex[*src & 0xf];
            src++;

        } else {
            *dst++ = *src++;
        }
        size--;
    }

    return (uintptr_t) dst;
}


void
ngx_unescape_uri(u_char **dst, u_char **src, size_t size, ngx_uint_t type)
{
    u_char  *d, *s, ch, c, decoded;
    enum {
        sw_usual = 0,
        sw_quoted,
        sw_quoted_second
    } state;

    d = *dst;
    s = *src;

    state = 0;
    decoded = 0;

    while (size--) {

        ch = *s++;

        switch (state) {
        case sw_usual:
            if (ch == '?'
                && (type & (NGX_UNESCAPE_URI|NGX_UNESCAPE_REDIRECT)))
            {
                *d++ = ch;
                goto done;
            }

            if (ch == '%') {
                state = sw_quoted;
                break;
            }

            *d++ = ch;
            break;

        case sw_quoted:

            if (ch >= '0' && ch <= '9') {
                decoded = (u_char) (ch - '0');
                state = sw_quoted_second;
                break;
            }

            c = (u_char) (ch | 0x20);
            if (c >= 'a' && c <= 'f') {
                decoded = (u_char) (c - 'a' + 10);
                state = sw_quoted_second;
                break;
            }

            /* the invalid quoted character */

            state = sw_usual;

            *d++ = ch;

            break;

        case sw_quoted_second:

```

```
            state = sw_usual;

            if (ch >= '0' && ch <= '9') {
                ch = (u_char) ((decoded << 4) + ch - '0');

                if (type & NGX_UNESCAPE_REDIRECT) {
                    if (ch > '%' && ch < 0x7f) {
                        *d++ = ch;
                        break;
                    }

                    *d++ = '%'; *d++ = *(s - 2); *d++ = *(s - 1);

                    break;
                }

                *d++ = ch;

                break;
            }

            c = (u_char) (ch | 0x20);
            if (c >= 'a' && c <= 'f') {
                ch = (u_char) ((decoded << 4) + c - 'a' + 10);

                if (type & NGX_UNESCAPE_URI) {
                    if (ch == '?') {
                        *d++ = ch;
                        goto done;
                    }

                    *d++ = ch;
                    break;
                }

                if (type & NGX_UNESCAPE_REDIRECT) {
                    if (ch == '?') {
                        *d++ = ch;
                        goto done;
                    }

                    if (ch > '%' && ch < 0x7f) {
                        *d++ = ch;
                        break;
                    }

                    *d++ = '%'; *d++ = *(s - 2); *d++ = *(s - 1);
                    break;
                }

                *d++ = ch;

                break;
            }

            /* the invalid quoted character */

            break;
        }
    }

done:

    *dst = d;
    *src = s;
}


uintptr_t
ngx_escape_html(u_char *dst, u_char *src, size_t size)
{
    u_char      ch;
    ngx_uint_t  len;

    if (dst == NULL) {

```

```
        len = 0;

        while (size) {
            switch (*src++) {

            case '<':
                len += sizeof("&lt;") - 2;
                break;

            case '>':
                len += sizeof("&gt;") - 2;
                break;

            case '&':
                len += sizeof("&amp;") - 2;
                break;

            case '"':
                len += sizeof("&quot;") - 2;
                break;

            default:
                break;
            }
            size--;
        }

        return (uintptr_t) len;
    }

    while (size) {
        ch = *src++;

        switch (ch) {

        case '<':
            *dst++ = '&'; *dst++ = 'l'; *dst++ = 't'; *dst++ = ';';
            break;

        case '>':
            *dst++ = '&'; *dst++ = 'g'; *dst++ = 't'; *dst++ = ';';
            break;

        case '&':
            *dst++ = '&'; *dst++ = 'a'; *dst++ = 'm'; *dst++ = 'p';
            *dst++ = ';';
            break;

        case '"':
            *dst++ = '&'; *dst++ = 'q'; *dst++ = 'u'; *dst++ = 'o';
            *dst++ = 't'; *dst++ = ';';
            break;

        default:
            *dst++ = ch;
            break;
        }
        size--;
    }

    return (uintptr_t) dst;
}


uintptr_t
ngx_escape_json(u_char *dst, u_char *src, size_t size)
{
    u_char      ch;
    ngx_uint_t  len;

    if (dst == NULL) {
        len = 0;

        while (size) {
            ch = *src++;

```

```
            if (ch == '\\' || ch == '"') {
                len++;

            } else if (ch <= 0x1f) {
                len += sizeof("\\u001F") - 2;
            }

            size--;
        }

        return (uintptr_t) len;
    }

    while (size) {
        ch = *src++;

        if (ch > 0x1f) {

            if (ch == '\\' || ch == '"') {
                *dst++ = '\\';
            }

            *dst++ = ch;

        } else {
            *dst++ = '\\'; *dst++ = 'u'; *dst++ = '0'; *dst++ = '0';
            *dst++ = '0' + (ch >> 4);

            ch &= 0xf;

            *dst++ = (ch < 10) ? ('0' + ch) : ('A' + ch - 10);
        }

        size--;
    }

    return (uintptr_t) dst;
}


void
ngx_str_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
    ngx_str_node_t      *n, *t;
    ngx_rbtree_node_t  **p;

    for ( ;; ) {

        n = (ngx_str_node_t *) node;
        t = (ngx_str_node_t *) temp;

        if (node->key != temp->key) {

            p = (node->key < temp->key) ? &temp->left : &temp->right;

        } else if (n->str.len != t->str.len) {

            p = (n->str.len < t->str.len) ? &temp->left : &temp->right;

        } else {
            p = (ngx_memcmp(n->str.data, t->str.data, n->str.len) < 0)
                 ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
```

```
    ngx_rbt_red(node);
}


ngx_str_node_t *
ngx_str_rbtree_lookup(ngx_rbtree_t *rbtree, ngx_str_t *val, uint32_t hash)
{
    ngx_int_t           rc;
    ngx_str_node_t     *n;
    ngx_rbtree_node_t  *node, *sentinel;

    node = rbtree->root;
    sentinel = rbtree->sentinel;

    while (node != sentinel) {

        n = (ngx_str_node_t *) node;

        if (hash != node->key) {
            node = (hash < node->key) ? node->left : node->right;
            continue;
        }

        if (val->len != n->str.len) {
            node = (val->len < n->str.len) ? node->left : node->right;
            continue;
        }

        rc = ngx_memcmp(val->data, n->str.data, val->len);

        if (rc < 0) {
            node = node->left;
            continue;
        }

        if (rc > 0) {
            node = node->right;
            continue;
        }

        return n;
    }

    return NULL;
}


/* ngx_sort() is implemented as insertion sort because we need stable sort */

void
ngx_sort(void *base, size_t n, size_t size,
    ngx_int_t (*cmp)(const void *, const void *))
{
    u_char  *p1, *p2, *p;

    p = ngx_alloc(size, ngx_cycle->log);
    if (p == NULL) {
        return;
    }

    for (p1 = (u_char *) base + size;
         p1 < (u_char *) base + n * size;
         p1 += size)
    {
        ngx_memcpy(p, p1, size);

        for (p2 = p1;
             p2 > (u_char *) base && cmp(p2 - size, p) > 0;
             p2 -= size)
        {
            ngx_memcpy(p2, p2 - size, size);
        }

        ngx_memcpy(p2, p, size);
    }

```

```
1944      ngx_free(p);
1945  }
1946
1947
1948  #if (NGX_MEMCPY_LIMIT)
1949
1950  void *
1951  ngx_memcpy(void *dst, const void *src, size_t n)
1952  {
1953      if (n > NGX_MEMCPY_LIMIT) {
1954          ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "memcpy %uz bytes", n);
1955          ngx_debug_point();
1956      }
1957
1958      return memcpy(dst, src, n);
1959  }
1960
1961  #endif
```

One Level Up Top Level

src/core/ngx_palloc.c - nginx-1.7.10

Functions defined

- ngx_create_pool
- ngx_destroy_pool
- ngx_palloc
- ngx_palloc_block
- ngx_palloc_large
- ngx_pcalloc
- ngx_pfree
- ngx_pmemalign
- ngx_pnalloc
- ngx_pool_cleanup_add
- ngx_pool_cleanup_file
- ngx_pool_delete_file
- ngx_pool_run_cleanup_file
- ngx_reset_pool

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  static void *ngx_palloc_block(ngx_pool_t *pool, size_t size);
13  static void *ngx_palloc_large(ngx_pool_t *pool, size_t size);
14
15
16  ngx_pool_t *
17  ngx_create_pool(size_t size, ngx_log_t *log)
18  {
19      ngx_pool_t  *p;
20
21      p = ngx_memalign(NGX_POOL_ALIGNMENT, size, log);
22      if (p == NULL) {
23          return NULL;
24      }
25
26      p->d.last = (u_char *) p + sizeof(ngx_pool_t);
27      p->d.end = (u_char *) p + size;
28      p->d.next = NULL;
29      p->d.failed = 0;
30
31      size = size - sizeof(ngx_pool_t);
32      p->max = (size < NGX_MAX_ALLOC_FROM_POOL) ? size : NGX_MAX_ALLOC_FROM_POOL;
33
```

```
    p->current = p;
    p->chain = NULL;
    p->large = NULL;
    p->cleanup = NULL;
    p->log = log;

    return p;
}


void
ngx_destroy_pool(ngx_pool_t *pool)
{
    ngx_pool_t          *p, *n;
    ngx_pool_large_t    *l;
    ngx_pool_cleanup_t  *c;

    for (c = pool->cleanup; c; c = c->next) {
        if (c->handler) {
            ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, pool->log, 0,
                           "run cleanup: %p", c);
            c->handler(c->data);
        }
    }

    for (l = pool->large; l; l = l->next) {

        ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, pool->log, 0, "free: %p", l->alloc);

        if (l->alloc) {
            ngx_free(l->alloc);
        }
    }

#if (NGX_DEBUG)

    /*
     * we could allocate the pool->log from this pool
     * so we cannot use this log while free()ing the pool
     */

    for (p = pool, n = pool->d.next; /* void */; p = n, n = n->d.next) {
        ngx_log_debug2(NGX_LOG_DEBUG_ALLOC, pool->log, 0,
                       "free: %p, unused: %uz", p, p->d.end - p->d.last);

        if (n == NULL) {
            break;
        }
    }

#endif

    for (p = pool, n = pool->d.next; /* void */; p = n, n = n->d.next) {
        ngx_free(p);

        if (n == NULL) {
            break;
        }
    }
}


void
ngx_reset_pool(ngx_pool_t *pool)
{
    ngx_pool_t        *p;
     ngx_pool_large_t  *l;

     for (l = pool->large; l; l = l->next) {
         if (l->alloc) {
             ngx_free(l->alloc);
         }
     }

     for (p = pool; p; p = p->d.next) {
         p->d.last = (u_char *) p + sizeof(ngx_pool_t);
```

```
        p->d.failed = 0;
    }

    pool->current = pool;
    pool->chain = NULL;
    pool->large = NULL;
}


void *
ngx_palloc(ngx_pool_t *pool, size_t size)
{
    u_char      *m;
    ngx_pool_t  *p;

    if (size <= pool->max) {

        p = pool->current;

        do {
            m = ngx_align_ptr(p->d.last, NGX_ALIGNMENT);

            if ((size_t) (p->d.end - m) >= size) {
                p->d.last = m + size;

                return m;
            }

            p = p->d.next;

        } while (p);

        return ngx_palloc_block(pool, size);
    }

    return ngx_palloc_large(pool, size);
}


void *
ngx_pnalloc(ngx_pool_t *pool, size_t size)
{
    u_char      *m;
    ngx_pool_t  *p;

    if (size <= pool->max) {

        p = pool->current;

        do {
            m = p->d.last;

            if ((size_t) (p->d.end - m) >= size) {
                p->d.last = m + size;

                return m;
            }

            p = p->d.next;

        } while (p);

        return ngx_palloc_block(pool, size);
    }

    return ngx_palloc_large(pool, size);
}


static void *
ngx_palloc_block(ngx_pool_t *pool, size_t size)
{
    u_char      *m;
    size_t       psize;
    ngx_pool_t  *p, *new;

```

```
    psize = (size_t) (pool->d.end - (u_char *) pool);

    m = ngx_memalign(NGX_POOL_ALIGNMENT, psize, pool->log);
    if (m == NULL) {
        return NULL;
    }

    new = (ngx_pool_t *) m;

    new->d.end = m + psize;
    new->d.next = NULL;
    new->d.failed = 0;

    m += sizeof(ngx_pool_data_t);
    m = ngx_align_ptr(m, NGX_ALIGNMENT);
    new->d.last = m + size;

    for (p = pool->current; p->d.next; p = p->d.next) {
        if (p->d.failed++ > 4) {
            pool->current = p->d.next;
        }
    }

    p->d.next = new;

    return m;
}


static void *
ngx_palloc_large(ngx_pool_t *pool, size_t size)
{
    void              *p;
    ngx_uint_t         n;
    ngx_pool_large_t  *large;

    p = ngx_alloc(size, pool->log);
    if (p == NULL) {
        return NULL;
    }

    n = 0;

    for (large = pool->large; large; large = large->next) {
        if (large->alloc == NULL) {
            large->alloc = p;
            return p;
        }

        if (n++ > 3) {
            break;
        }
    }

    large = ngx_palloc(pool, sizeof(ngx_pool_large_t));
    if (large == NULL) {
        ngx_free(p);
        return NULL;
    }

    large->alloc = p;
    large->next = pool->large;
    pool->large = large;

    return p;
}


void *
ngx_pmemalign(ngx_pool_t *pool, size_t size, size_t alignment)
{
    void              *p;
    ngx_pool_large_t  *large;

    p = ngx_memalign(alignment, size, pool->log);
    if (p == NULL) {
```

```
        return NULL;
    }

    large = ngx_palloc(pool, sizeof(ngx_pool_large_t));
    if (large == NULL) {
        ngx_free(p);
        return NULL;
    }

    large->alloc = p;
    large->next = pool->large;
    pool->large = large;

    return p;
}


ngx_int_t
ngx_pfree(ngx_pool_t *pool, void *p)
{
    ngx_pool_large_t  *l;

    for (l = pool->large; l; l = l->next) {
        if (p == l->alloc) {
            ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, pool->log, 0,
                           "free: %p", l->alloc);
            ngx_free(l->alloc);
            l->alloc = NULL;

            return NGX_OK;
        }
    }

    return NGX_DECLINED;
}


void *
ngx_pcalloc(ngx_pool_t *pool, size_t size)
{
    void *p;

    p = ngx_palloc(pool, size);
    if (p) {
        ngx_memzero(p, size);
    }

    return p;
}


ngx_pool_cleanup_t *
ngx_pool_cleanup_add(ngx_pool_t *p, size_t size)
{
    ngx_pool_cleanup_t  *c;

    c = ngx_palloc(p, sizeof(ngx_pool_cleanup_t));
    if (c == NULL) {
        return NULL;
    }

    if (size) {
        c->data = ngx_palloc(p, size);
        if (c->data == NULL) {
            return NULL;
        }

    } else {
        c->data = NULL;
    }

    c->handler = NULL;
    c->next = p->cleanup;

    p->cleanup = c;

```

```
    ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, p->log, 0, "add cleanup: %p", c);

    return c;
}


void
ngx_pool_run_cleanup_file(ngx_pool_t *p, ngx_fd_t fd)
{
    ngx_pool_cleanup_t       *c;
    ngx_pool_cleanup_file_t  *cf;

    for (c = p->cleanup; c; c = c->next) {
        if (c->handler == ngx_pool_cleanup_file) {

            cf = c->data;

            if (cf->fd == fd) {
                c->handler(cf);
                c->handler = NULL;
                return;
            }
        }
    }
}


void
ngx_pool_cleanup_file(void *data)
{
    ngx_pool_cleanup_file_t  *c = data;

    ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, c->log, 0, "file cleanup: fd:%d",
                   c->fd);

    if (ngx_close_file(c->fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", c->name);
    }
}


void
ngx_pool_delete_file(void *data)
{
    ngx_pool_cleanup_file_t  *c = data;

    ngx_err_t  err;

    ngx_log_debug2(NGX_LOG_DEBUG_ALLOC, c->log, 0, "file cleanup: fd:%d %s",
                   c->fd, c->name);

    if (ngx_delete_file(c->name) == NGX_FILE_ERROR) {
        err = ngx_errno;

        if (err != NGX_ENOENT) {
            ngx_log_error(NGX_LOG_CRIT, c->log, err,
                          ngx_delete_file_n " \"%s\" failed", c->name);
        }
    }

    if (ngx_close_file(c->fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", c->name);
    }
}


#if 0

static void *
ngx_get_cached_block(size_t size)
{
    void                     *p;
    ngx_cached_block_slot_t  *slot;

```

```
414     if (ngx_cycle->cache == NULL) {
415         return NULL;
416     }
417
418     slot = &ngx_cycle->cache[(size + ngx_pagesize - 1) / ngx_pagesize];
419
420     slot->tries++;
421
422     if (slot->number) {
423         p = slot->block;
424         slot->block = slot->block->next;
425         slot->number--;
426         return p;
427     }
428
429     return NULL;
430 }
431
432 #endif
```

One Level Up Top Level

src/os/unix/ngx_alloc.c - nginx-1.7.10

Global variables defined

- ngx_cacheline_size
- ngx_pagesize
- ngx_pagesize_shift

Functions defined

- ngx_alloc
- ngx_calloc
- ngx_memalign
- ngx_memalign

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 ngx_uint_t  ngx_pagesize;
13 ngx_uint_t  ngx_pagesize_shift;
14 ngx_uint_t  ngx_cacheline_size;
15
16
17 void *
18 ngx_alloc(size_t size, ngx_log_t *log)
19 {
20     void  *p;
21
22     p = malloc(size);
23     if (p == NULL) {
24         ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
25                       "malloc(%uz) failed", size);
26     }
27
28     ngx_log_debug2(NGX_LOG_DEBUG_ALLOC, log, 0, "malloc: %p:%uz", p, size);
29
30     return p;
31 }
32
33
34 void *
35 ngx_calloc(size_t size, ngx_log_t *log)
36 {
37     void  *p;
38
39     p = ngx_alloc(size, log);
40
41     if (p) {
42         ngx_memzero(p, size);
43     }
44
45     return p;
```

```
}


#if (NGX_HAVE_POSIX_MEMALIGN)

void *
ngx_memalign(size_t alignment, size_t size, ngx_log_t *log)
{
    void  *p;
    int    err;

    err = posix_memalign(&p, alignment, size);

    if (err) {
        ngx_log_error(NGX_LOG_EMERG, log, err,
                      "posix_memalign(%uz, %uz) failed", alignment, size);
        p = NULL;
    }

    ngx_log_debug3(NGX_LOG_DEBUG_ALLOC, log, 0,
                   "posix_memalign: %p:%uz @%uz", p, size, alignment);

    return p;
}

#elif (NGX_HAVE_MEMALIGN)

void *
ngx_memalign(size_t alignment, size_t size, ngx_log_t *log)
{
    void  *p;

    p = memalign(alignment, size);
    if (p == NULL) {
        ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
                      "memalign(%uz, %uz) failed", alignment, size);
    }

    ngx_log_debug3(NGX_LOG_DEBUG_ALLOC, log, 0,
                   "memalign: %p:%uz @%uz", p, size, alignment);

    return p;
}

#endif
```

One Level Up Top Level

src/os/unix/ngx_errno.h - nginx-1.7.10

Data types defined

- ngx_err_t

Macros defined

- NGX_EACCES
- NGX_EADDRINUSE
- NGX_EAGAIN
- NGX_EAGAIN
- NGX_EBADF
- NGX_EBUSY
- NGX_ECANCELED
- NGX_ECHILD
- NGX_ECONNABORTED
- NGX_ECONNREFUSED
- NGX_ECONNRESET
- NGX_EEXIST
- NGX_EHOSTDOWN
- NGX_EHOSTUNREACH
- NGX_EILSEQ
- NGX_EINPROGRESS
- NGX_EINTR
- NGX_EINVAL
- NGX_EISDIR
- NGX_ELOOP
- NGX_EMFILE
- NGX_EMLINK
- NGX_ENAMETOOLONG
- NGX_ENETDOWN
- NGX_ENETUNREACH
- NGX_ENFILE
- NGX_ENOENT

- NGX_ENOMEM
- NGX_ENOMOREFILES
- NGX_ENOPATH
- NGX_ENOPROTOOPT
- NGX_ENOSPC
- NGX_ENOSYS
- NGX_ENOTCONN
- NGX_ENOTDIR
- NGX_EOPNOTSUPP
- NGX_EPERM
- NGX_EPIPE
- NGX_ESRCH
- NGX_ETIMEDOUT
- NGX_EXDEV
- _NGX_ERRNO_H_INCLUDED_
- ngx_errno
- ngx_set_errno
- ngx_set_socket_errno
- ngx_socket_errno

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_ERRNO_H_INCLUDED_
9  #define _NGX_ERRNO_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef int               ngx_err_t;
17
18  #define NGX_EPERM         EPERM
19  #define NGX_ENOENT        ENOENT
20  #define NGX_ENOPATH       ENOENT
21  #define NGX_ESRCH         ESRCH
22  #define NGX_EINTR         EINTR
23  #define NGX_ECHILD        ECHILD
24  #define NGX_ENOMEM        ENOMEM
25  #define NGX_EACCES        EACCES
26  #define NGX_EBUSY         EBUSY
27  #define NGX_EEXIST        EEXIST
28  #define NGX_EXDEV         EXDEV
29  #define NGX_ENOTDIR       ENOTDIR
```

```
30  #define NGX_EISDIR        EISDIR
31  #define NGX_EINVAL        EINVAL
32  #define NGX_ENFILE        ENFILE
33  #define NGX_EMFILE        EMFILE
34  #define NGX_ENOSPC        ENOSPC
35  #define NGX_EPIPE         EPIPE
36  #define NGX_EINPROGRESS   EINPROGRESS
37  #define NGX_ENOPROTOOPT   ENOPROTOOPT
38  #define NGX_EOPNOTSUPP    EOPNOTSUPP
39  #define NGX_EADDRINUSE    EADDRINUSE
40  #define NGX_ECONNABORTED  ECONNABORTED
41  #define NGX_ECONNRESET    ECONNRESET
42  #define NGX_ENOTCONN      ENOTCONN
43  #define NGX_ETIMEDOUT     ETIMEDOUT
44  #define NGX_ECONNREFUSED  ECONNREFUSED
45  #define NGX_ENAMETOOLONG  ENAMETOOLONG
46  #define NGX_ENETDOWN      ENETDOWN
47  #define NGX_ENETUNREACH   ENETUNREACH
48  #define NGX_EHOSTDOWN     EHOSTDOWN
49  #define NGX_EHOSTUNREACH  EHOSTUNREACH
50  #define NGX_ENOSYS        ENOSYS
51  #define NGX_ECANCELED     ECANCELED
52  #define NGX_EILSEQ        EILSEQ
53  #define NGX_ENOMOREFILES  0
54  #define NGX_ELOOP         ELOOP
55  #define NGX_EBADF         EBADF
56
57  #if (NGX_HAVE_OPENAT)
58  #define NGX_EMLINK        EMLINK
59  #endif
60
61  #if (__hpux__)
62  #define NGX_EAGAIN        EWOULDBLOCK
63  #else
64  #define NGX_EAGAIN        EAGAIN
65  #endif
66
67
68  #define ngx_errno                  errno
69  #define ngx_socket_errno           errno
70  #define ngx_set_errno(err)         errno = err
71  #define ngx_set_socket_errno(err)  errno = err
72
73
74  u_char *ngx_strerror(ngx_err_t err, u_char *errstr, size_t size);
75  ngx_int_t ngx_strerror_init(void);
76
77
78  #endif /* _NGX_ERRNO_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_errno.c - nginx-1.7.10

Global variables defined

- ngx_sys_errlist
- ngx_unknown_error

Functions defined

- ngx_strerror
- ngx_strerror_init

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13   * The strerror() messages are copied because:
14   *
15   * 1) strerror() and strerror_r() functions are not Async-Signal-Safe,
16   *    therefore, they cannot be used in signal handlers;
17   *
18   * 2) a direct sys_errlist[] array may be used instead of these functions,
19   *    but Linux linker warns about its usage:
20   *
21   * warning: `sys_errlist' is deprecated; use `strerror' or `strerror_r' instead
22   * warning: `sys_nerr' is deprecated; use `strerror' or `strerror_r' instead
23   *
24   *    causing false bug reports.
25   */
26
27
28  static ngx_str_t  *ngx_sys_errlist;
29  static ngx_str_t   ngx_unknown_error = ngx_string("Unknown error");
30
31
32  u_char *
33  ngx_strerror(ngx_err_t err, u_char *errstr, size_t size)
34  {
35      ngx_str_t  *msg;
36
37      msg = ((ngx_uint_t) err < NGX_SYS_NERR) ? &ngx_sys_errlist[err]:
38                                                &ngx_unknown_error;
39      size = ngx_min(size, msg->len);
40
41      return ngx_cpymem(errstr, msg->data, size);
42  }
43
44
45  ngx_int_t
46  ngx_strerror_init(void)
47  {
48      char       *msg;
49      u_char     *p;
50      size_t      len;
51      ngx_err_t   err;
52
```

```
53      /*
54       * ngx_strerror() is not ready to work at this stage, therefore,
55       * malloc() is used and possible errors are logged using strerror().
56       */
57
58      len = NGX_SYS_NERR * sizeof(ngx_str_t);
59
60      ngx_sys_errlist = malloc(len);
61      if (ngx_sys_errlist == NULL) {
62          goto failed;
63      }
64
65      for (err = 0; err < NGX_SYS_NERR; err++) {
66          msg = strerror(err);
67          len = ngx_strlen(msg);
68
69          p = malloc(len);
70          if (p == NULL) {
71              goto failed;
72          }
73
74          ngx_memcpy(p, msg, len);
75          ngx_sys_errlist[err].len = len;
76          ngx_sys_errlist[err].data = p;
77      }
78
79      return NGX_OK;
80
81  failed:
82
83      err = errno;
84      ngx_log_stderr(0, "malloc(%uz) failed (%d: %s)", len, err, strerror(err));
85
86      return NGX_ERROR;
87  }
```

One Level Up Top Level

src/core/ngx_palloc.h - nginx-1.7.10

Data types defined

- ngx_pool_cleanup_file_t
- ngx_pool_cleanup_pt
- ngx_pool_cleanup_s
- ngx_pool_cleanup_t
- ngx_pool_data_t
- ngx_pool_large_s
- ngx_pool_large_t
- ngx_pool_s

Macros defined

- NGX_DEFAULT_POOL_SIZE
- NGX_MAX_ALLOC_FROM_POOL
- NGX_MIN_POOL_SIZE
- NGX_POOL_ALIGNMENT
- _NGX_PALLOC_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_PALLOC_H_INCLUDED_
9  #define _NGX_PALLOC_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  /*
17   * NGX_MAX_ALLOC_FROM_POOL should be (ngx_pagesize - 1), i.e. 4095 on x86.
18   * On Windows NT it decreases a number of locked pages in a kernel.
19   */
20  #define NGX_MAX_ALLOC_FROM_POOL  (ngx_pagesize - 1)
21
22  #define NGX_DEFAULT_POOL_SIZE    (16 * 1024)
23
24  #define NGX_POOL_ALIGNMENT       16
25  #define NGX_MIN_POOL_SIZE                                                     \
26      ngx_align((sizeof(ngx_pool_t) + 2 * sizeof(ngx_pool_large_t)),            \
27                NGX_POOL_ALIGNMENT)
28
29
30  typedef void (*ngx_pool_cleanup_pt)(void *data);
31
32  typedef struct ngx_pool_cleanup_s  ngx_pool_cleanup_t;
```

```
33
34  struct ngx_pool_cleanup_s {
35      ngx_pool_cleanup_pt   handler;
36      void                 *data;
37      ngx_pool_cleanup_t   *next;
38  };
39
40
41  typedef struct ngx_pool_large_s  ngx_pool_large_t;
42
43  struct ngx_pool_large_s {
44      ngx_pool_large_t     *next;
45      void                 *alloc;
46  };
47
48
49  typedef struct {
50      u_char               *last;
51      u_char               *end;
52      ngx_pool_t           *next;
53      ngx_uint_t            failed;
54  } ngx_pool_data_t;
55
56
57  struct ngx_pool_s {
58      ngx_pool_data_t       d;
59      size_t                max;
60      ngx_pool_t           *current;
61      ngx_chain_t          *chain;
62      ngx_pool_large_t     *large;
63      ngx_pool_cleanup_t   *cleanup;
64      ngx_log_t            *log;
65  };
66
67
68  typedef struct {
69      ngx_fd_t              fd;
70      u_char               *name;
71      ngx_log_t            *log;
72  } ngx_pool_cleanup_file_t;
73
74
75  void *ngx_alloc(size_t size, ngx_log_t *log);
76  void *ngx_calloc(size_t size, ngx_log_t *log);
77
78  ngx_pool_t *ngx_create_pool(size_t size, ngx_log_t *log);
79  void ngx_destroy_pool(ngx_pool_t *pool);
80  void ngx_reset_pool(ngx_pool_t *pool);
81
82  void *ngx_palloc(ngx_pool_t *pool, size_t size);
83  void *ngx_pnalloc(ngx_pool_t *pool, size_t size);
84  void *ngx_pcalloc(ngx_pool_t *pool, size_t size);
85  void *ngx_pmemalign(ngx_pool_t *pool, size_t size, size_t alignment);
86  ngx_int_t ngx_pfree(ngx_pool_t *pool, void *p);
87
88
89  ngx_pool_cleanup_t *ngx_pool_cleanup_add(ngx_pool_t *p, size_t size);
90  void ngx_pool_run_cleanup_file(ngx_pool_t *p, ngx_fd_t fd);
91  void ngx_pool_cleanup_file(void *data);
92  void ngx_pool_delete_file(void *data);
93
94
95  #endif /* _NGX_PALLOC_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_alloc.h - nginx-1.7.10

Macros defined

- _NGX_ALLOC_H_INCLUDED_
- ngx_free
- ngx_memalign

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_ALLOC_H_INCLUDED_
9  #define _NGX_ALLOC_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  void *ngx_alloc(size_t size, ngx_log_t *log);
17  void *ngx_calloc(size_t size, ngx_log_t *log);
18
19  #define ngx_free          free
20
21
22  /*
23  * Linux has memalign() or posix_memalign()
24  * Solaris has memalign()
25  * FreeBSD 7.0 has posix_memalign(), besides, early version's malloc()
26  * aligns allocations bigger than page size at the page boundary
27  */
28
29  #if (NGX_HAVE_POSIX_MEMALIGN || NGX_HAVE_MEMALIGN)
30
31  void *ngx_memalign(size_t alignment, size_t size, ngx_log_t *log);
32
33  #else
34
35  #define ngx_memalign(alignment, size, log)  ngx_alloc(size, log)
36
37  #endif
38
39
40  extern ngx_uint_t  ngx_pagesize;
41  extern ngx_uint_t  ngx_pagesize_shift;
42  extern ngx_uint_t  ngx_cacheline_size;
43
44
45  #endif /* _NGX_ALLOC_H_INCLUDED_ */
```

src/os/unix/ngx_solaris_config.h - nginx-1.7.10

Macros defined

- NGX_ALIGNMENT
- NGX_HAVE_INHERITED_NONBLOCK
- NGX_HAVE_OS_SPECIFIC_INIT
- NGX_HAVE_SO_SNDLOWAT
- NGX_LISTEN_BACKLOG
- _FILE_OFFSET_BITS
- _NGX_SOLARIS_CONFIG_H_INCLUDED_
- _REENTRANT
- ngx_debug_init

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_SOLARIS_CONFIG_H_INCLUDED_
9  #define _NGX_SOLARIS_CONFIG_H_INCLUDED_
10
11
12  #ifndef _REENTRANT
13  #define _REENTRANT
14  #endif
15
16  #define _FILE_OFFSET_BITS  64   /* must be before <sys/types.h> */
17
18  #include <sys/types.h>
19  #include <sys/time.h>
20  #include <unistd.h>
21  #include <stdarg.h>
22  #include <stddef.h>             /* offsetof() */
23  #include <stdio.h>
24  #include <stdlib.h>
25  #include <ctype.h>
26  #include <errno.h>
27  #include <string.h>
28  #include <signal.h>
29  #include <pwd.h>
30  #include <grp.h>
31  #include <dirent.h>
32  #include <glob.h>
33  #include <time.h>
34  #include <sys/statvfs.h>        /* statvfs() */
35
36  #include <sys/filio.h>          /* FIONBIO */
37  #include <sys/uio.h>
38  #include <sys/stat.h>
39  #include <fcntl.h>
40
41  #include <sys/wait.h>
42  #include <sys/mman.h>
43  #include <sys/resource.h>
44  #include <sched.h>
```

```
45
46  #include <sys/socket.h>
47  #include <netinet/in.h>
48  #include <netinet/tcp.h>        /* TCP_NODELAY */
49  #include <arpa/inet.h>
50  #include <netdb.h>
51  #include <sys/un.h>
52
53  #include <sys/systeminfo.h>
54  #include <limits.h>             /* IOV_MAX */
55  #include <inttypes.h>
56  #include <crypt.h>
57
58  #define NGX_ALIGNMENT  _MAX_ALIGNMENT
59
60  #include <ngx_auto_config.h>
61
62
63  #if (NGX_HAVE_POSIX_SEM)
64  #include <semaphore.h>
65  #endif
66
67
68  #if (NGX_HAVE_POLL)
69  #include <poll.h>
70  #endif
71
72
73  #if (NGX_HAVE_DEVPOLL)
74  #include <sys/ioctl.h>
75  #include <sys/devpoll.h>
76  #endif
77
78
79  #if (NGX_HAVE_EVENTPORT)
80  #include <port.h>
81  #endif
82
83
84  #if (NGX_HAVE_SENDFILE)
85  #include <sys/sendfile.h>
86  #endif
87
88
89  #define NGX_LISTEN_BACKLOG           511
90
91
92  #ifndef NGX_HAVE_INHERITED_NONBLOCK
93  #define NGX_HAVE_INHERITED_NONBLOCK  1
94  #endif
95
96
97  #ifndef NGX_HAVE_SO_SNDLOWAT
98  /* setsockopt(SO_SNDLOWAT) returns ENOPROTOOPT */
99  #define NGX_HAVE_SO_SNDLOWAT         0
100  #endif
101
102
103  #define NGX_HAVE_OS_SPECIFIC_INIT    1
104  #define ngx_debug_init()
105
106
107  extern char **environ;
108
109
110  #endif /* _NGX_SOLARIS_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_posix_config.h - nginx-1.7.10

Data types defined

- ngx_aiocb_t

Macros defined

- CMSG_DATA
- CMSG_DATA
- CMSG_LEN
- CMSG_LEN
- CMSG_SPACE
- CMSG_SPACE
- IOV_MAX
- NGX_BROKEN_SCM_RIGHTS
- NGX_LISTEN_BACKLOG
- _HPUX_ALT_XOPEN_SOCKET_API
- _NGX_POSIX_CONFIG_H_INCLUDED_
- _REENTRANT
- _XOPEN_SOURCE
- _XOPEN_SOURCE_EXTENDED
- ngx_debug_init
- timezonevar

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_POSIX_CONFIG_H_INCLUDED_
9  #define _NGX_POSIX_CONFIG_H_INCLUDED_
10
11
12  #if (NGX_HPUX)
13  #define _XOPEN_SOURCE
14  #define _XOPEN_SOURCE_EXTENDED  1
15  #define _HPUX_ALT_XOPEN_SOCKET_API
16  #endif
17
18
19  #if (NGX_TRU64)
20  #define _REENTRANT
21  #endif
22
23
```

```c
#ifdef __CYGWIN__
#define timezonevar             /* timezone is variable */
#define NGX_BROKEN_SCM_RIGHTS   1
#endif


#include <sys/types.h>
#include <sys/time.h>
#if (NGX_HAVE_UNISTD_H)
#include <unistd.h>
#endif
#if (NGX_HAVE_INTTYPES_H)
#include <inttypes.h>
#endif
#include <stdarg.h>
#include <stddef.h>             /* offsetof() */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <pwd.h>
#include <grp.h>
#include <dirent.h>
#include <glob.h>
#include <time.h>
#if (NGX_HAVE_SYS_PARAM_H)
#include <sys/param.h>          /* statfs() */
#endif
#if (NGX_HAVE_SYS_MOUNT_H)
#include <sys/mount.h>          /* statfs() */
#endif
#if (NGX_HAVE_SYS_STATVFS_H)
#include <sys/statvfs.h>        /* statvfs() */
#endif

#if (NGX_HAVE_SYS_FILIO_H)
#include <sys/filio.h>          /* FIONBIO */
#endif
#include <sys/ioctl.h>          /* FIONBIO */

#include <sys/uio.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/resource.h>
#include <sched.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>        /* TCP_NODELAY */
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/un.h>

#if (NGX_HAVE_LIMITS_H)
#include <limits.h>             /* IOV_MAX */
#endif

#ifdef __CYGWIN__
#include <malloc.h>             /* memalign() */
#endif

#if (NGX_HAVE_CRYPT_H)
#include <crypt.h>
#endif


#ifndef IOV_MAX
#define IOV_MAX   16
#endif


```

```
100 #include <ngx_auto_config.h>
101
102
103 #if (NGX_HAVE_POSIX_SEM)
104 #include <semaphore.h>
105 #endif
106
107
108 #if (NGX_HAVE_POLL)
109 #include <poll.h>
110 #endif
111
112
113 #if (NGX_HAVE_KQUEUE)
114 #include <sys/event.h>
115 #endif
116
117
118 #if (NGX_HAVE_DEVPOLL)
119 #include <sys/ioctl.h>
120 #include <sys/devpoll.h>
121 #endif
122
123
124 #if (NGX_HAVE_FILE_AIO)
125 #include <aio.h>
126 typedef struct aiocb  ngx_aiocb_t;
127 #endif
128
129
130 #define NGX_LISTEN_BACKLOG  511
131
132 #define ngx_debug_init()
133
134
135 #if (__FreeBSD__) && (__FreeBSD_version < 400017)
136
137 #include <sys/param.h>          /* ALIGN() */
138
139 /*
140 * FreeBSD 3.x has no CMSG_SPACE() and CMSG_LEN() and has the broken CMSG_DATA()
141 */
142
143 #undef  CMSG_SPACE
144 #define CMSG_SPACE(l)       (ALIGN(sizeof(struct cmsghdr)) + ALIGN(l))
145
146 #undef  CMSG_LEN
147 #define CMSG_LEN(l)         (ALIGN(sizeof(struct cmsghdr)) + (l))
148
149 #undef  CMSG_DATA
150 #define CMSG_DATA(cmsg)     ((u_char *)(cmsg) + ALIGN(sizeof(struct cmsghdr)))
151
152 #endif
153
154
155 extern char **environ;
156
157
158 #endif /* _NGX_POSIX_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_times.c - nginx-1.7.10

Global variables defined

- cached_err_log_time
- cached_gmtoff
- cached_http_log_iso8601
- cached_http_log_time
- cached_http_time
- cached_syslog_time
- cached_time
- months
- ngx_cached_err_log_time
- ngx_cached_http_log_iso8601
- ngx_cached_http_log_time
- ngx_cached_http_time
- ngx_cached_syslog_time
- ngx_cached_time
- ngx_current_msec
- ngx_time_lock
- slot
- week

Functions defined

- ngx_gmtime
- ngx_http_cookie_time
- ngx_http_time
- ngx_next_time
- ngx_time_init
- ngx_time_sigsafe_update
- ngx_time_update

Macros defined

- NGX_TIME_SLOTS

Source code

```c

/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>


/*
 * The time may be updated by signal handler or by several threads.
 * The time update operations are rare and require to hold the ngx_time_lock.
 * The time read operations are frequent, so they are lock-free and get time
 * values and strings from the current slot.  Thus thread may get the corrupted
 * values only if it is preempted while copying and then it is not scheduled
 * to run more than NGX_TIME_SLOTS seconds.
 */

#define NGX_TIME_SLOTS   64

static ngx_uint_t        slot;
static ngx_atomic_t      ngx_time_lock;

volatile ngx_msec_t      ngx_current_msec;
volatile ngx_time_t     *ngx_cached_time;
volatile ngx_str_t       ngx_cached_err_log_time;
volatile ngx_str_t       ngx_cached_http_time;
volatile ngx_str_t       ngx_cached_http_log_time;
volatile ngx_str_t       ngx_cached_http_log_iso8601;
volatile ngx_str_t       ngx_cached_syslog_time;

#if !(NGX_WIN32)

/*
 * localtime() and localtime_r() are not Async-Signal-Safe functions, therefore,
 * they must not be called by a signal handler, so we use the cached
 * GMT offset value. Fortunately the value is changed only two times a year.
 */

static ngx_int_t         cached_gmtoff;
#endif

static ngx_time_t        cached_time[NGX_TIME_SLOTS];
static u_char            cached_err_log_time[NGX_TIME_SLOTS]
                                    [sizeof("1970/09/28 12:00:00")];
static u_char            cached_http_time[NGX_TIME_SLOTS]
                                    [sizeof("Mon, 28 Sep 1970 06:00:00 GMT")];
static u_char            cached_http_log_time[NGX_TIME_SLOTS]
                                    [sizeof("28/Sep/1970:12:00:00 +0600")];
static u_char            cached_http_log_iso8601[NGX_TIME_SLOTS]
                                    [sizeof("1970-09-28T12:00:00+06:00")];
static u_char            cached_syslog_time[NGX_TIME_SLOTS]
                                    [sizeof("Sep 28 12:00:00")];


static char  *week[] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
static char  *months[] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                           "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

void
ngx_time_init(void)
{
    ngx_cached_err_log_time.len = sizeof("1970/09/28 12:00:00") - 1;
    ngx_cached_http_time.len = sizeof("Mon, 28 Sep 1970 06:00:00 GMT") - 1;
    ngx_cached_http_log_time.len = sizeof("28/Sep/1970:12:00:00 +0600") - 1;
    ngx_cached_http_log_iso8601.len = sizeof("1970-09-28T12:00:00+06:00") - 1;
    ngx_cached_syslog_time.len = sizeof("Sep 28 12:00:00") - 1;

    ngx_cached_time = &cached_time[0];

    ngx_time_update();
}


```

```
void
ngx_time_update(void)
{
    u_char          *p0, *p1, *p2, *p3, *p4;
    ngx_tm_t         tm, gmt;
    time_t           sec;
    ngx_uint_t       msec;
    ngx_time_t      *tp;
    struct timeval   tv;

    if (!ngx_trylock(&ngx_time_lock)) {
        return;
    }

    ngx_gettimeofday(&tv);

    sec = tv.tv_sec;
    msec = tv.tv_usec / 1000;

    ngx_current_msec = (ngx_msec_t) sec * 1000 + msec;

    tp = &cached_time[slot];

     if (tp->sec == sec) {
         tp->msec = msec;
         ngx_unlock(&ngx_time_lock);
         return;
     }

     if (slot == NGX_TIME_SLOTS - 1) {
         slot = 0;
     } else {
         slot++;
     }

     tp = &cached_time[slot];

     tp->sec = sec;
     tp->msec = msec;

     ngx_gmtime(sec, &gmt);


     p0 = &cached_http_time[slot][0];

     (void) ngx_sprintf(p0, "%s, %02d %s %4d %02d:%02d:%02d GMT",
                        week[gmt.ngx_tm_wday], gmt.ngx_tm_mday,
                        months[gmt.ngx_tm_mon - 1], gmt.ngx_tm_year,
                        gmt.ngx_tm_hour, gmt.ngx_tm_min, gmt.ngx_tm_sec);

#if (NGX_HAVE_GETTIMEZONE)

     tp->gmtoff = ngx_gettimezone();
     ngx_gmtime(sec + tp->gmtoff * 60, &tm);

#elif (NGX_HAVE_GMTOFF)

     ngx_localtime(sec, &tm);
     cached_gmtoff = (ngx_int_t) (tm.ngx_tm_gmtoff / 60);
     tp->gmtoff = cached_gmtoff;

#else

     ngx_localtime(sec, &tm);
     cached_gmtoff = ngx_timezone(tm.ngx_tm_isdst);
     tp->gmtoff = cached_gmtoff;

#endif


     p1 = &cached_err_log_time[slot][0];

     (void) ngx_sprintf(p1, "%4d/%02d/%02d %02d:%02d:%02d",
                        tm.ngx_tm_year, tm.ngx_tm_mon,
                        tm.ngx_tm_mday, tm.ngx_tm_hour,
                        tm.ngx_tm_min, tm.ngx_tm_sec);
```

```
    p2 = &cached_http_log_time[slot][0];

    (void) ngx_sprintf(p2, "%02d/%s/%d:%02d:%02d:%02d %c%02d%02d",
                       tm.ngx_tm_mday, months[tm.ngx_tm_mon - 1],
                       tm.ngx_tm_year, tm.ngx_tm_hour,
                       tm.ngx_tm_min, tm.ngx_tm_sec,
                       tp->gmtoff < 0 ? '-' : '+',
                       ngx_abs(tp->gmtoff / 60), ngx_abs(tp->gmtoff % 60));

    p3 = &cached_http_log_iso8601[slot][0];

    (void) ngx_sprintf(p3, "%4d-%02d-%02dT%02d:%02d:%02d%c%02d:%02d",
                       tm.ngx_tm_year, tm.ngx_tm_mon,
                       tm.ngx_tm_mday, tm.ngx_tm_hour,
                       tm.ngx_tm_min, tm.ngx_tm_sec,
                       tp->gmtoff < 0 ? '-' : '+',
                       ngx_abs(tp->gmtoff / 60), ngx_abs(tp->gmtoff % 60));

    p4 = &cached_syslog_time[slot][0];

    (void) ngx_sprintf(p4, "%s %2d %02d:%02d:%02d",
                       months[tm.ngx_tm_mon - 1], tm.ngx_tm_mday,
                       tm.ngx_tm_hour, tm.ngx_tm_min, tm.ngx_tm_sec);

    ngx_memory_barrier();

    ngx_cached_time = tp;
    ngx_cached_http_time.data = p0;
    ngx_cached_err_log_time.data = p1;
    ngx_cached_http_log_time.data = p2;
    ngx_cached_http_log_iso8601.data = p3;
    ngx_cached_syslog_time.data = p4;

    ngx_unlock(&ngx_time_lock);
}


#if !(NGX_WIN32)

void
ngx_time_sigsafe_update(void)
{
    u_char          *p, *p2;
    ngx_tm_t         tm;
    time_t           sec;
    ngx_time_t      *tp;
    struct timeval   tv;

    if (!ngx_trylock(&ngx_time_lock)) {
        return;
    }

    ngx_gettimeofday(&tv);

    sec = tv.tv_sec;

    tp = &cached_time[slot];

    if (tp->sec == sec) {
        ngx_unlock(&ngx_time_lock);
        return;
    }

    if (slot == NGX_TIME_SLOTS - 1) {
        slot = 0;
    } else {
        slot++;
    }

    tp = &cached_time[slot];

    tp->sec = 0;

    ngx_gmtime(sec + cached_gmtoff * 60, &tm);
```

```
229
230     p = &cached_err_log_time[slot][0];
231
232     (void) ngx_sprintf(p, "%4d/%02d/%02d %02d:%02d:%02d",
233                        tm.ngx_tm_year, tm.ngx_tm_mon,
234                        tm.ngx_tm_mday, tm.ngx_tm_hour,
235                        tm.ngx_tm_min, tm.ngx_tm_sec);
236
237     p2 = &cached_syslog_time[slot][0];
238
239     (void) ngx_sprintf(p2, "%s %2d %02d:%02d:%02d",
240                        months[tm.ngx_tm_mon - 1], tm.ngx_tm_mday,
241                        tm.ngx_tm_hour, tm.ngx_tm_min, tm.ngx_tm_sec);
242
243     ngx_memory_barrier();
244
245     ngx_cached_err_log_time.data = p;
246     ngx_cached_syslog_time.data = p2;
247
248     ngx_unlock(&ngx_time_lock);
249 }
250
251 #endif
252
253
254 u_char *
255 ngx_http_time(u_char *buf, time_t t)
256 {
257     ngx_tm_t  tm;
258
259     ngx_gmtime(t, &tm);
260
261     return ngx_sprintf(buf, "%s, %02d %s %4d %02d:%02d:%02d GMT",
262                        week[tm.ngx_tm_wday],
263                        tm.ngx_tm_mday,
264                        months[tm.ngx_tm_mon - 1],
265                        tm.ngx_tm_year,
266                        tm.ngx_tm_hour,
267                        tm.ngx_tm_min,
268                        tm.ngx_tm_sec);
269 }
270
271
272 u_char *
273 ngx_http_cookie_time(u_char *buf, time_t t)
274 {
275     ngx_tm_t  tm;
276
277     ngx_gmtime(t, &tm);
278
279     /*
280      * Netscape 3.x does not understand 4-digit years at all and
281      * 2-digit years more than "37"
282      */
283
284     return ngx_sprintf(buf,
285                        (tm.ngx_tm_year > 2037) ?
286                                        "%s, %02d-%s-%d %02d:%02d:%02d GMT":
287                                        "%s, %02d-%s-%02d %02d:%02d:%02d GMT",
288                        week[tm.ngx_tm_wday],
289                        tm.ngx_tm_mday,
290                        months[tm.ngx_tm_mon - 1],
291                        (tm.ngx_tm_year > 2037) ? tm.ngx_tm_year:
292                                                  tm.ngx_tm_year % 100,
293                        tm.ngx_tm_hour,
294                        tm.ngx_tm_min,
295                        tm.ngx_tm_sec);
296 }
297
298
299 void
300 ngx_gmtime(time_t t, ngx_tm_t *tp)
301 {
302     ngx_int_t   yday;
303     ngx_uint_t  n, sec, min, hour, mday, mon, year, wday, days, leap;
304
```

```
    /* the calculation is valid for positive time_t only */

    n = (ngx_uint_t) t;

    days = n / 86400;

    /* January 1, 1970 was Thursday */

    wday = (4 + days) % 7;

    n %= 86400;
    hour = n / 3600;
    n %= 3600;
    min = n / 60;
    sec = n % 60;

    /*
     * the algorithm based on Gauss' formula,
     * see src/http/ngx_http_parse_time.c
     */

    /* days since March 1, 1 BC */
    days = days - (31 + 28) + 719527;

    /*
     * The "days" should be adjusted to 1 only, however, some March 1st's go
     * to previous year, so we adjust them to 2.  This causes also shift of the
     * last February days to next year, but we catch the case when "yday"
     * becomes negative.
     */

    year = (days + 2) * 400 / (365 * 400 + 100 - 4 + 1);

    yday = days - (365 * year + year / 4 - year / 100 + year / 400);

    if (yday < 0) {
        leap = (year % 4 == 0) && (year % 100 || (year % 400 == 0));
        yday = 365 + leap + yday;
        year--;
    }

    /*
     * The empirical formula that maps "yday" to month.
     * There are at least 10 variants, some of them are:
     *     mon = (yday + 31) * 15 / 459
     *     mon = (yday + 31) * 17 / 520
     *     mon = (yday + 31) * 20 / 612
     */

    mon = (yday + 31) * 10 / 306;

    /* the Gauss' formula that evaluates days before the month */

    mday = yday - (367 * mon / 12 - 30) + 1;

    if (yday >= 306) {

        year++;
        mon -= 10;

        /*
         * there is no "yday" in Win32 SYSTEMTIME
         *
         * yday -= 306;
         */

    } else {

        mon += 2;

        /*
         * there is no "yday" in Win32 SYSTEMTIME
         *
         * yday += 31 + 28 + leap;
         */
    }
```

```
381
382     tp->ngx_tm_sec = (ngx_tm_sec_t) sec;
383     tp->ngx_tm_min = (ngx_tm_min_t) min;
384     tp->ngx_tm_hour = (ngx_tm_hour_t) hour;
385     tp->ngx_tm_mday = (ngx_tm_mday_t) mday;
386     tp->ngx_tm_mon = (ngx_tm_mon_t) mon;
387     tp->ngx_tm_year = (ngx_tm_year_t) year;
388     tp->ngx_tm_wday = (ngx_tm_wday_t) wday;
389 }
390
391
392 time_t
393 ngx_next_time(time_t when)
394 {
395     time_t     now, next;
396     struct tm  tm;
397
398     now = ngx_time();
399
400     ngx_libc_localtime(now, &tm);
401
402     tm.tm_hour = (int) (when / 3600);
403     when %= 3600;
404     tm.tm_min = (int) (when / 60);
405     tm.tm_sec = (int) (when % 60);
406
407     next = mktime(&tm);
408
409     if (next == -1) {
410         return -1;
411     }
412
413     if (next - now > 0) {
414         return next;
415     }
416
417     tm.tm_mday++;
418
419     /* mktime() should normalize a date (Jan 32, etc) */
420
421     next = mktime(&tm);
422
423     if (next != -1) {
424         return next;
425     }
426
427     return -1;
428 }
```

One Level Up Top Level

src/os/unix/ngx_atomic.h - nginx-1.7.10

Data types defined

- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_int_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t
- ngx_atomic_uint_t

Functions defined

- ngx_atomic_cmp_set
- ngx_atomic_fetch_add

Macros defined

- AO_REQUIRE_CAS
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_ATOMIC_T_LEN
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- NGX_HAVE_ATOMIC_OPS
- _NGX_ATOMIC_H_INCLUDED_
- ngx_atomic_cmp_set
- ngx_atomic_cmp_set

- ngx_atomic_cmp_set
- ngx_atomic_cmp_set
- ngx_atomic_fetch_add
- ngx_atomic_fetch_add
- ngx_atomic_fetch_add
- ngx_atomic_fetch_add
- ngx_cpu_pause
- ngx_cpu_pause
- ngx_cpu_pause
- ngx_cpu_pause
- ngx_cpu_pause
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_trylock
- ngx_unlock

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_ATOMIC_H_INCLUDED_
9  #define _NGX_ATOMIC_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #if (NGX_HAVE_LIBATOMIC)
17
18  #define AO_REQUIRE_CAS
19  #include <atomic_ops.h>
20
21  #define NGX_HAVE_ATOMIC_OPS  1
22
23  typedef long                        ngx_atomic_int_t;
24  typedef AO_t                        ngx_atomic_uint_t;
25  typedef volatile ngx_atomic_uint_t  ngx_atomic_t;
26
27  #if (NGX_PTR_SIZE == 8)
28  #define NGX_ATOMIC_T_LEN            (sizeof("-9223372036854775808") - 1)
29  #else
```

```
#define NGX_ATOMIC_T_LEN            (sizeof("-2147483648") - 1)
#endif

#define ngx_atomic_cmp_set(lock, old, new)                                   \
    AO_compare_and_swap(lock, old, new)
#define ngx_atomic_fetch_add(value, add)                                     \
    AO_fetch_and_add(value, add)
#define ngx_memory_barrier()        AO_nop()
#define ngx_cpu_pause()


#elif (NGX_DARWIN_ATOMIC)

/*
* use Darwin 8 atomic(3) and barrier(3) operations
* optimized at run-time for UP and SMP
*/

#include <libkern/OSAtomic.h>

/* "bool" conflicts with perl's CORE/handy.h */
#if 0
#undef bool
#endif


#define NGX_HAVE_ATOMIC_OPS  1

#if (NGX_PTR_SIZE == 8)

typedef int64_t                     ngx_atomic_int_t;
typedef uint64_t                    ngx_atomic_uint_t;
#define NGX_ATOMIC_T_LEN            (sizeof("-9223372036854775808") - 1)

#define ngx_atomic_cmp_set(lock, old, new)                                   \
    OSAtomicCompareAndSwap64Barrier(old, new, (int64_t *) lock)

#define ngx_atomic_fetch_add(value, add)                                     \
    (OSAtomicAdd64(add, (int64_t *) value) - add)

#else

typedef int32_t                     ngx_atomic_int_t;
typedef uint32_t                    ngx_atomic_uint_t;
#define NGX_ATOMIC_T_LEN            (sizeof("-2147483648") - 1)

#define ngx_atomic_cmp_set(lock, old, new)                                   \
    OSAtomicCompareAndSwap32Barrier(old, new, (int32_t *) lock)

#define ngx_atomic_fetch_add(value, add)                                     \
    (OSAtomicAdd32(add, (int32_t *) value) - add)

#endif

#define ngx_memory_barrier()        OSMemoryBarrier()

#define ngx_cpu_pause()

typedef volatile ngx_atomic_uint_t  ngx_atomic_t;


#elif (NGX_HAVE_GCC_ATOMIC)

/* GCC 4.1 builtin atomic operations */

#define NGX_HAVE_ATOMIC_OPS  1

typedef long                        ngx_atomic_int_t;
typedef unsigned long               ngx_atomic_uint_t;

 #if (NGX_PTR_SIZE == 8)
 #define NGX_ATOMIC_T_LEN            (sizeof("-9223372036854775808") - 1)
 #else
 #define NGX_ATOMIC_T_LEN            (sizeof("-2147483648") - 1)
 #endif

```

```
106 typedef volatile ngx_atomic_uint_t  ngx_atomic_t;
107
108
109 #define ngx_atomic_cmp_set(lock, old, set)                                   \
110     __sync_bool_compare_and_swap(lock, old, set)
111
112 #define ngx_atomic_fetch_add(value, add)                                     \
113     __sync_fetch_and_add(value, add)
114
115 #define ngx_memory_barrier()        __sync_synchronize()
116
117 #if ( __i386__ || __i386 || __amd64__ || __amd64 )
118 #define ngx_cpu_pause()             __asm__ ("pause")
119 #else
120 #define ngx_cpu_pause()
121 #endif
122
123
124 #elif ( __i386__ || __i386 )
125
126 typedef int32_t                     ngx_atomic_int_t;
127 typedef uint32_t                    ngx_atomic_uint_t;
128 typedef volatile ngx_atomic_uint_t  ngx_atomic_t;
129 #define NGX_ATOMIC_T_LEN            (sizeof("-2147483648") - 1)
130
131
132 #if ( __SUNPRO_C )
133
134 #define NGX_HAVE_ATOMIC_OPS  1
135
136 ngx_atomic_uint_t
137 ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
138     ngx_atomic_uint_t set);
139
140 ngx_atomic_int_t
141 ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add);
142
143 /*
144 * Sun Studio 12 exits with segmentation fault on '__asm ("pause")',
145 * so ngx_cpu_pause is declared in src/os/unix/ngx_sunpro_x86.il
146 */
147
148 void
149 ngx_cpu_pause(void);
150
151 /* the code in src/os/unix/ngx_sunpro_x86.il */
152
153 #define ngx_memory_barrier()        __asm (".volatile"); __asm (".nonvolatile")
154
155
156 #else /* ( __GNUC__ || __INTEL_COMPILER ) */
157
158 #define NGX_HAVE_ATOMIC_OPS  1
159
160 #include "ngx_gcc_atomic_x86.h"
161
162 #endif
163
164
165 #elif ( __amd64__ || __amd64 )
166
167 typedef int64_t                     ngx_atomic_int_t;
168 typedef uint64_t                    ngx_atomic_uint_t;
169 typedef volatile ngx_atomic_uint_t  ngx_atomic_t;
170 #define NGX_ATOMIC_T_LEN            (sizeof("-9223372036854775808") - 1)
171
172
173 #if ( __SUNPRO_C )
174
175 #define NGX_HAVE_ATOMIC_OPS  1
176
177 ngx_atomic_uint_t
178 ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
179     ngx_atomic_uint_t set);
180
181 ngx_atomic_int_t
```

```
ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add);

/*
* Sun Studio 12 exits with segmentation fault on '__asm ("pause")',
* so ngx_cpu_pause is declared in src/os/unix/ngx_sunpro_amd64.il
*/

void
ngx_cpu_pause(void);

/* the code in src/os/unix/ngx_sunpro_amd64.il */

#define ngx_memory_barrier()        __asm (".volatile"); __asm (".nonvolatile")


#else /* ( __GNUC__ || __INTEL_COMPILER ) */

#define NGX_HAVE_ATOMIC_OPS  1

#include "ngx_gcc_atomic_amd64.h"

#endif


#elif ( __sparc__ || __sparc || __sparcv9 )

#if (NGX_PTR_SIZE == 8)

typedef int64_t                     ngx_atomic_int_t;
typedef uint64_t                    ngx_atomic_uint_t;
#define NGX_ATOMIC_T_LEN            (sizeof("-9223372036854775808") - 1)

#else

typedef int32_t                     ngx_atomic_int_t;
typedef uint32_t                    ngx_atomic_uint_t;
#define NGX_ATOMIC_T_LEN            (sizeof("-2147483648") - 1)

#endif

typedef volatile ngx_atomic_uint_t  ngx_atomic_t;


#if ( __SUNPRO_C )

#define NGX_HAVE_ATOMIC_OPS  1

#include "ngx_sunpro_atomic_sparc64.h"


#else /* ( __GNUC__ || __INTEL_COMPILER ) */

#define NGX_HAVE_ATOMIC_OPS  1

#include "ngx_gcc_atomic_sparc64.h"

#endif


#elif ( __powerpc__ || __POWERPC__ )

#define NGX_HAVE_ATOMIC_OPS  1

#if (NGX_PTR_SIZE == 8)

typedef int64_t                     ngx_atomic_int_t;
typedef uint64_t                    ngx_atomic_uint_t;
#define NGX_ATOMIC_T_LEN            (sizeof("-9223372036854775808") - 1)

#else

typedef int32_t                     ngx_atomic_int_t;
typedef uint32_t                    ngx_atomic_uint_t;
#define NGX_ATOMIC_T_LEN            (sizeof("-2147483648") - 1)

#endif
```

```
258
259 typedef volatile ngx_atomic_uint_t  ngx_atomic_t;
260
261
262 #include "ngx_gcc_atomic_ppc.h"
263
264 #endif
265
266
267 #if !(NGX_HAVE_ATOMIC_OPS)
268
269 #define NGX_HAVE_ATOMIC_OPS  0
270
271 typedef int32_t                    ngx_atomic_int_t;
272 typedef uint32_t                   ngx_atomic_uint_t;
273 typedef volatile ngx_atomic_uint_t  ngx_atomic_t;
274 #define NGX_ATOMIC_T_LEN           (sizeof("-2147483648") - 1)
275
276
277 static ngx_inline ngx_atomic_uint_t
278 ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
279     ngx_atomic_uint_t set)
280 {
281     if (*lock == old) {
282         *lock = set;
283         return 1;
284     }
285
286     return 0;
287 }
288
289
290 static ngx_inline ngx_atomic_int_t
291 ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
292 {
293     ngx_atomic_int_t  old;
294
295     old = *value;
296     *value += add;
297
298     return old;
299 }
300
301 #define ngx_memory_barrier()
302 #define ngx_cpu_pause()
303
304 #endif
305
306
307 void ngx_spinlock(ngx_atomic_t *lock, ngx_atomic_int_t value, ngx_uint_t spin);
308
309 #define ngx_trylock(lock)  (*(lock) == 0 && ngx_atomic_cmp_set(lock, 0, 1))
310 #define ngx_unlock(lock)    *(lock) = 0
311
312
313 #endif /* _NGX_ATOMIC_H_INCLUDED_ */
```

One Level Up Top Level

src/http/modules/perl/ngx_http_perl_module.c - nginx-1.7.10

Global variables defined

- nginx_stash
- ngx_http_perl_commands
- ngx_http_perl_module
- ngx_http_perl_module_ctx
- ngx_http_perl_ssi_command
- ngx_http_perl_ssi_params
- ngx_null_name
- ngx_perl_term
- perl

Data types defined

- ngx_http_perl_loc_conf_t
- ngx_http_perl_main_conf_t
- ngx_http_perl_variable_t

Functions defined

- ngx_http_perl
- ngx_http_perl_call_handler
- ngx_http_perl_cleanup_perl
- ngx_http_perl_create_interpreter
- ngx_http_perl_create_loc_conf
- ngx_http_perl_create_main_conf
- ngx_http_perl_eval_anon_sub
- ngx_http_perl_exit
- ngx_http_perl_handle_request
- ngx_http_perl_handler
- ngx_http_perl_init_interpreter
- ngx_http_perl_init_main_conf
- ngx_http_perl_init_worker
- ngx_http_perl_merge_loc_conf
- ngx_http_perl_preconfiguration

- ngx_http_perl_run_requires
- ngx_http_perl_set
- ngx_http_perl_sleep_handler
- ngx_http_perl_ssi
- ngx_http_perl_variable
- ngx_http_perl_xs_init

Macros defined

- NGX_HTTP_PERL_SSI_ARG
- NGX_HTTP_PERL_SSI_SUB

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <ngx_http_perl_module.h>
12
13
14  typedef struct {
15      PerlInterpreter   *perl;
16      HV                *nginx;
17      ngx_array_t       *modules;
18      ngx_array_t       *requires;
19  } ngx_http_perl_main_conf_t;
20
21
22  typedef struct {
23      SV                *sub;
24      ngx_str_t          handler;
25  } ngx_http_perl_loc_conf_t;
26
27
28  typedef struct {
29      SV                *sub;
30      ngx_str_t          handler;
31  } ngx_http_perl_variable_t;
32
33
34  #if (NGX_HTTP_SSI)
35  static ngx_int_t ngx_http_perl_ssi(ngx_http_request_t *r,
36      ngx_http_ssi_ctx_t *ssi_ctx, ngx_str_t **params);
37  #endif
38
39  static char *ngx_http_perl_init_interpreter(ngx_conf_t *cf,
40      ngx_http_perl_main_conf_t *pmcf);
41  static PerlInterpreter *ngx_http_perl_create_interpreter(ngx_conf_t *cf,
42      ngx_http_perl_main_conf_t *pmcf);
43  static ngx_int_t ngx_http_perl_run_requires(pTHX_ ngx_array_t *requires,
44      ngx_log_t *log);
45  static ngx_int_t ngx_http_perl_call_handler(pTHX_ ngx_http_request_t *r,
46      HV *nginx, SV *sub, SV **args, ngx_str_t *handler, ngx_str_t *rv);
47  static void ngx_http_perl_eval_anon_sub(pTHX_ ngx_str_t *handler, SV **sv);
48
49  static ngx_int_t ngx_http_perl_preconfiguration(ngx_conf_t *cf);
50  static void *ngx_http_perl_create_main_conf(ngx_conf_t *cf);
51  static char *ngx_http_perl_init_main_conf(ngx_conf_t *cf, void *conf);
```

```
52  static void *ngx_http_perl_create_loc_conf(ngx_conf_t *cf);
53  static char *ngx_http_perl_merge_loc_conf(ngx_conf_t *cf, void *parent,
54      void *child);
55  static char *ngx_http_perl(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
56  static char *ngx_http_perl_set(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
57
58  #if (NGX_HAVE_PERL_MULTIPLICITY)
59  static void ngx_http_perl_cleanup_perl(void *data);
60  #endif
61
62  static ngx_int_t ngx_http_perl_init_worker(ngx_cycle_t *cycle);
63  static void ngx_http_perl_exit(ngx_cycle_t *cycle);
64
65
66  static ngx_command_t  ngx_http_perl_commands[] = {
67
68      { ngx_string("perl_modules"),
69        NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
70        ngx_conf_set_str_array_slot,
71        NGX_HTTP_MAIN_CONF_OFFSET,
72        offsetof(ngx_http_perl_main_conf_t, modules),
73        NULL },
74
75      { ngx_string("perl_require"),
76        NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
77        ngx_conf_set_str_array_slot,
78        NGX_HTTP_MAIN_CONF_OFFSET,
79        offsetof(ngx_http_perl_main_conf_t, requires),
80        NULL },
81
82      { ngx_string("perl"),
83        NGX_HTTP_LOC_CONF|NGX_HTTP_LMT_CONF|NGX_CONF_TAKE1,
84        ngx_http_perl,
85        NGX_HTTP_LOC_CONF_OFFSET,
86        0,
87        NULL },
88
89      { ngx_string("perl_set"),
90        NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE2,
91        ngx_http_perl_set,
92        NGX_HTTP_LOC_CONF_OFFSET,
93        0,
94        NULL },
95
96        ngx_null_command
97  };
98
99
100 static ngx_http_module_t  ngx_http_perl_module_ctx = {
101     ngx_http_perl_preconfiguration,        /* preconfiguration */
102     NULL,                                  /* postconfiguration */
103
104     ngx_http_perl_create_main_conf,        /* create main configuration */
105     ngx_http_perl_init_main_conf,          /* init main configuration */
106
107     NULL,                                  /* create server configuration */
108     NULL,                                  /* merge server configuration */
109
110     ngx_http_perl_create_loc_conf,         /* create location configuration */
111     ngx_http_perl_merge_loc_conf           /* merge location configuration */
112 };
113
114
115 ngx_module_t  ngx_http_perl_module = {
116     NGX_MODULE_V1,
117     &ngx_http_perl_module_ctx,             /* module context */
118     ngx_http_perl_commands,                /* module directives */
119     NGX_HTTP_MODULE,                       /* module type */
120     NULL,                                  /* init master */
121     NULL,                                  /* init module */
122     ngx_http_perl_init_worker,             /* init process */
123     NULL,                                  /* init thread */
124     NULL,                                  /* exit thread */
125     NULL,                                  /* exit process */
126     ngx_http_perl_exit,                    /* exit master */
127     NGX_MODULE_V1_PADDING
```

```
};


#if (NGX_HTTP_SSI)

#define NGX_HTTP_PERL_SSI_SUB  0
#define NGX_HTTP_PERL_SSI_ARG  1


static ngx_http_ssi_param_t  ngx_http_perl_ssi_params[] = {
    { ngx_string("sub"), NGX_HTTP_PERL_SSI_SUB, 1, 0 },
    { ngx_string("arg"), NGX_HTTP_PERL_SSI_ARG, 0, 1 },
    { ngx_null_string, 0, 0, 0 }
};

static ngx_http_ssi_command_t  ngx_http_perl_ssi_command = {
    ngx_string("perl"), ngx_http_perl_ssi, ngx_http_perl_ssi_params, 0, 0, 1
};

#endif


static ngx_str_t         ngx_null_name = ngx_null_string;
static HV               *nginx_stash;

#if (NGX_HAVE_PERL_MULTIPLICITY)
static ngx_uint_t        ngx_perl_term;
#else
static PerlInterpreter  *perl;
#endif


static void
ngx_http_perl_xs_init(pTHX)
{
    newXS("DynaLoader::boot_DynaLoader", boot_DynaLoader, __FILE__);

    nginx_stash = gv_stashpv("nginx", TRUE);
}


static ngx_int_t
ngx_http_perl_handler(ngx_http_request_t *r)
{
    r->main->count++;

    ngx_http_perl_handle_request(r);

    return NGX_DONE;
}


void
ngx_http_perl_handle_request(ngx_http_request_t *r)
{
    SV                        *sub;
    ngx_int_t                  rc;
    ngx_str_t                  uri, args, *handler;
    ngx_http_perl_ctx_t        *ctx;
    ngx_http_perl_loc_conf_t   *plcf;
    ngx_http_perl_main_conf_t  *pmcf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0, "perl handler");

    ctx = ngx_http_get_module_ctx(r, ngx_http_perl_module);

    if (ctx == NULL) {
        ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_perl_ctx_t));
        if (ctx == NULL) {
            ngx_http_finalize_request(r, NGX_ERROR);
            return;
        }

        ngx_http_set_ctx(r, ctx, ngx_http_perl_module);
    }

```

```
    pmcf = ngx_http_get_module_main_conf(r, ngx_http_perl_module);

    {

    dTHXa(pmcf->perl);
    PERL_SET_CONTEXT(pmcf->perl);

    if (ctx->next == NULL) {
        plcf = ngx_http_get_module_loc_conf(r, ngx_http_perl_module);
        sub = plcf->sub;
        handler = &plcf->handler;

    } else {
        sub = ctx->next;
        handler = &ngx_null_name;
        ctx->next = NULL;
    }

    rc = ngx_http_perl_call_handler(aTHX_ r, pmcf->nginx, sub, NULL, handler,
                                    NULL);

    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "perl handler done: %i", rc);

    if (rc == NGX_DONE) {
        ngx_http_finalize_request(r, rc);
        return;
    }

    if (rc > 600) {
        rc = NGX_OK;
    }

    if (ctx->redirect_uri.len) {
        uri = ctx->redirect_uri;
        args = ctx->redirect_args;

    } else {
        uri.len = 0;
    }

    ctx->filename.data = NULL;
    ctx->redirect_uri.len = 0;

    if (ctx->done || ctx->next) {
        ngx_http_finalize_request(r, NGX_DONE);
        return;
    }

    if (uri.len) {
        ngx_http_internal_redirect(r, &uri, &args);
        ngx_http_finalize_request(r, NGX_DONE);
        return;
    }

    if (rc == NGX_OK || rc == NGX_HTTP_OK) {
        ngx_http_send_special(r, NGX_HTTP_LAST);
        ctx->done = 1;
    }

    ngx_http_finalize_request(r, rc);
}


void
ngx_http_perl_sleep_handler(ngx_http_request_t *r)
{
    ngx_event_t  *wev;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "perl sleep handler");

    wev = r->connection->write;

```

```
    if (wev->timedout) {
        wev->timedout = 0;
        ngx_http_perl_handle_request(r);
        return;
    }

    if (ngx_handle_write_event(wev, 0) != NGX_OK) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
    }
}


static ngx_int_t
ngx_http_perl_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    ngx_http_perl_variable_t *pv = (ngx_http_perl_variable_t *) data;

    ngx_int_t                  rc;
    ngx_str_t                  value;
    ngx_http_perl_ctx_t       *ctx;
    ngx_http_perl_main_conf_t  *pmcf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "perl variable handler");

    ctx = ngx_http_get_module_ctx(r, ngx_http_perl_module);

    if (ctx == NULL) {
        ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_perl_ctx_t));
        if (ctx == NULL) {
            return NGX_ERROR;
        }

        ngx_http_set_ctx(r, ctx, ngx_http_perl_module);
    }

    pmcf = ngx_http_get_module_main_conf(r, ngx_http_perl_module);

    value.data = NULL;

    {

    dTHXa(pmcf->perl);
    PERL_SET_CONTEXT(pmcf->perl);

    rc = ngx_http_perl_call_handler(aTHX_ r, pmcf->nginx, pv->sub, NULL,
                                    &pv->handler, &value);

    }

    if (value.data) {
        v->len = value.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = value.data;

    } else {
        v->not_found = 1;
    }

    ctx->filename.data = NULL;
    ctx->redirect_uri.len = 0;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "perl variable done");

    return rc;
}


#if (NGX_HTTP_SSI)

static ngx_int_t
ngx_http_perl_ssi(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ssi_ctx,
```

```
    ngx_str_t **params)
{
    SV                          *sv, **asv;
    ngx_int_t                    rc;
    ngx_str_t                   *handler, **args;
    ngx_uint_t                   i;
    ngx_http_perl_ctx_t         *ctx;
    ngx_http_perl_main_conf_t   *pmcf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "perl ssi handler");

    ctx = ngx_http_get_module_ctx(r, ngx_http_perl_module);

    if (ctx == NULL) {
        ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_perl_ctx_t));
        if (ctx == NULL) {
            return NGX_ERROR;
        }

        ngx_http_set_ctx(r, ctx, ngx_http_perl_module);
    }

    pmcf = ngx_http_get_module_main_conf(r, ngx_http_perl_module);

    ctx->ssi = ssi_ctx;

    handler = params[NGX_HTTP_PERL_SSI_SUB];
    handler->data[handler->len] = '\0';

    {

    dTHXa(pmcf->perl);
    PERL_SET_CONTEXT(pmcf->perl);

#if 0

    /* the code is disabled to force the precompiled perl code using only */

    ngx_http_perl_eval_anon_sub(aTHX_ handler, &sv);

    if (sv == &PL_sv_undef) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "eval_pv(\"%V\") failed", handler);
        return NGX_ERROR;
    }

    if (sv == NULL) {
        sv = newSVpvn((char *) handler->data, handler->len);
    }

#endif

    sv = newSVpvn((char *) handler->data, handler->len);

    args = &params[NGX_HTTP_PERL_SSI_ARG];

    if (args) {

        for (i = 0; args[i]; i++) { /* void */ }

        asv = ngx_pcalloc(r->pool, (i + 1) * sizeof(SV *));

        if (asv == NULL) {
            SvREFCNT_dec(sv);
            return NGX_ERROR;
        }

        asv[0] = (SV *) (uintptr_t) i;

        for (i = 0; args[i]; i++) {
            asv[i + 1] = newSVpvn((char *) args[i]->data, args[i]->len);
        }

    } else {
        asv = NULL;
```

```
    }

    rc = ngx_http_perl_call_handler(aTHX_ r, pmcf->nginx, sv, asv, handler,
                                    NULL);

    SvREFCNT_dec(sv);

    }

    ctx->filename.data = NULL;
    ctx->redirect_uri.len = 0;
    ctx->ssi = NULL;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0, "perl ssi done");

    return rc;
}

#endif


static char *
ngx_http_perl_init_interpreter(ngx_conf_t *cf, ngx_http_perl_main_conf_t *pmcf)
{
    ngx_str_t           *m;
    ngx_uint_t           i;
#if (NGX_HAVE_PERL_MULTIPLICITY)
    ngx_pool_cleanup_t  *cln;

    cln = ngx_pool_cleanup_add(cf->pool, 0);
    if (cln == NULL) {
        return NGX_CONF_ERROR;
    }

#endif

#ifdef NGX_PERL_MODULES
    if (pmcf->modules == NGX_CONF_UNSET_PTR) {

        pmcf->modules = ngx_array_create(cf->pool, 1, sizeof(ngx_str_t));
        if (pmcf->modules == NULL) {
            return NGX_CONF_ERROR;
        }

        m = ngx_array_push(pmcf->modules);
        if (m == NULL) {
            return NGX_CONF_ERROR;
        }

        ngx_str_set(m, NGX_PERL_MODULES);
    }
#endif

    if (pmcf->modules != NGX_CONF_UNSET_PTR) {
        m = pmcf->modules->elts;
        for (i = 0; i < pmcf->modules->nelts; i++) {
            if (ngx_conf_full_name(cf->cycle, &m[i], 0) != NGX_OK) {
                return NGX_CONF_ERROR;
            }
        }
    }

#if !(NGX_HAVE_PERL_MULTIPLICITY)

    if (perl) {

        if (ngx_set_environment(cf->cycle, NULL) == NULL) {
            return NGX_CONF_ERROR;
        }

        if (ngx_http_perl_run_requires(aTHX_ pmcf->requires, cf->log)
            != NGX_OK)
        {
            return NGX_CONF_ERROR;
        }

```

```
508         pmcf->perl = perl;
509         pmcf->nginx = nginx_stash;
510 
511         return NGX_CONF_OK;
512     }
513 
514 #endif
515 
516     if (nginx_stash == NULL) {
517         PERL_SYS_INIT(&ngx_argc, &ngx_argv);
518     }
519 
520     pmcf->perl = ngx_http_perl_create_interpreter(cf, pmcf);
521 
522     if (pmcf->perl == NULL) {
523         return NGX_CONF_ERROR;
524     }
525 
526     pmcf->nginx = nginx_stash;
527 
528 #if (NGX_HAVE_PERL_MULTIPLICITY)
529 
530     cln->handler = ngx_http_perl_cleanup_perl;
531     cln->data = pmcf->perl;
532 
533 #else
534 
535     perl = pmcf->perl;
536 
537 #endif
538 
539     return NGX_CONF_OK;
540 }
541 
542 
543 static PerlInterpreter *
544 ngx_http_perl_create_interpreter(ngx_conf_t *cf,
545     ngx_http_perl_main_conf_t *pmcf)
546 {
547     int                n;
548     STRLEN             len;
549     SV                *sv;
550     char              *ver, **embedding;
551     ngx_str_t         *m;
552     ngx_uint_t         i;
553     PerlInterpreter   *perl;
554 
555     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, cf->log, 0, "create perl interpreter");
556 
557     if (ngx_set_environment(cf->cycle, NULL) == NULL) {
558         return NULL;
559     }
560 
561     perl = perl_alloc();
562     if (perl == NULL) {
563         ngx_log_error(NGX_LOG_ALERT, cf->log, 0, "perl_alloc() failed");
564         return NULL;
565     }
566 
567     {
568 
569     dTHXa(perl);
570     PERL_SET_CONTEXT(perl);
571 
572     perl_construct(perl);
573 
574 #ifdef PERL_EXIT_DESTRUCT_END
575     PL_exit_flags |= PERL_EXIT_DESTRUCT_END;
576 #endif
577 
578     n = (pmcf->modules != NGX_CONF_UNSET_PTR) ? pmcf->modules->nelts * 2 : 0;
579 
580     embedding = ngx_palloc(cf->pool, (5 + n) * sizeof(char *));
581     if (embedding == NULL) {
582         goto fail;
583     }
```

```
584
585     embedding[0] = "";
586
587     if (n++) {
588         m = pmcf->modules->elts;
589         for (i = 0; i < pmcf->modules->nelts; i++) {
590             embedding[2 * i + 1] = "-I";
591             embedding[2 * i + 2] = (char *) m[i].data;
592         }
593     }
594
595     embedding[n++] = "-Mnginx";
596     embedding[n++] = "-e";
597     embedding[n++] = "0";
598     embedding[n] = NULL;
599
600     n = perl_parse(perl, ngx_http_perl_xs_init, n, embedding, NULL);
601
602     if (n != 0) {
603         ngx_log_error(NGX_LOG_ALERT, cf->log, 0, "perl_parse() failed: %d", n);
604         goto fail;
605     }
606
607     sv = get_sv("nginx::VERSION", FALSE);
608     ver = SvPV(sv, len);
609
610     if (ngx_strcmp(ver, NGINX_VERSION) != 0) {
611         ngx_log_error(NGX_LOG_ALERT, cf->log, 0,
612                       "version " NGINX_VERSION " of nginx.pm is required, "
613                       "but %s was found", ver);
614         goto fail;
615     }
616
617     if (ngx_http_perl_run_requires(aTHX_ pmcf->requires, cf->log) != NGX_OK) {
618         goto fail;
619     }
620
621     }
622
623     return perl;
624
625 fail:
626
627     (void) perl_destruct(perl);
628
629     perl_free(perl);
630
631     return NULL;
632 }
633
634
635 static ngx_int_t
636 ngx_http_perl_run_requires(pTHX_ ngx_array_t *requires, ngx_log_t *log)
637 {
638     u_char      *err;
639     STRLEN       len;
640     ngx_str_t   *script;
641     ngx_uint_t   i;
642
643     if (requires == NGX_CONF_UNSET_PTR) {
644         return NGX_OK;
645     }
646
647     script = requires->elts;
648     for (i = 0; i < requires->nelts; i++) {
649
650         require_pv((char *) script[i].data);
651
652         if (SvTRUE(ERRSV)) {
653
654             err = (u_char *) SvPV(ERRSV, len);
655             while (--len && (err[len] == CR || err[len] == LF)) { /* void */ }
656
657             ngx_log_error(NGX_LOG_EMERG, log, 0,
658                           "require_pv(\"%s\") failed: \"%*s\"",
659                           script[i].data, len + 1, err);
```

```
660
661             return NGX_ERROR;
662         }
663     }
664
665     return NGX_OK;
666 }
667
668
669 static ngx_int_t
670 ngx_http_perl_call_handler(pTHX_ ngx_http_request_t *r, HV *nginx, SV *sub,
671     SV **args, ngx_str_t *handler, ngx_str_t *rv)
672 {
673     SV               *sv;
674     int               n, status;
675     char             *line;
676     u_char           *err;
677     STRLEN            len, n_a;
678     ngx_uint_t        i;
679     ngx_connection_t  *c;
680
681     dSP;
682
683     status = 0;
684
685     ENTER;
686     SAVETMPS;
687
688     PUSHMARK(sp);
689
690     sv = sv_2mortal(sv_bless(newRV_noinc(newSViv(PTR2IV(r))), nginx));
691     XPUSHs(sv);
692
693     if (args) {
694         EXTEND(sp, (intptr_t) args[0]);
695
696         for (i = 1; i <= (uintptr_t) args[0]; i++) {
697             PUSHs(sv_2mortal(args[i]));
698         }
699     }
700
701     PUTBACK;
702
703     c = r->connection;
704
705     n = call_sv(sub, G_EVAL);
706
707     SPAGAIN;
708
709     if (n) {
710         if (rv == NULL) {
711             status = POPi;
712
713             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
714                            "call_sv: %d", status);
715
716         } else {
717             line = SvPVx(POPs, n_a);
718             rv->len = n_a;
719
720             rv->data = ngx_pnalloc(r->pool, n_a);
721             if (rv->data == NULL) {
722                 return NGX_ERROR;
723             }
724
725             ngx_memcpy(rv->data, line, n_a);
726         }
727     }
728
729     PUTBACK;
730
731     FREETMPS;
732     LEAVE;
733
734     /* check $@ */
735
```

```
    if (SvTRUE(ERRSV)) {

        err = (u_char *) SvPV(ERRSV, len);
        while (--len && (err[len] == CR || err[len] == LF)) { /* void */ }

        ngx_log_error(NGX_LOG_ERR, c->log, 0,
                      "call_sv(\"%V\") failed: \"%*s\"", handler, len + 1, err);

        if (rv) {
            return NGX_ERROR;
        }

        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (n != 1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                      "call_sv(\"%V\") returned %d results", handler, n);
        status = NGX_OK;
    }

    if (rv) {
        return NGX_OK;
    }

    return (ngx_int_t) status;
}


static void
ngx_http_perl_eval_anon_sub(pTHX_ ngx_str_t *handler, SV **sv)
{
    u_char  *p;

    for (p = handler->data; *p; p++) {
        if (*p != ' ' && *p != '\t' && *p != CR && *p != LF) {
            break;
        }
    }

    if (ngx_strncmp(p, "sub ", 4) == 0
        || ngx_strncmp(p, "sub{", 4) == 0
        || ngx_strncmp(p, "use ", 4) == 0)
    {
        *sv = eval_pv((char *) p, FALSE);

        /* eval_pv() does not set ERRSV on failure */

        return;
    }

    *sv = NULL;
}


static void *
ngx_http_perl_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_perl_main_conf_t  *pmcf;

    pmcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_perl_main_conf_t));
    if (pmcf == NULL) {
        return NULL;
    }

    pmcf->modules = NGX_CONF_UNSET_PTR;
    pmcf->requires = NGX_CONF_UNSET_PTR;

    return pmcf;
}


static char *
ngx_http_perl_init_main_conf(ngx_conf_t *cf, void *conf)
{
    ngx_http_perl_main_conf_t *pmcf = conf;
```

```
812
813     if (pmcf->perl == NULL) {
814         if (ngx_http_perl_init_interpreter(cf, pmcf) != NGX_CONF_OK) {
815             return NGX_CONF_ERROR;
816         }
817     }
818
819     return NGX_CONF_OK;
820 }
821
822
823 #if (NGX_HAVE_PERL_MULTIPLICITY)
824
825 static void
826 ngx_http_perl_cleanup_perl(void *data)
827 {
828     PerlInterpreter  *perl = data;
829
830     PERL_SET_CONTEXT(perl);
831
832     (void) perl_destruct(perl);
833
834     perl_free(perl);
835
836     if (ngx_perl_term) {
837         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0, "perl term");
838
839         PERL_SYS_TERM();
840     }
841 }
842
843 #endif
844
845
846 static ngx_int_t
847 ngx_http_perl_preconfiguration(ngx_conf_t *cf)
848 {
849 #if (NGX_HTTP_SSI)
850     ngx_int_t                   rc;
851     ngx_http_ssi_main_conf_t  *smcf;
852
853     smcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_ssi_filter_module);
854
855     rc = ngx_hash_add_key(&smcf->commands, &ngx_http_perl_ssi_command.name,
856                           &ngx_http_perl_ssi_command, NGX_HASH_READONLY_KEY);
857
858     if (rc != NGX_OK) {
859         if (rc == NGX_BUSY) {
860             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
861                                "conflicting SSI command \"%V\"",
862                                &ngx_http_perl_ssi_command.name);
863         }
864
865         return NGX_ERROR;
866     }
867 #endif
868
869     return NGX_OK;
870 }
871
872
873 static void *
874 ngx_http_perl_create_loc_conf(ngx_conf_t *cf)
875 {
876     ngx_http_perl_loc_conf_t *plcf;
877
878     plcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_perl_loc_conf_t));
879     if (plcf == NULL) {
880         return NULL;
881     }
882
883     /*
884      * set by ngx_pcalloc():
885      *
886      *     plcf->handler = { 0, NULL };
887      */
```

```
888
889     return plcf;
890 }
891
892
893 static char *
894 ngx_http_perl_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
895 {
896     ngx_http_perl_loc_conf_t *prev = parent;
897     ngx_http_perl_loc_conf_t *conf = child;
898
899     if (conf->sub == NULL) {
900         conf->sub = prev->sub;
901         conf->handler = prev->handler;
902     }
903
904     return NGX_CONF_OK;
905 }
906
907
908 static char *
909 ngx_http_perl(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
910 {
911     ngx_http_perl_loc_conf_t *plcf = conf;
912
913     ngx_str_t                  *value;
914     ngx_http_core_loc_conf_t   *clcf;
915     ngx_http_perl_main_conf_t  *pmcf;
916
917     value = cf->args->elts;
918
919     if (plcf->handler.data) {
920         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
921                            "duplicate perl handler \"%V\"", &value[1]);
922         return NGX_CONF_ERROR;
923     }
924
925     pmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_perl_module);
926
927     if (pmcf->perl == NULL) {
928         if (ngx_http_perl_init_interpreter(cf, pmcf) != NGX_CONF_OK) {
929             return NGX_CONF_ERROR;
930         }
931     }
932
933     plcf->handler = value[1];
934
935     {
936
937     dTHXa(pmcf->perl);
938     PERL_SET_CONTEXT(pmcf->perl);
939
940     ngx_http_perl_eval_anon_sub(aTHX_ &value[1], &plcf->sub);
941
942     if (plcf->sub == &PL_sv_undef) {
943         ngx_conf_log_error(NGX_LOG_ERR, cf, 0,
944                            "eval_pv(\"%V\") failed", &value[1]);
945         return NGX_CONF_ERROR;
946     }
947
948     if (plcf->sub == NULL) {
949         plcf->sub = newSVpvn((char *) value[1].data, value[1].len);
950     }
951
952     }
953
954     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
955     clcf->handler = ngx_http_perl_handler;
956
957     return NGX_CONF_OK;
958 }
959
960
961 static char *
962 ngx_http_perl_set(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
963 {
```

```
964     ngx_int_t                   index;
965     ngx_str_t                  *value;
966     ngx_http_variable_t        *v;
967     ngx_http_perl_variable_t   *pv;
968     ngx_http_perl_main_conf_t  *pmcf;
969
970     value = cf->args->elts;
971
972     if (value[1].data[0] != '$') {
973         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
974                            "invalid variable name \"%V\"", &value[1]);
975         return NGX_CONF_ERROR;
976     }
977
978     value[1].len--;
979     value[1].data++;
980
981     v = ngx_http_add_variable(cf, &value[1], NGX_HTTP_VAR_CHANGEABLE);
982     if (v == NULL) {
983         return NGX_CONF_ERROR;
984     }
985
986     pv = ngx_palloc(cf->pool, sizeof(ngx_http_perl_variable_t));
987     if (pv == NULL) {
988         return NGX_CONF_ERROR;
989     }
990
991     index = ngx_http_get_variable_index(cf, &value[1]);
992     if (index == NGX_ERROR) {
993         return NGX_CONF_ERROR;
994     }
995
996     pmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_perl_module);
997
998     if (pmcf->perl == NULL) {
999         if (ngx_http_perl_init_interpreter(cf, pmcf) != NGX_CONF_OK) {
1000             return NGX_CONF_ERROR;
1001         }
1002     }
1003
1004     pv->handler = value[2];
1005
1006     {
1007
1008     dTHXa(pmcf->perl);
1009     PERL_SET_CONTEXT(pmcf->perl);
1010
1011     ngx_http_perl_eval_anon_sub(aTHX_ &value[2], &pv->sub);
1012
1013     if (pv->sub == &PL_sv_undef) {
1014         ngx_conf_log_error(NGX_LOG_ERR, cf, 0,
1015                            "eval_pv(\"%V\") failed", &value[2]);
1016         return NGX_CONF_ERROR;
1017     }
1018
1019     if (pv->sub == NULL) {
1020         pv->sub = newSVpvn((char *) value[2].data, value[2].len);
1021     }
1022
1023     }
1024
1025     v->get_handler = ngx_http_perl_variable;
1026     v->data = (uintptr_t) pv;
1027
1028     return NGX_CONF_OK;
1029 }
1030
1031
1032 static ngx_int_t
1033 ngx_http_perl_init_worker(ngx_cycle_t *cycle)
1034 {
1035     ngx_http_perl_main_conf_t  *pmcf;
1036
1037     pmcf = ngx_http_cycle_get_module_main_conf(cycle, ngx_http_perl_module);
1038
1039     if (pmcf) {
```

```
1040         dTHXa(pmcf->perl);
1041         PERL_SET_CONTEXT(pmcf->perl);
1042 
1043         /* set worker's $$ */
1044 
1045         sv_setiv(GvSV(gv_fetchpv("$", TRUE, SVt_PV)), (I32) ngx_pid);
1046     }
1047 
1048     return NGX_OK;
1049 }
1050 
1051 
1052 static void
1053 ngx_http_perl_exit(ngx_cycle_t *cycle)
1054 {
1055 #if (NGX_HAVE_PERL_MULTIPLICITY)
1056 
1057     /*
1058      * the master exit hook is run before global pool cleanup,
1059      * therefore just set flag here
1060      */
1061 
1062     ngx_perl_term = 1;
1063 
1064 #else
1065 
1066     if (nginx_stash) {
1067         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, cycle->log, 0, "perl term");
1068 
1069         (void) perl_destruct(perl);
1070 
1071         perl_free(perl);
1072 
1073         PERL_SYS_TERM();
1074     }
1075 
1076 #endif
1077 }
```

One Level Up Top Level

src/core/ngx_array.h - nginx-1.7.10

Data types defined

- ngx_array_t

Functions defined

- ngx_array_init

Macros defined

- _NGX_ARRAY_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_ARRAY_H_INCLUDED_
9  #define _NGX_ARRAY_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct {
17      void        *elts;
18      ngx_uint_t   nelts;
19      size_t       size;
20      ngx_uint_t   nalloc;
21      ngx_pool_t  *pool;
22  } ngx_array_t;
23
24
25  ngx_array_t *ngx_array_create(ngx_pool_t *p, ngx_uint_t n, size_t size);
26  void ngx_array_destroy(ngx_array_t *a);
27  void *ngx_array_push(ngx_array_t *a);
28  void *ngx_array_push_n(ngx_array_t *a, ngx_uint_t n);
29
30
31  static ngx_inline ngx_int_t
32  ngx_array_init(ngx_array_t *array, ngx_pool_t *pool, ngx_uint_t n, size_t size)
33  {
34      /*
35       * set "array->nelts" before "array->elts", otherwise MSVC thinks
36       * that "array->nelts" may be used without having been initialized
37       */
38
39      array->nelts = 0;
40      array->size = size;
41      array->nalloc = n;
42      array->pool = pool;
43
44      array->elts = ngx_palloc(pool, n * size);
45      if (array->elts == NULL) {
46          return NGX_ERROR;
47      }
48
49      return NGX_OK;
50  }
51
```

```
52
53  #endif /* _NGX_ARRAY_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_array.c - nginx-1.7.10

Functions defined

- ngx_array_create
- ngx_array_destroy
- ngx_array_push
- ngx_array_push_n

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 ngx_array_t *
13 ngx_array_create(ngx_pool_t *p, ngx_uint_t n, size_t size)
14 {
15     ngx_array_t *a;
16
17     a = ngx_palloc(p, sizeof(ngx_array_t));
18     if (a == NULL) {
19         return NULL;
20     }
21
22     if (ngx_array_init(a, p, n, size) != NGX_OK) {
23         return NULL;
24     }
25
26     return a;
27 }
28
29
30 void
31 ngx_array_destroy(ngx_array_t *a)
32 {
33     ngx_pool_t  *p;
34
35     p = a->pool;
36
37     if ((u_char *) a->elts + a->size * a->nalloc == p->d.last) {
38         p->d.last -= a->size * a->nalloc;
39     }
40
41     if ((u_char *) a + sizeof(ngx_array_t) == p->d.last) {
42         p->d.last = (u_char *) a;
43     }
44 }
45
46
47 void *
48 ngx_array_push(ngx_array_t *a)
49 {
50     void        *elt, *new;
51     size_t       size;
52     ngx_pool_t  *p;
53
54     if (a->nelts == a->nalloc) {
55
56         /* the array is full */
```

```
57
58          size = a->size * a->nalloc;
59
60          p = a->pool;
61
62          if ((u_char *) a->elts + size == p->d.last
63              && p->d.last + a->size <= p->d.end)
64          {
65              /*
66               * the array allocation is the last in the pool
67               * and there is space for new allocation
68               */
69
70              p->d.last += a->size;
71              a->nalloc++;
72
73          } else {
74              /* allocate a new array */
75
76              new = ngx_palloc(p, 2 * size);
77              if (new == NULL) {
78                  return NULL;
79              }
80
81              ngx_memcpy(new, a->elts, size);
82              a->elts = new;
83              a->nalloc *= 2;
84          }
85      }
86
87      elt = (u_char *) a->elts + a->size * a->nelts;
88      a->nelts++;
89
90      return elt;
91  }
92
93
94  void *
95  ngx_array_push_n(ngx_array_t *a, ngx_uint_t n)
96  {
97      void        *elt, *new;
98      size_t       size;
99      ngx_uint_t   nalloc;
100      ngx_pool_t  *p;
101
102      size = n * a->size;
103
104      if (a->nelts + n > a->nalloc) {
105
106          /* the array is full */
107
108          p = a->pool;
109
110          if ((u_char *) a->elts + a->size * a->nalloc == p->d.last
111              && p->d.last + size <= p->d.end)
112          {
113              /*
114               * the array allocation is the last in the pool
115               * and there is space for new allocation
116               */
117
118              p->d.last += size;
119              a->nalloc += n;
120
121          } else {
122              /* allocate a new array */
123
124              nalloc = 2 * ((n >= a->nalloc) ? n : a->nalloc);
125
126              new = ngx_palloc(p, nalloc * a->size);
127              if (new == NULL) {
128                  return NULL;
129              }
130
131              ngx_memcpy(new, a->elts, a->nelts * a->size);
132              a->elts = new;
```

```
133             a->nalloc = nalloc;
134         }
135     }
136
137     elt = (u_char *) a->elts + a->size * a->nelts;
138     a->nelts += n;
139
140     return elt;
141 }
```

One Level Up Top Level

src/http/ngx_http.h - nginx-1.7.10

Data types defined

- ngx_http_cache_t
- ngx_http_chunked_s
- ngx_http_chunked_t
- ngx_http_file_cache_t
- ngx_http_header_handler_pt
- ngx_http_log_ctx_s
- ngx_http_log_ctx_t
- ngx_http_log_handler_pt
- ngx_http_request_t
- ngx_http_spdy_stream_t
- ngx_http_status_t
- ngx_http_upstream_t

Macros defined

- NGX_HTTP_FLUSH
- NGX_HTTP_LAST
- _NGX_HTTP_H_INCLUDED_
- ngx_http_ephemeral
- ngx_http_get_module_ctx
- ngx_http_set_ctx

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_HTTP_H_INCLUDED_
9  #define _NGX_HTTP_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct ngx_http_request_s     ngx_http_request_t;
17  typedef struct ngx_http_upstream_s    ngx_http_upstream_t;
18  typedef struct ngx_http_cache_s       ngx_http_cache_t;
19  typedef struct ngx_http_file_cache_s  ngx_http_file_cache_t;
20  typedef struct ngx_http_log_ctx_s     ngx_http_log_ctx_t;
```

```
typedef struct ngx_http_chunked_s     ngx_http_chunked_t;

#if (NGX_HTTP_SPDY)
typedef struct ngx_http_spdy_stream_s  ngx_http_spdy_stream_t;
#endif

typedef ngx_int_t (*ngx_http_header_handler_pt)(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
typedef u_char *(*ngx_http_log_handler_pt)(ngx_http_request_t *r,
    ngx_http_request_t *sr, u_char *buf, size_t len);


#include <ngx_http_variables.h>
#include <ngx_http_config.h>
#include <ngx_http_request.h>
#include <ngx_http_script.h>
#include <ngx_http_upstream.h>
#include <ngx_http_upstream_round_robin.h>
#include <ngx_http_busy_lock.h>
#include <ngx_http_core_module.h>

#if (NGX_HTTP_SPDY)
#include <ngx_http_spdy.h>
#endif
#if (NGX_HTTP_CACHE)
#include <ngx_http_cache.h>
#endif
#if (NGX_HTTP_SSI)
#include <ngx_http_ssi_filter_module.h>
#endif
#if (NGX_HTTP_SSL)
#include <ngx_http_ssl_module.h>
#endif


struct ngx_http_log_ctx_s {
    ngx_connection_t    *connection;
    ngx_http_request_t  *request;
    ngx_http_request_t  *current_request;
};


struct ngx_http_chunked_s {
    ngx_uint_t           state;
    off_t                size;
    off_t                length;
};


typedef struct {
    ngx_uint_t           http_version;
    ngx_uint_t           code;
    ngx_uint_t           count;
    u_char              *start;
    u_char              *end;
} ngx_http_status_t;


#define ngx_http_get_module_ctx(r, module)  (r)->ctx[module.ctx_index]
#define ngx_http_set_ctx(r, c, module)      r->ctx[module.ctx_index] = c;


ngx_int_t ngx_http_add_location(ngx_conf_t *cf, ngx_queue_t **locations,
    ngx_http_core_loc_conf_t *clcf);
ngx_int_t ngx_http_add_listen(ngx_conf_t *cf, ngx_http_core_srv_conf_t *cscf,
    ngx_http_listen_opt_t *lsopt);


void ngx_http_init_connection(ngx_connection_t *c);
void ngx_http_close_connection(ngx_connection_t *c);

#if (NGX_HTTP_SSL && defined SSL_CTRL_SET_TLSEXT_HOSTNAME)
int ngx_http_ssl_servername(ngx_ssl_conn_t *ssl_conn, int *ad, void *arg);
#endif

ngx_int_t ngx_http_parse_request_line(ngx_http_request_t *r, ngx_buf_t *b);
```

```
97  ngx_int_t ngx_http_parse_uri(ngx_http_request_t *r);
98  ngx_int_t ngx_http_parse_complex_uri(ngx_http_request_t *r,
99      ngx_uint_t merge_slashes);
100 ngx_int_t ngx_http_parse_status_line(ngx_http_request_t *r, ngx_buf_t *b,
101     ngx_http_status_t *status);
102 ngx_int_t ngx_http_parse_unsafe_uri(ngx_http_request_t *r, ngx_str_t *uri,
103     ngx_str_t *args, ngx_uint_t *flags);
104 ngx_int_t ngx_http_parse_header_line(ngx_http_request_t *r, ngx_buf_t *b,
105     ngx_uint_t allow_underscores);
106 ngx_int_t ngx_http_parse_multi_header_lines(ngx_array_t *headers,
107     ngx_str_t *name, ngx_str_t *value);
108 ngx_int_t ngx_http_parse_set_cookie_lines(ngx_array_t *headers,
109     ngx_str_t *name, ngx_str_t *value);
110 ngx_int_t ngx_http_arg(ngx_http_request_t *r, u_char *name, size_t len,
111     ngx_str_t *value);
112 void ngx_http_split_args(ngx_http_request_t *r, ngx_str_t *uri,
113     ngx_str_t *args);
114 ngx_int_t ngx_http_parse_chunked(ngx_http_request_t *r, ngx_buf_t *b,
115     ngx_http_chunked_t *ctx);
116
117
118 ngx_http_request_t *ngx_http_create_request(ngx_connection_t *c);
119 ngx_int_t ngx_http_process_request_uri(ngx_http_request_t *r);
120 ngx_int_t ngx_http_process_request_header(ngx_http_request_t *r);
121 void ngx_http_process_request(ngx_http_request_t *r);
122 void ngx_http_update_location_config(ngx_http_request_t *r);
123 void ngx_http_handler(ngx_http_request_t *r);
124 void ngx_http_run_posted_requests(ngx_connection_t *c);
125 ngx_int_t ngx_http_post_request(ngx_http_request_t *r,
126     ngx_http_posted_request_t *pr);
127 void ngx_http_finalize_request(ngx_http_request_t *r, ngx_int_t rc);
128 void ngx_http_free_request(ngx_http_request_t *r, ngx_int_t rc);
129
130 void ngx_http_empty_handler(ngx_event_t *wev);
131 void ngx_http_request_empty_handler(ngx_http_request_t *r);
132
133
134 #define ngx_http_ephemeral(r)  (void *) (&r->uri_start)
135
136
137 #define NGX_HTTP_LAST   1
138 #define NGX_HTTP_FLUSH  2
139
140 ngx_int_t ngx_http_send_special(ngx_http_request_t *r, ngx_uint_t flags);
141
142
143 ngx_int_t ngx_http_read_client_request_body(ngx_http_request_t *r,
144     ngx_http_client_body_handler_pt post_handler);
145
146 ngx_int_t ngx_http_send_header(ngx_http_request_t *r);
147 ngx_int_t ngx_http_special_response_handler(ngx_http_request_t *r,
148     ngx_int_t error);
149 ngx_int_t ngx_http_filter_finalize_request(ngx_http_request_t *r,
150     ngx_module_t *m, ngx_int_t error);
151 void ngx_http_clean_header(ngx_http_request_t *r);
152
153
154 time_t ngx_http_parse_time(u_char *value, size_t len);
155 size_t ngx_http_get_time(char *buf, time_t t);
156
157
158
159 ngx_int_t ngx_http_discard_request_body(ngx_http_request_t *r);
160 void ngx_http_discarded_request_body_handler(ngx_http_request_t *r);
161 void ngx_http_block_reading(ngx_http_request_t *r);
162 void ngx_http_test_reading(ngx_http_request_t *r);
163
164
165 char *ngx_http_types_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
166 char *ngx_http_merge_types(ngx_conf_t *cf, ngx_array_t **keys,
167     ngx_hash_t *types_hash, ngx_array_t **prev_keys,
168     ngx_hash_t *prev_types_hash, ngx_str_t *default_types);
169 ngx_int_t ngx_http_set_default_types(ngx_conf_t *cf, ngx_array_t **types,
170     ngx_str_t *default_type);
171
172 #if (NGX_HTTP_DEGRADATION)
```

```
173 ngx_uint_t  ngx_http_degraded(ngx_http_request_t *);
174 #endif
175
176
177 extern ngx_module_t  ngx_http_module;
178
179 extern ngx_str_t  ngx_http_html_default_types[];
180
181
182 extern ngx_http_output_header_filter_pt  ngx_http_top_header_filter;
183 extern ngx_http_output_body_filter_pt    ngx_http_top_body_filter;
184
185
186 #endif /* _NGX_HTTP_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_request.h - nginx-1.7.10

Data types defined

- ngx_http_addr_conf_t
- ngx_http_cleanup_pt
- ngx_http_cleanup_s
- ngx_http_cleanup_t
- ngx_http_client_body_handler_pt
- ngx_http_connection_t
- ngx_http_ephemeral_t
- ngx_http_event_handler_pt
- ngx_http_handler_pt
- ngx_http_header_out_t
- ngx_http_header_t
- ngx_http_headers_in_t
- ngx_http_headers_out_t
- ngx_http_post_subrequest_pt
- ngx_http_post_subrequest_t
- ngx_http_posted_request_s
- ngx_http_posted_request_t
- ngx_http_postponed_request_s
- ngx_http_postponed_request_t
- ngx_http_request_body_t
- ngx_http_request_s
- ngx_http_state_e

Macros defined

- NGX_HTTPS_CERT_ERROR
- NGX_HTTPS_NO_CERT
- NGX_HTTP_ACCEPTED
- NGX_HTTP_BAD_GATEWAY
- NGX_HTTP_BAD_REQUEST
- NGX_HTTP_CLIENT_CLOSED_REQUEST

- NGX_HTTP_CLIENT_ERROR
- NGX_HTTP_CLOSE
- NGX_HTTP_CONFLICT
- NGX_HTTP_CONNECTION_CLOSE
- NGX_HTTP_CONNECTION_KEEP_ALIVE
- NGX_HTTP_CONTINUE
- NGX_HTTP_COPY
- NGX_HTTP_COPY_BUFFERED
- NGX_HTTP_CREATED
- NGX_HTTP_DELETE
- NGX_HTTP_DISCARD_BUFFER_SIZE
- NGX_HTTP_FORBIDDEN
- NGX_HTTP_GATEWAY_TIME_OUT
- NGX_HTTP_GET
- NGX_HTTP_GZIP_BUFFERED
- NGX_HTTP_HEAD
- NGX_HTTP_INSUFFICIENT_STORAGE
- NGX_HTTP_INTERNAL_SERVER_ERROR
- NGX_HTTP_LC_HEADER_LEN
- NGX_HTTP_LENGTH_REQUIRED
- NGX_HTTP_LINGERING_BUFFER_SIZE
- NGX_HTTP_LOCK
- NGX_HTTP_LOG_UNSAFE
- NGX_HTTP_LOWLEVEL_BUFFERED
- NGX_HTTP_MAX_SUBREQUESTS
- NGX_HTTP_MAX_URI_CHANGES
- NGX_HTTP_MKCOL
- NGX_HTTP_MOVE
- NGX_HTTP_MOVED_PERMANENTLY
- NGX_HTTP_MOVED_TEMPORARILY
- NGX_HTTP_NGINX_CODES
- NGX_HTTP_NOT_ALLOWED
- NGX_HTTP_NOT_FOUND

- NGX_HTTP_NOT_IMPLEMENTED
- NGX_HTTP_NOT_MODIFIED
- NGX_HTTP_NO_CONTENT
- NGX_HTTP_OK
- NGX_HTTP_OPTIONS
- NGX_HTTP_PARSE_HEADER_DONE
- NGX_HTTP_PARSE_INVALID_09_METHOD
- NGX_HTTP_PARSE_INVALID_HEADER
- NGX_HTTP_PARSE_INVALID_METHOD
- NGX_HTTP_PARSE_INVALID_REQUEST
- NGX_HTTP_PARTIAL_CONTENT
- NGX_HTTP_PATCH
- NGX_HTTP_POST
- NGX_HTTP_PRECONDITION_FAILED
- NGX_HTTP_PROCESSING
- NGX_HTTP_PROPFIND
- NGX_HTTP_PROPPATCH
- NGX_HTTP_PUT
- NGX_HTTP_RANGE_NOT_SATISFIABLE
- NGX_HTTP_REQUEST_ENTITY_TOO_LARGE
- NGX_HTTP_REQUEST_HEADER_TOO_LARGE
- NGX_HTTP_REQUEST_TIME_OUT
- NGX_HTTP_REQUEST_URI_TOO_LARGE
- NGX_HTTP_SEE_OTHER
- NGX_HTTP_SERVICE_UNAVAILABLE
- NGX_HTTP_SPECIAL_RESPONSE
- NGX_HTTP_SSI_BUFFERED
- NGX_HTTP_SUBREQUEST_IN_MEMORY
- NGX_HTTP_SUBREQUEST_WAITED
- NGX_HTTP_SUB_BUFFERED
- NGX_HTTP_SWITCHING_PROTOCOLS
- NGX_HTTP_TEMPORARY_REDIRECT
- NGX_HTTP_TO_HTTPS

- NGX_HTTP_TRACE
- NGX_HTTP_UNAUTHORIZED
- NGX_HTTP_UNKNOWN
- NGX_HTTP_UNLOCK
- NGX_HTTP_UNSUPPORTED_MEDIA_TYPE
- NGX_HTTP_VERSION_10
- NGX_HTTP_VERSION_11
- NGX_HTTP_VERSION_9
- NGX_HTTP_WRITE_BUFFERED
- NGX_NONE
- _NGX_HTTP_REQUEST_H_INCLUDED_
- ngx_http_set_connection_log
- ngx_http_set_log_request

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_REQUEST_H_INCLUDED_
9  #define _NGX_HTTP_REQUEST_H_INCLUDED_
10
11
12  #define NGX_HTTP_MAX_URI_CHANGES           10
13  #define NGX_HTTP_MAX_SUBREQUESTS           200
14
15  /* must be 2^n */
16  #define NGX_HTTP_LC_HEADER_LEN             32
17
18
19  #define NGX_HTTP_DISCARD_BUFFER_SIZE       4096
20  #define NGX_HTTP_LINGERING_BUFFER_SIZE     4096
21
22
23  #define NGX_HTTP_VERSION_9                 9
24  #define NGX_HTTP_VERSION_10                1000
25  #define NGX_HTTP_VERSION_11                1001
26
27  #define NGX_HTTP_UNKNOWN                   0x0001
28  #define NGX_HTTP_GET                       0x0002
29  #define NGX_HTTP_HEAD                      0x0004
30  #define NGX_HTTP_POST                      0x0008
31  #define NGX_HTTP_PUT                       0x0010
32  #define NGX_HTTP_DELETE                    0x0020
33  #define NGX_HTTP_MKCOL                     0x0040
34  #define NGX_HTTP_COPY                      0x0080
35  #define NGX_HTTP_MOVE                      0x0100
36  #define NGX_HTTP_OPTIONS                   0x0200
37  #define NGX_HTTP_PROPFIND                  0x0400
38  #define NGX_HTTP_PROPPATCH                 0x0800
39  #define NGX_HTTP_LOCK                      0x1000
40  #define NGX_HTTP_UNLOCK                    0x2000
41  #define NGX_HTTP_PATCH                     0x4000
42  #define NGX_HTTP_TRACE                     0x8000
43
```

```
#define NGX_HTTP_CONNECTION_CLOSE          1
#define NGX_HTTP_CONNECTION_KEEP_ALIVE     2


#define NGX_NONE                           1


#define NGX_HTTP_PARSE_HEADER_DONE         1

#define NGX_HTTP_CLIENT_ERROR              10
#define NGX_HTTP_PARSE_INVALID_METHOD      10
#define NGX_HTTP_PARSE_INVALID_REQUEST     11
#define NGX_HTTP_PARSE_INVALID_09_METHOD   12

#define NGX_HTTP_PARSE_INVALID_HEADER      13


/* unused                                  1 */
#define NGX_HTTP_SUBREQUEST_IN_MEMORY      2
#define NGX_HTTP_SUBREQUEST_WAITED         4
#define NGX_HTTP_LOG_UNSAFE                8


#define NGX_HTTP_CONTINUE                  100
#define NGX_HTTP_SWITCHING_PROTOCOLS       101
#define NGX_HTTP_PROCESSING                102

#define NGX_HTTP_OK                        200
#define NGX_HTTP_CREATED                   201
#define NGX_HTTP_ACCEPTED                  202
#define NGX_HTTP_NO_CONTENT                204
#define NGX_HTTP_PARTIAL_CONTENT           206

#define NGX_HTTP_SPECIAL_RESPONSE          300
#define NGX_HTTP_MOVED_PERMANENTLY         301
#define NGX_HTTP_MOVED_TEMPORARILY         302
#define NGX_HTTP_SEE_OTHER                 303
#define NGX_HTTP_NOT_MODIFIED              304
#define NGX_HTTP_TEMPORARY_REDIRECT        307

#define NGX_HTTP_BAD_REQUEST               400
#define NGX_HTTP_UNAUTHORIZED              401
#define NGX_HTTP_FORBIDDEN                 403
#define NGX_HTTP_NOT_FOUND                 404
#define NGX_HTTP_NOT_ALLOWED               405
#define NGX_HTTP_REQUEST_TIME_OUT          408
#define NGX_HTTP_CONFLICT                  409
#define NGX_HTTP_LENGTH_REQUIRED           411
#define NGX_HTTP_PRECONDITION_FAILED       412
#define NGX_HTTP_REQUEST_ENTITY_TOO_LARGE  413
#define NGX_HTTP_REQUEST_URI_TOO_LARGE     414
#define NGX_HTTP_UNSUPPORTED_MEDIA_TYPE    415
#define NGX_HTTP_RANGE_NOT_SATISFIABLE     416


/* Our own HTTP codes */

 /* The special code to close connection without any response */
 #define NGX_HTTP_CLOSE                     444

 #define NGX_HTTP_NGINX_CODES               494

 #define NGX_HTTP_REQUEST_HEADER_TOO_LARGE  494

 #define NGX_HTTPS_CERT_ERROR               495
 #define NGX_HTTPS_NO_CERT                  496

 /*
  * We use the special code for the plain HTTP requests that are sent to
  * HTTPS port to distinguish it from 4XX in an error page redirection
  */
 #define NGX_HTTP_TO_HTTPS                  497

 /* 498 is the canceled code for the requests with invalid host name */

 /*
```

```
 * HTTP does not define the code for the case when a client closed
 * the connection while we are processing its request so we introduce
 * own code to log such situation when a client has closed the connection
 * before we even try to send the HTTP header to it
 */
#define NGX_HTTP_CLIENT_CLOSED_REQUEST     499


#define NGX_HTTP_INTERNAL_SERVER_ERROR     500
#define NGX_HTTP_NOT_IMPLEMENTED           501
#define NGX_HTTP_BAD_GATEWAY               502
#define NGX_HTTP_SERVICE_UNAVAILABLE       503
#define NGX_HTTP_GATEWAY_TIME_OUT          504
#define NGX_HTTP_INSUFFICIENT_STORAGE      507


#define NGX_HTTP_LOWLEVEL_BUFFERED         0xf0
#define NGX_HTTP_WRITE_BUFFERED            0x10
#define NGX_HTTP_GZIP_BUFFERED             0x20
#define NGX_HTTP_SSI_BUFFERED              0x01
#define NGX_HTTP_SUB_BUFFERED              0x02
#define NGX_HTTP_COPY_BUFFERED             0x04


typedef enum {
    NGX_HTTP_INITING_REQUEST_STATE = 0,
    NGX_HTTP_READING_REQUEST_STATE,
    NGX_HTTP_PROCESS_REQUEST_STATE,

    NGX_HTTP_CONNECT_UPSTREAM_STATE,
    NGX_HTTP_WRITING_UPSTREAM_STATE,
    NGX_HTTP_READING_UPSTREAM_STATE,

    NGX_HTTP_WRITING_REQUEST_STATE,
    NGX_HTTP_LINGERING_CLOSE_STATE,
    NGX_HTTP_KEEPALIVE_STATE
} ngx_http_state_e;


typedef struct {
    ngx_str_t                       name;
    ngx_uint_t                      offset;
    ngx_http_header_handler_pt      handler;
} ngx_http_header_t;


typedef struct {
    ngx_str_t                       name;
    ngx_uint_t                      offset;
} ngx_http_header_out_t;


typedef struct {
    ngx_list_t                      headers;

    ngx_table_elt_t                *host;
    ngx_table_elt_t                *connection;
    ngx_table_elt_t                *if_modified_since;
    ngx_table_elt_t                *if_unmodified_since;
    ngx_table_elt_t                *if_match;
    ngx_table_elt_t                *if_none_match;
    ngx_table_elt_t                *user_agent;
    ngx_table_elt_t                *referer;
    ngx_table_elt_t                *content_length;
    ngx_table_elt_t                *content_type;

    ngx_table_elt_t                *range;
    ngx_table_elt_t                *if_range;

    ngx_table_elt_t                *transfer_encoding;
    ngx_table_elt_t                *expect;
    ngx_table_elt_t                *upgrade;

#if (NGX_HTTP_GZIP)
    ngx_table_elt_t                *accept_encoding;
    ngx_table_elt_t                *via;
```

```
#endif

    ngx_table_elt_t                  *authorization;

    ngx_table_elt_t                  *keep_alive;

#if (NGX_HTTP_X_FORWARDED_FOR)
    ngx_array_t                       x_forwarded_for;
#endif

#if (NGX_HTTP_REALIP)
    ngx_table_elt_t                  *x_real_ip;
#endif

#if (NGX_HTTP_HEADERS)
    ngx_table_elt_t                  *accept;
    ngx_table_elt_t                  *accept_language;
#endif

#if (NGX_HTTP_DAV)
    ngx_table_elt_t                  *depth;
    ngx_table_elt_t                  *destination;
    ngx_table_elt_t                  *overwrite;
    ngx_table_elt_t                  *date;
#endif

    ngx_str_t                         user;
    ngx_str_t                         passwd;

    ngx_array_t                       cookies;

    ngx_str_t                         server;
    off_t                             content_length_n;
    time_t                            keep_alive_n;

    unsigned                          connection_type:2;
    unsigned                          chunked:1;
    unsigned                          msie:1;
    unsigned                          msie6:1;
    unsigned                          opera:1;
    unsigned                          gecko:1;
    unsigned                          chrome:1;
    unsigned                          safari:1;
    unsigned                          konqueror:1;
} ngx_http_headers_in_t;


typedef struct {
    ngx_list_t                        headers;

    ngx_uint_t                        status;
    ngx_str_t                         status_line;

    ngx_table_elt_t                  *server;
    ngx_table_elt_t                  *date;
    ngx_table_elt_t                  *content_length;
    ngx_table_elt_t                  *content_encoding;
    ngx_table_elt_t                  *location;
    ngx_table_elt_t                  *refresh;
    ngx_table_elt_t                  *last_modified;
    ngx_table_elt_t                  *content_range;
    ngx_table_elt_t                  *accept_ranges;
    ngx_table_elt_t                  *www_authenticate;
    ngx_table_elt_t                  *expires;
    ngx_table_elt_t                  *etag;

    ngx_str_t                        *override_charset;

    size_t                            content_type_len;
    ngx_str_t                         content_type;
    ngx_str_t                         charset;
    u_char                           *content_type_lowcase;
    ngx_uint_t                        content_type_hash;

    ngx_array_t                       cache_control;

```

```
    off_t                            content_length_n;
    time_t                           date_time;
    time_t                           last_modified_time;
} ngx_http_headers_out_t;


typedef void (*ngx_http_client_body_handler_pt)(ngx_http_request_t *r);

typedef struct {
    ngx_temp_file_t                 *temp_file;
    ngx_chain_t                     *bufs;
    ngx_buf_t                       *buf;
    off_t                            rest;
    ngx_chain_t                     *free;
    ngx_chain_t                     *busy;
    ngx_http_chunked_t              *chunked;
    ngx_http_client_body_handler_pt  post_handler;
} ngx_http_request_body_t;


typedef struct ngx_http_addr_conf_s  ngx_http_addr_conf_t;

typedef struct {
    ngx_http_addr_conf_t            *addr_conf;
    ngx_http_conf_ctx_t             *conf_ctx;

#if (NGX_HTTP_SSL && defined SSL_CTRL_SET_TLSEXT_HOSTNAME)
    ngx_str_t                       *ssl_servername;
#if (NGX_PCRE)
    ngx_http_regex_t                *ssl_servername_regex;
#endif
#endif

    ngx_buf_t                      **busy;
    ngx_int_t                        nbusy;

    ngx_buf_t                      **free;
    ngx_int_t                        nfree;

#if (NGX_HTTP_SSL)
    unsigned                         ssl:1;
#endif
    unsigned                         proxy_protocol:1;
} ngx_http_connection_t;


typedef void (*ngx_http_cleanup_pt)(void *data);

typedef struct ngx_http_cleanup_s  ngx_http_cleanup_t;

struct ngx_http_cleanup_s {
    ngx_http_cleanup_pt              handler;
    void                            *data;
    ngx_http_cleanup_t              *next;
};


typedef ngx_int_t (*ngx_http_post_subrequest_pt)(ngx_http_request_t *r,
    void *data, ngx_int_t rc);

typedef struct {
    ngx_http_post_subrequest_pt      handler;
    void                            *data;
} ngx_http_post_subrequest_t;


typedef struct ngx_http_postponed_request_s  ngx_http_postponed_request_t;

struct ngx_http_postponed_request_s {
    ngx_http_request_t              *request;
    ngx_chain_t                     *out;
    ngx_http_postponed_request_t    *next;
};


typedef struct ngx_http_posted_request_s  ngx_http_posted_request_t;
```

```
348
349 struct ngx_http_posted_request_s {
350     ngx_http_request_t               *request;
351     ngx_http_posted_request_t        *next;
352 };
353
354
355 typedef ngx_int_t (*ngx_http_handler_pt)(ngx_http_request_t *r);
356 typedef void (*ngx_http_event_handler_pt)(ngx_http_request_t *r);
357
358
359 struct ngx_http_request_s {
360     uint32_t                          signature;         /* "HTTP" */
361
362     ngx_connection_t                 *connection;
363
364     void                            **ctx;
365     void                            **main_conf;
366     void                            **srv_conf;
367     void                            **loc_conf;
368
369     ngx_http_event_handler_pt         read_event_handler;
370     ngx_http_event_handler_pt         write_event_handler;
371
372 #if (NGX_HTTP_CACHE)
373     ngx_http_cache_t                 *cache;
374 #endif
375
376     ngx_http_upstream_t              *upstream;
377     ngx_array_t                      *upstream_states;
378                                          /* of ngx_http_upstream_state_t */
379
380     ngx_pool_t                       *pool;
381     ngx_buf_t                        *header_in;
382
383     ngx_http_headers_in_t             headers_in;
384     ngx_http_headers_out_t            headers_out;
385
386     ngx_http_request_body_t          *request_body;
387
388     time_t                            lingering_time;
389     time_t                            start_sec;
390     ngx_msec_t                        start_msec;
391
392     ngx_uint_t                        method;
393     ngx_uint_t                        http_version;
394
395     ngx_str_t                         request_line;
396     ngx_str_t                         uri;
397     ngx_str_t                         args;
398     ngx_str_t                         exten;
399     ngx_str_t                         unparsed_uri;
400
401     ngx_str_t                         method_name;
402     ngx_str_t                         http_protocol;
403
404     ngx_chain_t                      *out;
405     ngx_http_request_t               *main;
406     ngx_http_request_t               *parent;
407     ngx_http_postponed_request_t     *postponed;
408     ngx_http_post_subrequest_t       *post_subrequest;
409     ngx_http_posted_request_t        *posted_requests;
410
411     ngx_int_t                         phase_handler;
412     ngx_http_handler_pt               content_handler;
413     ngx_uint_t                        access_code;
414
415     ngx_http_variable_value_t        *variables;
416
417 #if (NGX_PCRE)
418     ngx_uint_t                        ncaptures;
419     int                              *captures;
420     u_char                           *captures_data;
421 #endif
422
423     size_t                            limit_rate;
```

```
    size_t                           limit_rate_after;

    /* used to learn the Apache compatible response length without a header */
    size_t                           header_size;

    off_t                            request_length;

    ngx_uint_t                       err_status;

    ngx_http_connection_t           *http_connection;
#if (NGX_HTTP_SPDY)
    ngx_http_spdy_stream_t          *spdy_stream;
#endif

    ngx_http_log_handler_pt          log_handler;

    ngx_http_cleanup_t              *cleanup;

    unsigned                         subrequests:8;
    unsigned                         count:8;
    unsigned                         blocked:8;

    unsigned                         aio:1;

    unsigned                         http_state:4;

    /* URI with "/." and on Win32 with "//" */
    unsigned                         complex_uri:1;

    /* URI with "%" */
    unsigned                         quoted_uri:1;

    /* URI with "+" */
    unsigned                         plus_in_uri:1;

    /* URI with " " */
    unsigned                         space_in_uri:1;

    unsigned                         invalid_header:1;

    unsigned                         add_uri_to_alias:1;
    unsigned                         valid_location:1;
    unsigned                         valid_unparsed_uri:1;
    unsigned                         uri_changed:1;
    unsigned                         uri_changes:4;

    unsigned                         request_body_in_single_buf:1;
    unsigned                         request_body_in_file_only:1;
    unsigned                         request_body_in_persistent_file:1;
    unsigned                         request_body_in_clean_file:1;
    unsigned                         request_body_file_group_access:1;
    unsigned                         request_body_file_log_level:3;

    unsigned                         subrequest_in_memory:1;
    unsigned                         waited:1;

#if (NGX_HTTP_CACHE)
    unsigned                         cached:1;
#endif

#if (NGX_HTTP_GZIP)
    unsigned                         gzip_tested:1;
    unsigned                         gzip_ok:1;
    unsigned                         gzip_vary:1;
#endif

    unsigned                         proxy:1;
    unsigned                         bypass_cache:1;
    unsigned                         no_cache:1;

    /*
     * instead of using the request context data in
     * ngx_http_limit_conn_module and ngx_http_limit_req_module
     * we use the single bits in the request structure
     */
    unsigned                         limit_conn_set:1;
```

```
500     unsigned                         limit_req_set:1;
501
502 #if 0
503     unsigned                         cacheable:1;
504 #endif
505
506     unsigned                         pipeline:1;
507     unsigned                         chunked:1;
508     unsigned                         header_only:1;
509     unsigned                         keepalive:1;
510     unsigned                         lingering_close:1;
511     unsigned                         discard_body:1;
512     unsigned                         internal:1;
513     unsigned                         error_page:1;
514     unsigned                         filter_finalize:1;
515     unsigned                         post_action:1;
516     unsigned                         request_complete:1;
517     unsigned                         request_output:1;
518     unsigned                         header_sent:1;
519     unsigned                         expect_tested:1;
520     unsigned                         root_tested:1;
521     unsigned                         done:1;
522     unsigned                         logged:1;
523
524     unsigned                         buffered:4;
525
526     unsigned                         main_filter_need_in_memory:1;
527     unsigned                         filter_need_in_memory:1;
528     unsigned                         filter_need_temporary:1;
529     unsigned                         allow_ranges:1;
530     unsigned                         single_range:1;
531     unsigned                         disable_not_modified:1;
532
533 #if (NGX_STAT_STUB)
534     unsigned                         stat_reading:1;
535     unsigned                         stat_writing:1;
536 #endif
537
538     /* used to parse HTTP headers */
539
540     ngx_uint_t                       state;
541
542     ngx_uint_t                       header_hash;
543     ngx_uint_t                       lowcase_index;
544     u_char                           lowcase_header[NGX_HTTP_LC_HEADER_LEN];
545
546     u_char                          *header_name_start;
547     u_char                          *header_name_end;
548     u_char                          *header_start;
549     u_char                          *header_end;
550
551     /*
552      * a memory that can be reused after parsing a request line
553      * via ngx_http_ephemeral_t
554      */
555
556     u_char                          *uri_start;
557     u_char                          *uri_end;
558     u_char                          *uri_ext;
559     u_char                          *args_start;
560     u_char                          *request_start;
561     u_char                          *request_end;
562     u_char                          *method_end;
563     u_char                          *schema_start;
564     u_char                          *schema_end;
565     u_char                          *host_start;
566     u_char                          *host_end;
567     u_char                          *port_start;
568     u_char                          *port_end;
569
570     unsigned                         http_minor:16;
571     unsigned                         http_major:16;
572 };
573
574
575 typedef struct {
```

```
    ngx_http_posted_request_t         terminal_posted_request;
#if (NGX_HAVE_AIO_SENDFILE)
    u_char                            aio_preload;
#endif
} ngx_http_ephemeral_t;


extern ngx_http_header_t       ngx_http_headers_in[];
extern ngx_http_header_out_t   ngx_http_headers_out[];


#define ngx_http_set_connection_log(c, l)                                     \
                                                                              \
    c->log->file = l->file;                                                   \
    c->log->next = l->next;                                                   \
    c->log->writer = l->writer;                                               \
    c->log->wdata = l->wdata;                                                 \
    if (!(c->log->log_level & NGX_LOG_DEBUG_CONNECTION)) {                    \
        c->log->log_level = l->log_level;                                     \
    }


#define ngx_http_set_log_request(log, r)                                      \
    ((ngx_http_log_ctx_t *) log->data)->current_request = r


#endif /* _NGX_HTTP_REQUEST_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_list.h - nginx-1.7.10

Data types defined

- ngx_list_part_s
- ngx_list_part_t
- ngx_list_t

Functions defined

- ngx_list_init

Macros defined

- _NGX_LIST_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_LIST_H_INCLUDED_
9  #define _NGX_LIST_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct ngx_list_part_s  ngx_list_part_t;
17
18  struct ngx_list_part_s {
19      void             *elts;
20      ngx_uint_t        nelts;
21      ngx_list_part_t  *next;
22  };
23
24
25  typedef struct {
26      ngx_list_part_t  *last;
27      ngx_list_part_t   part;
28      size_t            size;
29      ngx_uint_t        nalloc;
30      ngx_pool_t       *pool;
31  } ngx_list_t;
32
33
34  ngx_list_t *ngx_list_create(ngx_pool_t *pool, ngx_uint_t n, size_t size);
35
36  static ngx_inline ngx_int_t
37  ngx_list_init(ngx_list_t *list, ngx_pool_t *pool, ngx_uint_t n, size_t size)
38  {
39      list->part.elts = ngx_palloc(pool, n * size);
40      if (list->part.elts == NULL) {
41          return NGX_ERROR;
42      }
43
44      list->part.nelts = 0;
45      list->part.next = NULL;
46      list->last = &list->part;
```

```
    list->size = size;
    list->nalloc = n;
    list->pool = pool;

    return NGX_OK;
}


/*
*
*  the iteration through the list:
*
*  part = &list.part;
*  data = part->elts;
*
*  for (i = 0 ;; i++) {
*
*      if (i >= part->nelts) {
*          if (part->next == NULL) {
*              break;
*          }
*
*          part = part->next;
*          data = part->elts;
*          i = 0;
*      }
*
*      ...  data[i] ...
*
*  }
*/


void *ngx_list_push(ngx_list_t *list);


#endif /* _NGX_LIST_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_list.c - nginx-1.7.10

Functions defined

- ngx_list_create
- ngx_list_push

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 ngx_list_t *
13 ngx_list_create(ngx_pool_t *pool, ngx_uint_t n, size_t size)
14 {
15     ngx_list_t  *list;
16
17     list = ngx_palloc(pool, sizeof(ngx_list_t));
18     if (list == NULL) {
19         return NULL;
20     }
21
22     if (ngx_list_init(list, pool, n, size) != NGX_OK) {
23         return NULL;
24     }
25
26     return list;
27 }
28
29
30 void *
31 ngx_list_push(ngx_list_t *l)
32 {
33     void             *elt;
34     ngx_list_part_t  *last;
35
36     last = l->last;
37
38     if (last->nelts == l->nalloc) {
39
40         /* the last part is full, allocate a new list part */
41
42         last = ngx_palloc(l->pool, sizeof(ngx_list_part_t));
43         if (last == NULL) {
44             return NULL;
45         }
46
47         last->elts = ngx_palloc(l->pool, l->nalloc * l->size);
48         if (last->elts == NULL) {
49             return NULL;
50         }
51
52         last->nelts = 0;
53         last->next = NULL;
54
55         l->last->next = last;
56         l->last = last;
57     }
58
59     elt = (char *) last->elts + l->size * last->nelts;
60     last->nelts++;
```

```
61
62      return elt;
63  }
```

One Level Up Top Level

src/core/ngx_hash.h - nginx-1.7.10

Data types defined

- ngx_hash_combined_t
- ngx_hash_elt_t
- ngx_hash_init_t
- ngx_hash_key_pt
- ngx_hash_key_t
- ngx_hash_keys_arrays_t
- ngx_hash_t
- ngx_hash_wildcard_t
- ngx_table_elt_t

Macros defined

- NGX_HASH_LARGE
- NGX_HASH_LARGE_ASIZE
- NGX_HASH_LARGE_HSIZE
- NGX_HASH_READONLY_KEY
- NGX_HASH_SMALL
- NGX_HASH_WILDCARD_KEY
- _NGX_HASH_H_INCLUDED_
- ngx_hash

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HASH_H_INCLUDED_
9  #define _NGX_HASH_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct {
17      void             *value;
18      u_short           len;
19      u_char            name[1];
20  } ngx_hash_elt_t;
21
22
23  typedef struct {
```

```
    ngx_hash_elt_t  **buckets;
    ngx_uint_t        size;
} ngx_hash_t;


typedef struct {
    ngx_hash_t        hash;
    void             *value;
} ngx_hash_wildcard_t;


typedef struct {
    ngx_str_t         key;
    ngx_uint_t        key_hash;
    void             *value;
} ngx_hash_key_t;


typedef ngx_uint_t (*ngx_hash_key_pt) (u_char *data, size_t len);


typedef struct {
    ngx_hash_t            hash;
    ngx_hash_wildcard_t  *wc_head;
    ngx_hash_wildcard_t  *wc_tail;
} ngx_hash_combined_t;


typedef struct {
    ngx_hash_t       *hash;
    ngx_hash_key_pt   key;

    ngx_uint_t        max_size;
    ngx_uint_t        bucket_size;

    char             *name;
    ngx_pool_t       *pool;
    ngx_pool_t       *temp_pool;
} ngx_hash_init_t;


#define NGX_HASH_SMALL            1
#define NGX_HASH_LARGE            2

#define NGX_HASH_LARGE_ASIZE      16384
#define NGX_HASH_LARGE_HSIZE      10007

#define NGX_HASH_WILDCARD_KEY     1
#define NGX_HASH_READONLY_KEY     2


typedef struct {
    ngx_uint_t        hsize;

    ngx_pool_t       *pool;
    ngx_pool_t       *temp_pool;

    ngx_array_t       keys;
    ngx_array_t      *keys_hash;

    ngx_array_t       dns_wc_head;
    ngx_array_t      *dns_wc_head_hash;

    ngx_array_t       dns_wc_tail;
    ngx_array_t      *dns_wc_tail_hash;
} ngx_hash_keys_arrays_t;


typedef struct {
    ngx_uint_t        hash;
    ngx_str_t         key;
    ngx_str_t         value;
    u_char           *lowcase_key;
} ngx_table_elt_t;


```

```
100 void *ngx_hash_find(ngx_hash_t *hash, ngx_uint_t key, u_char *name, size_t len);
101 void *ngx_hash_find_wc_head(ngx_hash_wildcard_t *hwc, u_char *name, size_t len);
102 void *ngx_hash_find_wc_tail(ngx_hash_wildcard_t *hwc, u_char *name, size_t len);
103 void *ngx_hash_find_combined(ngx_hash_combined_t *hash, ngx_uint_t key,
104     u_char *name, size_t len);
105
106 ngx_int_t ngx_hash_init(ngx_hash_init_t *hinit, ngx_hash_key_t *names,
107     ngx_uint_t nelts);
108 ngx_int_t ngx_hash_wildcard_init(ngx_hash_init_t *hinit, ngx_hash_key_t *names,
109     ngx_uint_t nelts);
110
111 #define ngx_hash(key, c)   ((ngx_uint_t) key * 31 + c)
112 ngx_uint_t ngx_hash_key(u_char *data, size_t len);
113 ngx_uint_t ngx_hash_key_lc(u_char *data, size_t len);
114 ngx_uint_t ngx_hash_strlow(u_char *dst, u_char *src, size_t n);
115
116
117 ngx_int_t ngx_hash_keys_array_init(ngx_hash_keys_arrays_t *ha, ngx_uint_t type);
118 ngx_int_t ngx_hash_add_key(ngx_hash_keys_arrays_t *ha, ngx_str_t *key,
119     void *value, ngx_uint_t flags);
120
121
122 #endif /* _NGX_HASH_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_hash.c - nginx-1.7.10

Functions defined

- ngx_hash_add_key
- ngx_hash_find
- ngx_hash_find_combined
- ngx_hash_find_wc_head
- ngx_hash_find_wc_tail
- ngx_hash_init
- ngx_hash_key
- ngx_hash_key_lc
- ngx_hash_keys_array_init
- ngx_hash_strlow
- ngx_hash_wildcard_init

Macros defined

- NGX_HASH_ELT_SIZE

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  void *
13  ngx_hash_find(ngx_hash_t *hash, ngx_uint_t key, u_char *name, size_t len)
14  {
15      ngx_uint_t       i;
16      ngx_hash_elt_t  *elt;
17
18  #if 0
19      ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "hf:\"%*s\"", len, name);
20  #endif
21
22      elt = hash->buckets[key % hash->size];
23
24      if (elt == NULL) {
25          return NULL;
26      }
27
28      while (elt->value) {
29          if (len != (size_t) elt->len) {
30              goto next;
31          }
32
33          for (i = 0; i < len; i++) {
34              if (name[i] != elt->name[i]) {
```

```
                goto next;
            }
        }

        return elt->value;

    next:

        elt = (ngx_hash_elt_t *) ngx_align_ptr(&elt->name[0] + elt->len,
                                               sizeof(void *));
        continue;
    }

    return NULL;
}


void *
ngx_hash_find_wc_head(ngx_hash_wildcard_t *hwc, u_char *name, size_t len)
{
    void        *value;
    ngx_uint_t   i, n, key;

#if 0
    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "wch:\"%*s\"", len, name);
#endif

    n = len;

    while (n) {
        if (name[n - 1] == '.') {
            break;
        }

        n--;
    }

    key = 0;

    for (i = n; i < len; i++) {
        key = ngx_hash(key, name[i]);
    }

#if 0
    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "key:\"%ui\"", key);
#endif

    value = ngx_hash_find(&hwc->hash, key, &name[n], len - n);

#if 0
    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "value:\"%p\"", value);
#endif

    if (value) {

        /*
         * the 2 low bits of value have the special meaning:
         *     00 - value is data pointer for both "example.com"
         *          and "*.example.com";
         *     01 - value is data pointer for "*.example.com" only;
         *     10 - value is pointer to wildcard hash allowing
         *          both "example.com" and "*.example.com";
         *     11 - value is pointer to wildcard hash allowing
         *          "*.example.com" only.
         */

        if ((uintptr_t) value & 2) {

            if (n == 0) {

                /* "example.com" */

                if ((uintptr_t) value & 1) {
                    return NULL;
                }

```

```
                hwc = (ngx_hash_wildcard_t *)
                                          ((uintptr_t) value & (uintptr_t) ~3);
                return hwc->value;
            }

            hwc = (ngx_hash_wildcard_t *) ((uintptr_t) value & (uintptr_t) ~3);

            value = ngx_hash_find_wc_head(hwc, name, n - 1);

            if (value) {
                return value;
            }

            return hwc->value;
        }

        if ((uintptr_t) value & 1) {

            if (n == 0) {

                /* "example.com" */

                return NULL;
            }

            return (void *) ((uintptr_t) value & (uintptr_t) ~3);
        }

        return value;
    }

    return hwc->value;
}


void *
ngx_hash_find_wc_tail(ngx_hash_wildcard_t *hwc, u_char *name, size_t len)
{
    void        *value;
    ngx_uint_t   i, key;

#if 0
    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "wct:\"%*s\"", len, name);
#endif

    key = 0;

    for (i = 0; i < len; i++) {
        if (name[i] == '.') {
            break;
        }

        key = ngx_hash(key, name[i]);
    }

    if (i == len) {
        return NULL;
    }

#if 0
    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "key:\"%ui\"", key);
#endif

    value = ngx_hash_find(&hwc->hash, key, name, i);

#if 0
    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0, "value:\"%p\"", value);
#endif

    if (value) {

        /*
         * the 2 low bits of value have the special meaning:
         *     00 - value is data pointer;
         *     11 - value is pointer to wildcard hash allowing "example.*".
         */
```

```
        if ((uintptr_t) value & 2) {

            i++;

            hwc = (ngx_hash_wildcard_t *) ((uintptr_t) value & (uintptr_t) ~3);

            value = ngx_hash_find_wc_tail(hwc, &name[i], len - i);

            if (value) {
                return value;
            }

            return hwc->value;
        }

        return value;
    }

    return hwc->value;
}


void *
ngx_hash_find_combined(ngx_hash_combined_t *hash, ngx_uint_t key, u_char *name,
    size_t len)
{
    void  *value;

    if (hash->hash.buckets) {
        value = ngx_hash_find(&hash->hash, key, name, len);

        if (value) {
            return value;
        }
    }

    if (len == 0) {
        return NULL;
    }

    if (hash->wc_head && hash->wc_head->hash.buckets) {
        value = ngx_hash_find_wc_head(hash->wc_head, name, len);

        if (value) {
            return value;
        }
    }

    if (hash->wc_tail && hash->wc_tail->hash.buckets) {
        value = ngx_hash_find_wc_tail(hash->wc_tail, name, len);

        if (value) {
            return value;
        }
    }

    return NULL;
}


#define NGX_HASH_ELT_SIZE(name)                                               \
    (sizeof(void *) + ngx_align((name)->key.len + 2, sizeof(void *)))

ngx_int_t
ngx_hash_init(ngx_hash_init_t *hinit, ngx_hash_key_t *names, ngx_uint_t nelts)
{
    u_char          *elts;
    size_t           len;
    u_short         *test;
    ngx_uint_t       i, n, key, size, start, bucket_size;
    ngx_hash_elt_t  *elt, **buckets;

    for (n = 0; n < nelts; n++) {
        if (hinit->bucket_size < NGX_HASH_ELT_SIZE(&names[n]) + sizeof(void *))
        {
```

```
            ngx_log_error(NGX_LOG_EMERG, hinit->pool->log, 0,
                          "could not build the %s, you should "
                          "increase %s_bucket_size: %i",
                          hinit->name, hinit->name, hinit->bucket_size);
            return NGX_ERROR;
        }
    }

    test = ngx_alloc(hinit->max_size * sizeof(u_short), hinit->pool->log);
    if (test == NULL) {
        return NGX_ERROR;
    }

    bucket_size = hinit->bucket_size - sizeof(void *);

    start = nelts / (bucket_size / (2 * sizeof(void *)));
    start = start ? start : 1;

    if (hinit->max_size > 10000 && nelts && hinit->max_size / nelts < 100) {
        start = hinit->max_size - 1000;
    }

    for (size = start; size <= hinit->max_size; size++) {

        ngx_memzero(test, size * sizeof(u_short));

        for (n = 0; n < nelts; n++) {
            if (names[n].key.data == NULL) {
                continue;
            }

            key = names[n].key_hash % size;
            test[key] = (u_short) (test[key] + NGX_HASH_ELT_SIZE(&names[n]));

#if 0
            ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                          "%ui: %ui %ui \"%V\"",
                          size, key, test[key], &names[n].key);
#endif

            if (test[key] > (u_short) bucket_size) {
                goto next;
            }
        }

        goto found;

    next:

        continue;
    }

    size--;

    ngx_log_error(NGX_LOG_WARN, hinit->pool->log, 0,
                  "could not build optimal %s, you should increase "
                  "either %s_max_size: %i or %s_bucket_size: %i; "
                  "ignoring %s_bucket_size",
                  hinit->name, hinit->name, hinit->max_size,
                  hinit->name, hinit->bucket_size, hinit->name);

found:

    for (i = 0; i < size; i++) {
        test[i] = sizeof(void *);
    }

    for (n = 0; n < nelts; n++) {
        if (names[n].key.data == NULL) {
            continue;
        }

        key = names[n].key_hash % size;
        test[key] = (u_short) (test[key] + NGX_HASH_ELT_SIZE(&names[n]));
    }

```

```
    len = 0;

    for (i = 0; i < size; i++) {
        if (test[i] == sizeof(void *)) {
            continue;
        }

        test[i] = (u_short) (ngx_align(test[i], ngx_cacheline_size));

        len += test[i];
    }

    if (hinit->hash == NULL) {
        hinit->hash = ngx_pcalloc(hinit->pool, sizeof(ngx_hash_wildcard_t)
                                               + size * sizeof(ngx_hash_elt_t *));
        if (hinit->hash == NULL) {
            ngx_free(test);
            return NGX_ERROR;
        }

        buckets = (ngx_hash_elt_t **)
                      ((u_char *) hinit->hash + sizeof(ngx_hash_wildcard_t));

    } else {
        buckets = ngx_pcalloc(hinit->pool, size * sizeof(ngx_hash_elt_t *));
        if (buckets == NULL) {
            ngx_free(test);
            return NGX_ERROR;
        }
    }

    elts = ngx_palloc(hinit->pool, len + ngx_cacheline_size);
    if (elts == NULL) {
        ngx_free(test);
        return NGX_ERROR;
    }

    elts = ngx_align_ptr(elts, ngx_cacheline_size);

    for (i = 0; i < size; i++) {
        if (test[i] == sizeof(void *)) {
            continue;
        }

        buckets[i] = (ngx_hash_elt_t *) elts;
        elts += test[i];

    }

    for (i = 0; i < size; i++) {
        test[i] = 0;
    }

    for (n = 0; n < nelts; n++) {
        if (names[n].key.data == NULL) {
            continue;
        }

        key = names[n].key_hash % size;
        elt = (ngx_hash_elt_t *) ((u_char *) buckets[key] + test[key]);

        elt->value = names[n].value;
        elt->len = (u_short) names[n].key.len;

        ngx_strlow(elt->name, names[n].key.data, names[n].key.len);

        test[key] = (u_short) (test[key] + NGX_HASH_ELT_SIZE(&names[n]));
    }

    for (i = 0; i < size; i++) {
        if (buckets[i] == NULL) {
            continue;
        }

        elt = (ngx_hash_elt_t *) ((u_char *) buckets[i] + test[i]);

```

```
        elt->value = NULL;
    }

    ngx_free(test);

    hinit->hash->buckets = buckets;
    hinit->hash->size = size;

#if 0

    for (i = 0; i < size; i++) {
        ngx_str_t   val;
        ngx_uint_t  key;

        elt = buckets[i];

        if (elt == NULL) {
            ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                          "%ui: NULL", i);
            continue;
        }

        while (elt->value) {
            val.len = elt->len;
            val.data = &elt->name[0];

            key = hinit->key(val.data, val.len);

            ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                          "%ui: %p \"%V\" %ui", i, elt, &val, key);

            elt = (ngx_hash_elt_t *) ngx_align_ptr(&elt->name[0] + elt->len,
                                                   sizeof(void *));
        }
    }

#endif

    return NGX_OK;
}


ngx_int_t
ngx_hash_wildcard_init(ngx_hash_init_t *hinit, ngx_hash_key_t *names,
    ngx_uint_t nelts)
{
    size_t                len, dot_len;
    ngx_uint_t            i, n, dot;
    ngx_array_t           curr_names, next_names;
    ngx_hash_key_t       *name, *next_name;
    ngx_hash_init_t       h;
    ngx_hash_wildcard_t  *wdc;

    if (ngx_array_init(&curr_names, hinit->temp_pool, nelts,
                       sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&next_names, hinit->temp_pool, nelts,
                       sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    for (n = 0; n < nelts; n = i) {

#if 0
        ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                      "wc0: \"%V\"", &names[n].key);
#endif

        dot = 0;

```

```
        for (len = 0; len < names[n].key.len; len++) {
            if (names[n].key.data[len] == '.') {
                dot = 1;
                break;
            }
        }

        name = ngx_array_push(&curr_names);
        if (name == NULL) {
            return NGX_ERROR;
        }

        name->key.len = len;
        name->key.data = names[n].key.data;
        name->key_hash = hinit->key(name->key.data, name->key.len);
        name->value = names[n].value;

#if 0
        ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                      "wc1: \"%V\" %ui", &name->key, dot);
#endif

        dot_len = len + 1;

        if (dot) {
            len++;
        }

        next_names.nelts = 0;

        if (names[n].key.len != len) {
            next_name = ngx_array_push(&next_names);
            if (next_name == NULL) {
                return NGX_ERROR;
            }

            next_name->key.len = names[n].key.len - len;
            next_name->key.data = names[n].key.data + len;
            next_name->key_hash = 0;
            next_name->value = names[n].value;

#if 0
            ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                          "wc2: \"%V\"", &next_name->key);
#endif
        }

        for (i = n + 1; i < nelts; i++) {
            if (ngx_strncmp(names[n].key.data, names[i].key.data, len) != 0) {
                break;
            }

            if (!dot
                && names[i].key.len > len
                && names[i].key.data[len] != '.')
            {
                break;
            }

            next_name = ngx_array_push(&next_names);
            if (next_name == NULL) {
                return NGX_ERROR;
            }

            next_name->key.len = names[i].key.len - dot_len;
            next_name->key.data = names[i].key.data + dot_len;
            next_name->key_hash = 0;
            next_name->value = names[i].value;

#if 0
            ngx_log_error(NGX_LOG_ALERT, hinit->pool->log, 0,
                          "wc3: \"%V\"", &next_name->key);
#endif
        }

        if (next_names.nelts) {
```

```
            h = *hinit;
            h.hash = NULL;

            if (ngx_hash_wildcard_init(&h, (ngx_hash_key_t *) next_names.elts,
                                       next_names.nelts)
                != NGX_OK)
            {
                return NGX_ERROR;
            }

            wdc = (ngx_hash_wildcard_t *) h.hash;

            if (names[n].key.len == len) {
                wdc->value = names[n].value;
            }

            name->value = (void *) ((uintptr_t) wdc | (dot ? 3 : 2));

        } else if (dot) {
            name->value = (void *) ((uintptr_t) name->value | 1);
        }
    }

    if (ngx_hash_init(hinit, (ngx_hash_key_t *) curr_names.elts,
                      curr_names.nelts)
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    return NGX_OK;
}


ngx_uint_t
ngx_hash_key(u_char *data, size_t len)
{
    ngx_uint_t  i, key;

    key = 0;

    for (i = 0; i < len; i++) {
        key = ngx_hash(key, data[i]);
    }

    return key;
}


ngx_uint_t
ngx_hash_key_lc(u_char *data, size_t len)
{
    ngx_uint_t  i, key;

    key = 0;

    for (i = 0; i < len; i++) {
        key = ngx_hash(key, ngx_tolower(data[i]));
    }

    return key;
}


ngx_uint_t
ngx_hash_strlow(u_char *dst, u_char *src, size_t n)
{
    ngx_uint_t  key;

    key = 0;

    while (n--) {
        *dst = ngx_tolower(*src);
        key = ngx_hash(key, *dst);
        dst++;
```

```
        src++;
    }

    return key;
}


ngx_int_t
ngx_hash_keys_array_init(ngx_hash_keys_arrays_t *ha, ngx_uint_t type)
{
    ngx_uint_t  asize;

    if (type == NGX_HASH_SMALL) {
        asize = 4;
        ha->hsize = 107;

    } else {
        asize = NGX_HASH_LARGE_ASIZE;
        ha->hsize = NGX_HASH_LARGE_HSIZE;
    }

    if (ngx_array_init(&ha->keys, ha->temp_pool, asize, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&ha->dns_wc_head, ha->temp_pool, asize,
                       sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&ha->dns_wc_tail, ha->temp_pool, asize,
                       sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    ha->keys_hash = ngx_pcalloc(ha->temp_pool, sizeof(ngx_array_t) * ha->hsize);
    if (ha->keys_hash == NULL) {
        return NGX_ERROR;
    }

    ha->dns_wc_head_hash = ngx_pcalloc(ha->temp_pool,
                                       sizeof(ngx_array_t) * ha->hsize);
    if (ha->dns_wc_head_hash == NULL) {
        return NGX_ERROR;
    }

    ha->dns_wc_tail_hash = ngx_pcalloc(ha->temp_pool,
                                       sizeof(ngx_array_t) * ha->hsize);
    if (ha->dns_wc_tail_hash == NULL) {
        return NGX_ERROR;
    }

    return NGX_OK;
}


ngx_int_t
ngx_hash_add_key(ngx_hash_keys_arrays_t *ha, ngx_str_t *key, void *value,
    ngx_uint_t flags)
{
    size_t           len;
    u_char          *p;
    ngx_str_t       *name;
    ngx_uint_t       i, k, n, skip, last;
    ngx_array_t     *keys, *hwc;
    ngx_hash_key_t  *hk;

    last = key->len;

    if (flags & NGX_HASH_WILDCARD_KEY) {
```

```
719
720         /*
721          * supported wildcards:
722          *     "*.example.com", ".example.com", and "www.example.*"
723          */
724
725         n = 0;
726
727         for (i = 0; i < key->len; i++) {
728
729             if (key->data[i] == '*') {
730                 if (++n > 1) {
731                     return NGX_DECLINED;
732                 }
733             }
734
735             if (key->data[i] == '.' && key->data[i + 1] == '.') {
736                 return NGX_DECLINED;
737             }
738         }
739
740         if (key->len > 1 && key->data[0] == '.') {
741             skip = 1;
742             goto wildcard;
743         }
744
745         if (key->len > 2) {
746
747             if (key->data[0] == '*' && key->data[1] == '.') {
748                 skip = 2;
749                 goto wildcard;
750             }
751
752             if (key->data[i - 2] == '.' && key->data[i - 1] == '*') {
753                 skip = 0;
754                 last -= 2;
755                 goto wildcard;
756             }
757         }
758
759         if (n) {
760             return NGX_DECLINED;
761         }
762     }
763
764     /* exact hash */
765
766     k = 0;
767
768     for (i = 0; i < last; i++) {
769         if (!(flags & NGX_HASH_READONLY_KEY)) {
770             key->data[i] = ngx_tolower(key->data[i]);
771         }
772         k = ngx_hash(k, key->data[i]);
773     }
774
775     k %= ha->hsize;
776
777     /* check conflicts in exact hash */
778
779     name = ha->keys_hash[k].elts;
780
781     if (name) {
782         for (i = 0; i < ha->keys_hash[k].nelts; i++) {
783             if (last != name[i].len) {
784                 continue;
785             }
786
787             if (ngx_strncmp(key->data, name[i].data, last) == 0) {
788                 return NGX_BUSY;
789             }
790         }
791
792     } else {
793         if (ngx_array_init(&ha->keys_hash[k], ha->temp_pool, 4,
794                            sizeof(ngx_str_t))
```

```
            != NGX_OK)
        {
            return NGX_ERROR;
        }
    }

    name = ngx_array_push(&ha->keys_hash[k]);
    if (name == NULL) {
        return NGX_ERROR;
    }

    *name = *key;

    hk = ngx_array_push(&ha->keys);
    if (hk == NULL) {
        return NGX_ERROR;
    }

    hk->key = *key;
    hk->key_hash = ngx_hash_key(key->data, last);
    hk->value = value;

    return NGX_OK;


wildcard:

    /* wildcard hash */

    k = ngx_hash_strlow(&key->data[skip], &key->data[skip], last - skip);

    k %= ha->hsize;

    if (skip == 1) {

        /* check conflicts in exact hash for ".example.com" */

        name = ha->keys_hash[k].elts;

        if (name) {
            len = last - skip;

            for (i = 0; i < ha->keys_hash[k].nelts; i++) {
                if (len != name[i].len) {
                    continue;
                }

                if (ngx_strncmp(&key->data[1], name[i].data, len) == 0) {
                    return NGX_BUSY;
                }
            }

        } else {
            if (ngx_array_init(&ha->keys_hash[k], ha->temp_pool, 4,
                               sizeof(ngx_str_t))
                != NGX_OK)
            {
                return NGX_ERROR;
            }
        }

        name = ngx_array_push(&ha->keys_hash[k]);
        if (name == NULL) {
            return NGX_ERROR;
        }

        name->len = last - 1;
        name->data = ngx_pnalloc(ha->temp_pool, name->len);
        if (name->data == NULL) {
            return NGX_ERROR;
        }

        ngx_memcpy(name->data, &key->data[1], name->len);
    }


```

```
    if (skip) {

        /*
         * convert "*.example.com" to "com.example.\0"
         *      and ".example.com" to "com.example\0"
         */

        p = ngx_pnalloc(ha->temp_pool, last);
        if (p == NULL) {
            return NGX_ERROR;
        }

        len = 0;
        n = 0;

        for (i = last - 1; i; i--) {
            if (key->data[i] == '.') {
                ngx_memcpy(&p[n], &key->data[i + 1], len);
                n += len;
                p[n++] = '.';
                len = 0;
                continue;
            }

            len++;
        }

        if (len) {
            ngx_memcpy(&p[n], &key->data[1], len);
            n += len;
        }

        p[n] = '\0';

        hwc = &ha->dns_wc_head;
        keys = &ha->dns_wc_head_hash[k];

    } else {

        /* convert "www.example.*" to "www.example\0" */

        last++;

        p = ngx_pnalloc(ha->temp_pool, last);
        if (p == NULL) {
            return NGX_ERROR;
        }

        ngx_cpystrn(p, key->data, last);

        hwc = &ha->dns_wc_tail;
        keys = &ha->dns_wc_tail_hash[k];
    }


    /* check conflicts in wildcard hash */

    name = keys->elts;

    if (name) {
        len = last - skip;

        for (i = 0; i < keys->nelts; i++) {
            if (len != name[i].len) {
                continue;
            }

            if (ngx_strncmp(key->data + skip, name[i].data, len) == 0) {
                return NGX_BUSY;
            }
        }

    } else {
        if (ngx_array_init(keys, ha->temp_pool, 4, sizeof(ngx_str_t)) != NGX_OK)
        {
            return NGX_ERROR;
```

```
947         }
948     }
949
950     name = ngx_array_push(keys);
951     if (name == NULL) {
952         return NGX_ERROR;
953     }
954
955     name->len = last - skip;
956     name->data = ngx_pnalloc(ha->temp_pool, name->len);
957     if (name->data == NULL) {
958         return NGX_ERROR;
959     }
960
961     ngx_memcpy(name->data, key->data + skip, name->len);
962
963
964     /* add to wildcard hash */
965
966     hk = ngx_array_push(hwc);
967     if (hk == NULL) {
968         return NGX_ERROR;
969     }
970
971     hk->key.len = last - 1;
972     hk->key.data = p;
973     hk->key_hash = 0;
974     hk->value = value;
975
976     return NGX_OK;
977 }
```

One Level Up Top Level

src/core/ngx_cycle.c - nginx-1.7.10

Global variables defined

- dumb
- ngx_cleaner_event
- ngx_core_tls_key
- ngx_cycle
- ngx_old_cycles
- ngx_quiet_mode
- ngx_temp_pool
- ngx_test_config

Functions defined

- ngx_clean_old_cycles
- ngx_create_pidfile
- ngx_delete_pidfile
- ngx_destroy_cycle_pools
- ngx_init_cycle
- ngx_init_zone_pool
- ngx_reopen_files
- ngx_shared_memory_add
- ngx_signal_process
- ngx_test_lockfile

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  static void ngx_destroy_cycle_pools(ngx_conf_t *conf);
14  static ngx_int_t ngx_init_zone_pool(ngx_cycle_t *cycle,
15      ngx_shm_zone_t *shm_zone);
16  static ngx_int_t ngx_test_lockfile(u_char *file, ngx_log_t *log);
17  static void ngx_clean_old_cycles(ngx_event_t *ev);
18
19
20  volatile ngx_cycle_t  *ngx_cycle;
```

```
ngx_array_t            ngx_old_cycles;

static ngx_pool_t     *ngx_temp_pool;
static ngx_event_t     ngx_cleaner_event;

ngx_uint_t             ngx_test_config;
ngx_uint_t             ngx_quiet_mode;

#if (NGX_THREADS)
ngx_tls_key_t          ngx_core_tls_key;
#endif


/* STUB NAME */
static ngx_connection_t  dumb;
/* STUB */


ngx_cycle_t *
ngx_init_cycle(ngx_cycle_t *old_cycle)
{
    void                *rv;
    char               **senv, **env;
    ngx_uint_t           i, n;
    ngx_log_t           *log;
    ngx_time_t          *tp;
    ngx_conf_t           conf;
    ngx_pool_t          *pool;
    ngx_cycle_t         *cycle, **old;
    ngx_shm_zone_t      *shm_zone, *oshm_zone;
    ngx_list_part_t     *part, *opart;
    ngx_open_file_t     *file;
    ngx_listening_t     *ls, *nls;
    ngx_core_conf_t     *ccf, *old_ccf;
    ngx_core_module_t   *module;
    char                 hostname[NGX_MAXHOSTNAMELEN];

    ngx_timezone_update();

    /* force localtime update with a new timezone */

    tp = ngx_timeofday();
    tp->sec = 0;

    ngx_time_update();


    log = old_cycle->log;

    pool = ngx_create_pool(NGX_CYCLE_POOL_SIZE, log);
    if (pool == NULL) {
        return NULL;
    }
    pool->log = log;

    cycle = ngx_pcalloc(pool, sizeof(ngx_cycle_t));
    if (cycle == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    cycle->pool = pool;
    cycle->log = log;
    cycle->old_cycle = old_cycle;

    cycle->conf_prefix.len = old_cycle->conf_prefix.len;
    cycle->conf_prefix.data = ngx_pstrdup(pool, &old_cycle->conf_prefix);
    if (cycle->conf_prefix.data == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    cycle->prefix.len = old_cycle->prefix.len;
    cycle->prefix.data = ngx_pstrdup(pool, &old_cycle->prefix);
    if (cycle->prefix.data == NULL) {
        ngx_destroy_pool(pool);
```

```
        return NULL;
    }

    cycle->conf_file.len = old_cycle->conf_file.len;
    cycle->conf_file.data = ngx_pnalloc(pool, old_cycle->conf_file.len + 1);
    if (cycle->conf_file.data == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }
    ngx_cpystrn(cycle->conf_file.data, old_cycle->conf_file.data,
                old_cycle->conf_file.len + 1);

    cycle->conf_param.len = old_cycle->conf_param.len;
    cycle->conf_param.data = ngx_pstrdup(pool, &old_cycle->conf_param);
    if (cycle->conf_param.data == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }


    n = old_cycle->paths.nelts ? old_cycle->paths.nelts : 10;

    cycle->paths.elts = ngx_pcalloc(pool, n * sizeof(ngx_path_t *));
    if (cycle->paths.elts == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    cycle->paths.nelts = 0;
    cycle->paths.size = sizeof(ngx_path_t *);
    cycle->paths.nalloc = n;
    cycle->paths.pool = pool;


    if (old_cycle->open_files.part.nelts) {
        n = old_cycle->open_files.part.nelts;
        for (part = old_cycle->open_files.part.next; part; part = part->next) {
            n += part->nelts;
        }

    } else {
        n = 20;
    }

    if (ngx_list_init(&cycle->open_files, pool, n, sizeof(ngx_open_file_t))
        != NGX_OK)
    {
        ngx_destroy_pool(pool);
        return NULL;
    }


    if (old_cycle->shared_memory.part.nelts) {
        n = old_cycle->shared_memory.part.nelts;
        for (part = old_cycle->shared_memory.part.next; part; part = part->next)
        {
            n += part->nelts;
        }

    } else {
        n = 1;
    }

    if (ngx_list_init(&cycle->shared_memory, pool, n, sizeof(ngx_shm_zone_t))
        != NGX_OK)
    {
        ngx_destroy_pool(pool);
        return NULL;
    }

    n = old_cycle->listening.nelts ? old_cycle->listening.nelts : 10;

    cycle->listening.elts = ngx_pcalloc(pool, n * sizeof(ngx_listening_t));
    if (cycle->listening.elts == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
```

```
    }

    cycle->listening.nelts = 0;
    cycle->listening.size = sizeof(ngx_listening_t);
    cycle->listening.nalloc = n;
    cycle->listening.pool = pool;


    ngx_queue_init(&cycle->reusable_connections_queue);


    cycle->conf_ctx = ngx_pcalloc(pool, ngx_max_module * sizeof(void *));
    if (cycle->conf_ctx == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }


    if (gethostname(hostname, NGX_MAXHOSTNAMELEN) == -1) {
        ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "gethostname() failed");
        ngx_destroy_pool(pool);
        return NULL;
    }

    /* on Linux gethostname() silently truncates name that does not fit */

    hostname[NGX_MAXHOSTNAMELEN - 1] = '\0';
    cycle->hostname.len = ngx_strlen(hostname);

    cycle->hostname.data = ngx_pnalloc(pool, cycle->hostname.len);
    if (cycle->hostname.data == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    ngx_strlow(cycle->hostname.data, (u_char *) hostname, cycle->hostname.len);


    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_CORE_MODULE) {
            continue;
        }

        module = ngx_modules[i]->ctx;

        if (module->create_conf) {
            rv = module->create_conf(cycle);
            if (rv == NULL) {
                ngx_destroy_pool(pool);
                return NULL;
            }
            cycle->conf_ctx[ngx_modules[i]->index] = rv;
        }
    }


    senv = environ;


    ngx_memzero(&conf, sizeof(ngx_conf_t));
    /* STUB: init array ? */
    conf.args = ngx_array_create(pool, 10, sizeof(ngx_str_t));
    if (conf.args == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    conf.temp_pool = ngx_create_pool(NGX_CYCLE_POOL_SIZE, log);
    if (conf.temp_pool == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }


    conf.ctx = cycle->conf_ctx;
    conf.cycle = cycle;
```

```
    conf.pool = pool;
    conf.log = log;
    conf.module_type = NGX_CORE_MODULE;
    conf.cmd_type = NGX_MAIN_CONF;

#if 0
    log->log_level = NGX_LOG_DEBUG_ALL;
#endif

    if (ngx_conf_param(&conf) != NGX_CONF_OK) {
        environ = senv;
        ngx_destroy_cycle_pools(&conf);
        return NULL;
    }

    if (ngx_conf_parse(&conf, &cycle->conf_file) != NGX_CONF_OK) {
        environ = senv;
        ngx_destroy_cycle_pools(&conf);
        return NULL;
    }

    if (ngx_test_config && !ngx_quiet_mode) {
        ngx_log_stderr(0, "the configuration file %s syntax is ok",
                       cycle->conf_file.data);
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_CORE_MODULE) {
            continue;
        }

        module = ngx_modules[i]->ctx;

        if (module->init_conf) {
            if (module->init_conf(cycle, cycle->conf_ctx[ngx_modules[i]->index])
                == NGX_CONF_ERROR)
            {
                environ = senv;
                ngx_destroy_cycle_pools(&conf);
                return NULL;
            }
        }
    }

    if (ngx_process == NGX_PROCESS_SIGNALLER) {
        return cycle;
    }

    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);

    if (ngx_test_config) {

        if (ngx_create_pidfile(&ccf->pid, log) != NGX_OK) {
            goto failed;
        }

    } else if (!ngx_is_init_cycle(old_cycle)) {

        /*
         * we do not create the pid file in the first ngx_init_cycle() call
         * because we need to write the demonized process pid
         */

        old_ccf = (ngx_core_conf_t *) ngx_get_conf(old_cycle->conf_ctx,
                                                   ngx_core_module);
        if (ccf->pid.len != old_ccf->pid.len
            || ngx_strcmp(ccf->pid.data, old_ccf->pid.data) != 0)
        {
            /* new pid file name */

            if (ngx_create_pidfile(&ccf->pid, log) != NGX_OK) {
                goto failed;
            }

            ngx_delete_pidfile(old_cycle);
        }
```

```
    }


    if (ngx_test_lockfile(cycle->lock_file.data, log) != NGX_OK) {
        goto failed;
    }


    if (ngx_create_paths(cycle, ccf->user) != NGX_OK) {
        goto failed;
    }


    if (ngx_log_open_default(cycle) != NGX_OK) {
        goto failed;
    }

    /* open the new files */

    part = &cycle->open_files.part;
    file = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }
            part = part->next;
            file = part->elts;
            i = 0;
        }

        if (file[i].name.len == 0) {
            continue;
        }

        file[i].fd = ngx_open_file(file[i].name.data,
                                   NGX_FILE_APPEND,
                                   NGX_FILE_CREATE_OR_OPEN,
                                   NGX_FILE_DEFAULT_ACCESS);

        ngx_log_debug3(NGX_LOG_DEBUG_CORE, log, 0,
                       "log: %p %d \"%s\"",
                       &file[i], file[i].fd, file[i].name.data);

        if (file[i].fd == NGX_INVALID_FILE) {
            ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
                          ngx_open_file_n " \"%s\" failed",
                          file[i].name.data);
            goto failed;
        }

#if !(NGX_WIN32)
        if (fcntl(file[i].fd, F_SETFD, FD_CLOEXEC) == -1) {
            ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
                          "fcntl(FD_CLOEXEC) \"%s\" failed",
                          file[i].name.data);
            goto failed;
        }
#endif
    }

    cycle->log = &cycle->new_log;
    pool->log = &cycle->new_log;


    /* create shared memory */

    part = &cycle->shared_memory.part;
    shm_zone = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
```

```
                break;
            }
            part = part->next;
            shm_zone = part->elts;
            i = 0;
        }

        if (shm_zone[i].shm.size == 0) {
            ngx_log_error(NGX_LOG_EMERG, log, 0,
                          "zero size shared memory zone \"%V\"",
                          &shm_zone[i].shm.name);
            goto failed;
        }

        shm_zone[i].shm.log = cycle->log;

        opart = &old_cycle->shared_memory.part;
        oshm_zone = opart->elts;

        for (n = 0; /* void */ ; n++) {

            if (n >= opart->nelts) {
                if (opart->next == NULL) {
                    break;
                }
                opart = opart->next;
                oshm_zone = opart->elts;
                n = 0;
            }

            if (shm_zone[i].shm.name.len != oshm_zone[n].shm.name.len) {
                continue;
            }

            if (ngx_strncmp(shm_zone[i].shm.name.data,
                            oshm_zone[n].shm.name.data,
                            shm_zone[i].shm.name.len)
                != 0)
            {
                continue;
            }

            if (shm_zone[i].tag == oshm_zone[n].tag
                && shm_zone[i].shm.size == oshm_zone[n].shm.size)
            {
                shm_zone[i].shm.addr = oshm_zone[n].shm.addr;

                if (shm_zone[i].init(&shm_zone[i], oshm_zone[n].data)
                    != NGX_OK)
                {
                    goto failed;
                }

                goto shm_zone_found;
            }

            ngx_shm_free(&oshm_zone[n].shm);

            break;
        }

        if (ngx_shm_alloc(&shm_zone[i].shm) != NGX_OK) {
            goto failed;
        }

        if (ngx_init_zone_pool(cycle, &shm_zone[i]) != NGX_OK) {
            goto failed;
        }

        if (shm_zone[i].init(&shm_zone[i], NULL) != NGX_OK) {
            goto failed;
        }

    shm_zone_found:

        continue;
```

```
    }


    /* handle the listening sockets */

    if (old_cycle->listening.nelts) {
        ls = old_cycle->listening.elts;
        for (i = 0; i < old_cycle->listening.nelts; i++) {
            ls[i].remain = 0;
        }

        nls = cycle->listening.elts;
        for (n = 0; n < cycle->listening.nelts; n++) {

            for (i = 0; i < old_cycle->listening.nelts; i++) {
                if (ls[i].ignore) {
                    continue;
                }

                if (ngx_cmp_sockaddr(nls[n].sockaddr, nls[n].socklen,
                                     ls[i].sockaddr, ls[i].socklen, 1)
                    == NGX_OK)
                {
                    nls[n].fd = ls[i].fd;
                    nls[n].previous = &ls[i];
                    ls[i].remain = 1;

                    if (ls[i].backlog != nls[n].backlog) {
                        nls[n].listen = 1;
                    }

#if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)

                    /*
                     * FreeBSD, except the most recent versions,
                     * could not remove accept filter
                     */
                    nls[n].deferred_accept = ls[i].deferred_accept;

                    if (ls[i].accept_filter && nls[n].accept_filter) {
                        if (ngx_strcmp(ls[i].accept_filter,
                                       nls[n].accept_filter)
                            != 0)
                        {
                            nls[n].delete_deferred = 1;
                            nls[n].add_deferred = 1;
                        }

                    } else if (ls[i].accept_filter) {
                        nls[n].delete_deferred = 1;

                    } else if (nls[n].accept_filter) {
                        nls[n].add_deferred = 1;
                    }
#endif

#if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)

                    if (ls[i].deferred_accept && !nls[n].deferred_accept) {
                        nls[n].delete_deferred = 1;

                    } else if (ls[i].deferred_accept != nls[n].deferred_accept)
                    {
                        nls[n].add_deferred = 1;
                    }
#endif
                    break;
                }
            }

            if (nls[n].fd == (ngx_socket_t) -1) {
                nls[n].open = 1;
#if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
                if (nls[n].accept_filter) {
                    nls[n].add_deferred = 1;
                }
```

```
#endif
#if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
                if (nls[n].deferred_accept) {
                    nls[n].add_deferred = 1;
                }
#endif
            }
        }

    } else {
        ls = cycle->listening.elts;
        for (i = 0; i < cycle->listening.nelts; i++) {
            ls[i].open = 1;
#if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
            if (ls[i].accept_filter) {
                ls[i].add_deferred = 1;
            }
#endif
#if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
            if (ls[i].deferred_accept) {
                ls[i].add_deferred = 1;
            }
#endif
        }
    }

    if (ngx_open_listening_sockets(cycle) != NGX_OK) {
        goto failed;
    }

    if (!ngx_test_config) {
        ngx_configure_listening_sockets(cycle);
    }


    /* commit the new cycle configuration */

    if (!ngx_use_stderr) {
        (void) ngx_log_redirect_stderr(cycle);
    }

    pool->log = cycle->log;

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->init_module) {
            if (ngx_modules[i]->init_module(cycle) != NGX_OK) {
                /* fatal */
                exit(1);
            }
        }
    }


    /* close and delete stuff that lefts from an old cycle */

    /* free the unnecessary shared memory */

    opart = &old_cycle->shared_memory.part;
    oshm_zone = opart->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= opart->nelts) {
            if (opart->next == NULL) {
                goto old_shm_zone_done;
            }
            opart = opart->next;
            oshm_zone = opart->elts;
            i = 0;
        }

        part = &cycle->shared_memory.part;
        shm_zone = part->elts;

        for (n = 0; /* void */ ; n++) {

```

```
            if (n >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }
                part = part->next;
                shm_zone = part->elts;
                n = 0;
            }

            if (oshm_zone[i].shm.name.len == shm_zone[n].shm.name.len
                && ngx_strncmp(oshm_zone[i].shm.name.data,
                               shm_zone[n].shm.name.data,
                               oshm_zone[i].shm.name.len)
                == 0)
            {
                goto live_shm_zone;
            }
        }

        ngx_shm_free(&oshm_zone[i].shm);

    live_shm_zone:

        continue;
    }

old_shm_zone_done:


    /* close the unnecessary listening sockets */

    ls = old_cycle->listening.elts;
    for (i = 0; i < old_cycle->listening.nelts; i++) {

        if (ls[i].remain || ls[i].fd == (ngx_socket_t) -1) {
            continue;
        }

        if (ngx_close_socket(ls[i].fd) == -1) {
            ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
                          ngx_close_socket_n " listening socket on %V failed",
                          &ls[i].addr_text);
        }

#if (NGX_HAVE_UNIX_DOMAIN)

        if (ls[i].sockaddr->sa_family == AF_UNIX) {
            u_char  *name;

            name = ls[i].addr_text.data + sizeof("unix:") - 1;

            ngx_log_error(NGX_LOG_WARN, cycle->log, 0,
                          "deleting socket %s", name);

            if (ngx_delete_file(name) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_socket_errno,
                              ngx_delete_file_n " %s failed", name);
            }
        }

#endif
    }


    /* close the unnecessary open files */

    part = &old_cycle->open_files.part;
    file = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }
            part = part->next;
```

```
            file = part->elts;
            i = 0;
        }

        if (file[i].fd == NGX_INVALID_FILE || file[i].fd == ngx_stderr) {
            continue;
        }

        if (ngx_close_file(file[i].fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
                          ngx_close_file_n " \"%s\" failed",
                          file[i].name.data);
        }
    }

    ngx_destroy_pool(conf.temp_pool);

    if (ngx_process == NGX_PROCESS_MASTER || ngx_is_init_cycle(old_cycle)) {

        /*
         * perl_destruct() frees environ, if it is not the same as it was at
         * perl_construct() time, therefore we save the previous cycle
         * environment before ngx_conf_parse() where it will be changed.
         */

        env = environ;
        environ = senv;

        ngx_destroy_pool(old_cycle->pool);
        cycle->old_cycle = NULL;

        environ = env;

        return cycle;
    }


    if (ngx_temp_pool == NULL) {
        ngx_temp_pool = ngx_create_pool(128, cycle->log);
        if (ngx_temp_pool == NULL) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
                          "could not create ngx_temp_pool");
            exit(1);
        }

        n = 10;
        ngx_old_cycles.elts = ngx_pcalloc(ngx_temp_pool,
                                          n * sizeof(ngx_cycle_t *));
        if (ngx_old_cycles.elts == NULL) {
            exit(1);
        }
        ngx_old_cycles.nelts = 0;
        ngx_old_cycles.size = sizeof(ngx_cycle_t *);
        ngx_old_cycles.nalloc = n;
        ngx_old_cycles.pool = ngx_temp_pool;

        ngx_cleaner_event.handler = ngx_clean_old_cycles;
        ngx_cleaner_event.log = cycle->log;
        ngx_cleaner_event.data = &dumb;
        dumb.fd = (ngx_socket_t) -1;
    }

    ngx_temp_pool->log = cycle->log;

    old = ngx_array_push(&ngx_old_cycles);
    if (old == NULL) {
        exit(1);
    }
    *old = old_cycle;

    if (!ngx_cleaner_event.timer_set) {
        ngx_add_timer(&ngx_cleaner_event, 30000);
        ngx_cleaner_event.timer_set = 1;
    }

    return cycle;
```

```
781
782
783 failed:
784
785     if (!ngx_is_init_cycle(old_cycle)) {
786         old_ccf = (ngx_core_conf_t *) ngx_get_conf(old_cycle->conf_ctx,
787                                                    ngx_core_module);
788         if (old_ccf->environment) {
789             environ = old_ccf->environment;
790         }
791     }
792
793     /* rollback the new cycle configuration */
794
795     part = &cycle->open_files.part;
796     file = part->elts;
797
798     for (i = 0; /* void */ ; i++) {
799
800         if (i >= part->nelts) {
801             if (part->next == NULL) {
802                 break;
803             }
804             part = part->next;
805             file = part->elts;
806             i = 0;
807         }
808
809         if (file[i].fd == NGX_INVALID_FILE || file[i].fd == ngx_stderr) {
810             continue;
811         }
812
813         if (ngx_close_file(file[i].fd) == NGX_FILE_ERROR) {
814             ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
815                           ngx_close_file_n " \"%s\" failed",
816                           file[i].name.data);
817         }
818     }
819
820     if (ngx_test_config) {
821         ngx_destroy_cycle_pools(&conf);
822         return NULL;
823     }
824
825     ls = cycle->listening.elts;
826     for (i = 0; i < cycle->listening.nelts; i++) {
827         if (ls[i].fd == (ngx_socket_t) -1 || !ls[i].open) {
828             continue;
829         }
830
831         if (ngx_close_socket(ls[i].fd) == -1) {
832             ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
833                           ngx_close_socket_n " %V failed",
834                           &ls[i].addr_text);
835         }
836     }
837
838     ngx_destroy_cycle_pools(&conf);
839
840     return NULL;
841 }
842
843
844 static void
845 ngx_destroy_cycle_pools(ngx_conf_t *conf)
846 {
847     ngx_destroy_pool(conf->temp_pool);
848     ngx_destroy_pool(conf->pool);
849 }
850
851
852 static ngx_int_t
853 ngx_init_zone_pool(ngx_cycle_t *cycle, ngx_shm_zone_t *zn)
854 {
855     u_char           *file;
856     ngx_slab_pool_t  *sp;
```

```
857
858     sp = (ngx_slab_pool_t *) zn->shm.addr;
859
860     if (zn->shm.exists) {
861
862         if (sp == sp->addr) {
863             return NGX_OK;
864         }
865
866         ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
867                       "shared zone \"%V\" has no equal addresses: %p vs %p",
868                       &zn->shm.name, sp->addr, sp);
869         return NGX_ERROR;
870     }
871
872     sp->end = zn->shm.addr + zn->shm.size;
873     sp->min_shift = 3;
874     sp->addr = zn->shm.addr;
875
876 #if (NGX_HAVE_ATOMIC_OPS)
877
878     file = NULL;
879
880 #else
881
882     file = ngx_pnalloc(cycle->pool, cycle->lock_file.len + zn->shm.name.len);
883     if (file == NULL) {
884         return NGX_ERROR;
885     }
886
887     (void) ngx_sprintf(file, "%V%V%Z", &cycle->lock_file, &zn->shm.name);
888
889 #endif
890
891     if (ngx_shmtx_create(&sp->mutex, &sp->lock, file) != NGX_OK) {
892         return NGX_ERROR;
893     }
894
895     ngx_slab_init(sp);
896
897     return NGX_OK;
898 }
899
900
901 ngx_int_t
902 ngx_create_pidfile(ngx_str_t *name, ngx_log_t *log)
903 {
904     size_t      len;
905     ngx_uint_t  create;
906     ngx_file_t  file;
907     u_char      pid[NGX_INT64_LEN + 2];
908
909     if (ngx_process > NGX_PROCESS_MASTER) {
910         return NGX_OK;
911     }
912
913     ngx_memzero(&file, sizeof(ngx_file_t));
914
915     file.name = *name;
916     file.log = log;
917
918     create = ngx_test_config ? NGX_FILE_CREATE_OR_OPEN : NGX_FILE_TRUNCATE;
919
920     file.fd = ngx_open_file(file.name.data, NGX_FILE_RDWR,
921                             create, NGX_FILE_DEFAULT_ACCESS);
922
923     if (file.fd == NGX_INVALID_FILE) {
924         ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
925                       ngx_open_file_n " \"%s\" failed", file.name.data);
926         return NGX_ERROR;
927     }
928
929     if (!ngx_test_config) {
930         len = ngx_snprintf(pid, NGX_INT64_LEN + 2, "%P%N", ngx_pid) - pid;
931
932         if (ngx_write_file(&file, pid, len, 0) == NGX_ERROR) {
```

```
            return NGX_ERROR;
        }
    }

    if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", file.name.data);
    }

    return NGX_OK;
}


void
ngx_delete_pidfile(ngx_cycle_t *cycle)
{
    u_char           *name;
    ngx_core_conf_t  *ccf;

    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);

    name = ngx_new_binary ? ccf->oldpid.data : ccf->pid.data;

    if (ngx_delete_file(name) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      ngx_delete_file_n " \"%s\" failed", name);
    }
}


ngx_int_t
ngx_signal_process(ngx_cycle_t *cycle, char *sig)
{
    ssize_t           n;
    ngx_int_t         pid;
    ngx_file_t        file;
    ngx_core_conf_t  *ccf;
    u_char            buf[NGX_INT64_LEN + 2];

    ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "signal process started");

    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);

    ngx_memzero(&file, sizeof(ngx_file_t));

    file.name = ccf->pid;
    file.log = cycle->log;

    file.fd = ngx_open_file(file.name.data, NGX_FILE_RDONLY,
                            NGX_FILE_OPEN, NGX_FILE_DEFAULT_ACCESS);

    if (file.fd == NGX_INVALID_FILE) {
        ngx_log_error(NGX_LOG_ERR, cycle->log, ngx_errno,
                      ngx_open_file_n " \"%s\" failed", file.name.data);
        return 1;
    }

    n = ngx_read_file(&file, buf, NGX_INT64_LEN + 2, 0);

    if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", file.name.data);
    }

    if (n == NGX_ERROR) {
        return 1;
    }

     while (n-- && (buf[n] == CR || buf[n] == LF)) { /* void */ }

     pid = ngx_atoi(buf, ++n);

     if (pid == NGX_ERROR) {
         ngx_log_error(NGX_LOG_ERR, cycle->log, 0,
                       "invalid PID number \"%*s\" in \"%s\"",
                       n, buf, file.name.data);
```

```
        return 1;
    }

    return ngx_os_signal_process(cycle, sig, pid);

}


static ngx_int_t
ngx_test_lockfile(u_char *file, ngx_log_t *log)
{
#if !(NGX_HAVE_ATOMIC_OPS)
    ngx_fd_t  fd;

    fd = ngx_open_file(file, NGX_FILE_RDWR, NGX_FILE_CREATE_OR_OPEN,
                       NGX_FILE_DEFAULT_ACCESS);

    if (fd == NGX_INVALID_FILE) {
        ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
                      ngx_open_file_n " \"%s\" failed", file);
        return NGX_ERROR;
    }

    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", file);
    }

    if (ngx_delete_file(file) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      ngx_delete_file_n " \"%s\" failed", file);
    }

#endif

    return NGX_OK;
}


void
ngx_reopen_files(ngx_cycle_t *cycle, ngx_uid_t user)
{
    ngx_fd_t          fd;
    ngx_uint_t        i;
    ngx_list_part_t  *part;
    ngx_open_file_t  *file;

    part = &cycle->open_files.part;
    file = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }
            part = part->next;
            file = part->elts;
            i = 0;
        }

        if (file[i].name.len == 0) {
            continue;
        }

        if (file[i].flush) {
            file[i].flush(&file[i], cycle->log);
        }

        fd = ngx_open_file(file[i].name.data, NGX_FILE_APPEND,
                           NGX_FILE_CREATE_OR_OPEN, NGX_FILE_DEFAULT_ACCESS);

        ngx_log_debug3(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "reopen file \"%s\", old:%d new:%d",
                       file[i].name.data, file[i].fd, fd);

```

```
        if (fd == NGX_INVALID_FILE) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          ngx_open_file_n " \"%s\" failed", file[i].name.data);
            continue;
        }

#if !(NGX_WIN32)
        if (user != (ngx_uid_t) NGX_CONF_UNSET_UINT) {
            ngx_file_info_t  fi;

            if (ngx_file_info((const char *) file[i].name.data, &fi)
                == NGX_FILE_ERROR)
            {
                ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                              ngx_file_info_n " \"%s\" failed",
                              file[i].name.data);

                if (ngx_close_file(fd) == NGX_FILE_ERROR) {
                    ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                                  ngx_close_file_n " \"%s\" failed",
                                  file[i].name.data);
                }

                continue;
            }

            if (fi.st_uid != user) {
                if (chown((const char *) file[i].name.data, user, -1) == -1) {
                    ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                                  "chown(\"%s\", %d) failed",
                                  file[i].name.data, user);

                    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
                        ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                                      ngx_close_file_n " \"%s\" failed",
                                      file[i].name.data);
                    }

                    continue;
                }
            }

            if ((fi.st_mode & (S_IRUSR|S_IWUSR)) != (S_IRUSR|S_IWUSR)) {

                fi.st_mode |= (S_IRUSR|S_IWUSR);

                if (chmod((const char *) file[i].name.data, fi.st_mode) == -1) {
                    ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                                  "chmod() \"%s\" failed", file[i].name.data);

                    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
                        ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                                      ngx_close_file_n " \"%s\" failed",
                                      file[i].name.data);
                    }

                    continue;
                }
            }
        }

        if (fcntl(fd, F_SETFD, FD_CLOEXEC) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "fcntl(FD_CLOEXEC) \"%s\" failed",
                          file[i].name.data);

            if (ngx_close_file(fd) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                              ngx_close_file_n " \"%s\" failed",
                              file[i].name.data);
            }

            continue;
        }
#endif

```

```
        if (ngx_close_file(file[i].fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          ngx_close_file_n " \"%s\" failed",
                          file[i].name.data);
        }

        file[i].fd = fd;
    }

    (void) ngx_log_redirect_stderr(cycle);
}


ngx_shm_zone_t *
ngx_shared_memory_add(ngx_conf_t *cf, ngx_str_t *name, size_t size, void *tag)
{
    ngx_uint_t        i;
    ngx_shm_zone_t   *shm_zone;
    ngx_list_part_t  *part;

    part = &cf->cycle->shared_memory.part;
    shm_zone = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }
            part = part->next;
            shm_zone = part->elts;
            i = 0;
        }

        if (name->len != shm_zone[i].shm.name.len) {
            continue;
        }

        if (ngx_strncmp(name->data, shm_zone[i].shm.name.data, name->len)
            != 0)
        {
            continue;
        }

        if (tag != shm_zone[i].tag) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "the shared memory zone \"%V\" is "
                               "already declared for a different use",
                               &shm_zone[i].shm.name);
            return NULL;
        }

        if (size && size != shm_zone[i].shm.size) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "the size %uz of shared memory zone \"%V\" "
                               "conflicts with already declared size %uz",
                               size, &shm_zone[i].shm.name, shm_zone[i].shm.size);
            return NULL;
        }

        return &shm_zone[i];
    }

    shm_zone = ngx_list_push(&cf->cycle->shared_memory);

    if (shm_zone == NULL) {
        return NULL;
    }

    shm_zone->data = NULL;
    shm_zone->shm.log = cf->cycle->log;
    shm_zone->shm.size = size;
    shm_zone->shm.name = *name;
    shm_zone->shm.exists = 0;
    shm_zone->init = NULL;
    shm_zone->tag = tag;
```

```
1237
1238        return shm_zone;
1239    }
1240
1241
1242    static void
1243    ngx_clean_old_cycles(ngx_event_t *ev)
1244    {
1245        ngx_uint_t     i, n, found, live;
1246        ngx_log_t     *log;
1247        ngx_cycle_t  **cycle;
1248
1249        log = ngx_cycle->log;
1250        ngx_temp_pool->log = log;
1251
1252        ngx_log_debug0(NGX_LOG_DEBUG_CORE, log, 0, "clean old cycles");
1253
1254        live = 0;
1255
1256        cycle = ngx_old_cycles.elts;
1257        for (i = 0; i < ngx_old_cycles.nelts; i++) {
1258
1259            if (cycle[i] == NULL) {
1260                continue;
1261            }
1262
1263            found = 0;
1264
1265            for (n = 0; n < cycle[i]->connection_n; n++) {
1266                if (cycle[i]->connections[n].fd != (ngx_socket_t) -1) {
1267                    found = 1;
1268
1269                    ngx_log_debug1(NGX_LOG_DEBUG_CORE, log, 0, "live fd:%d", n);
1270
1271                    break;
1272                }
1273            }
1274
1275            if (found) {
1276                live = 1;
1277                continue;
1278            }
1279
1280            ngx_log_debug1(NGX_LOG_DEBUG_CORE, log, 0, "clean old cycle: %d", i);
1281
1282            ngx_destroy_pool(cycle[i]->pool);
1283            cycle[i] = NULL;
1284        }
1285
1286        ngx_log_debug1(NGX_LOG_DEBUG_CORE, log, 0, "old cycles status: %d", live);
1287
1288        if (live) {
1289            ngx_add_timer(ev, 30000);
1290
1291        } else {
1292            ngx_destroy_pool(ngx_temp_pool);
1293            ngx_temp_pool = NULL;
1294            ngx_old_cycles.nelts = 0;
1295        }
1296    }
```

One Level Up Top Level

src/core/ngx_cycle.h - nginx-1.7.10

Data types defined

- ngx_core_conf_t
- ngx_core_tls_t
- ngx_cycle_s
- ngx_shm_zone_init_pt
- ngx_shm_zone_s
- ngx_shm_zone_t

Macros defined

- NGX_CYCLE_POOL_SIZE
- NGX_DEBUG_POINTS_ABORT
- NGX_DEBUG_POINTS_STOP
- _NGX_CYCLE_H_INCLUDED_
- ngx_is_init_cycle

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_CYCLE_H_INCLUDED_
9  #define _NGX_CYCLE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #ifndef NGX_CYCLE_POOL_SIZE
17  #define NGX_CYCLE_POOL_SIZE     NGX_DEFAULT_POOL_SIZE
18  #endif
19
20
21  #define NGX_DEBUG_POINTS_STOP   1
22  #define NGX_DEBUG_POINTS_ABORT  2
23
24
25  typedef struct ngx_shm_zone_s  ngx_shm_zone_t;
26
27  typedef ngx_int_t (*ngx_shm_zone_init_pt) (ngx_shm_zone_t *zone, void *data);
28
29  struct ngx_shm_zone_s {
30      void                     *data;
31      ngx_shm_t                 shm;
32      ngx_shm_zone_init_pt      init;
33      void                     *tag;
34  };
35
36
```

```
struct ngx_cycle_s {
    void                  ****conf_ctx;
    ngx_pool_t              *pool;

    ngx_log_t               *log;
    ngx_log_t                new_log;

    ngx_uint_t               log_use_stderr;  /* unsigned  log_use_stderr:1; */

    ngx_connection_t        **files;
    ngx_connection_t         *free_connections;
    ngx_uint_t               free_connection_n;

    ngx_queue_t              reusable_connections_queue;

    ngx_array_t              listening;
    ngx_array_t              paths;
    ngx_list_t               open_files;
    ngx_list_t               shared_memory;

    ngx_uint_t               connection_n;
    ngx_uint_t               files_n;

    ngx_connection_t        *connections;
    ngx_event_t             *read_events;
    ngx_event_t             *write_events;

    ngx_cycle_t             *old_cycle;

    ngx_str_t                conf_file;
    ngx_str_t                conf_param;
    ngx_str_t                conf_prefix;
    ngx_str_t                prefix;
    ngx_str_t                lock_file;
    ngx_str_t                hostname;
};


typedef struct {
     ngx_flag_t              daemon;
     ngx_flag_t              master;

     ngx_msec_t              timer_resolution;

     ngx_int_t               worker_processes;
     ngx_int_t               debug_points;

     ngx_int_t               rlimit_nofile;
     ngx_int_t               rlimit_sigpending;
     off_t                   rlimit_core;

     int                     priority;

     ngx_uint_t              cpu_affinity_n;
     uint64_t               *cpu_affinity;

     char                   *username;
     ngx_uid_t               user;
     ngx_gid_t               group;

     ngx_str_t               working_directory;
     ngx_str_t               lock_file;

      ngx_str_t               pid;
      ngx_str_t               oldpid;

      ngx_array_t             env;
      char                  **environment;

#if (NGX_THREADS)
     ngx_int_t               worker_threads;
     size_t                  thread_stack_size;
#endif

} ngx_core_conf_t;

```

```
113
114 typedef struct {
115     ngx_pool_t              *pool;   /* pcre's malloc() pool */
116 } ngx_core_tls_t;
117
118
119 #define ngx_is_init_cycle(cycle)  (cycle->conf_ctx == NULL)
120
121
122 ngx_cycle_t *ngx_init_cycle(ngx_cycle_t *old_cycle);
123 ngx_int_t ngx_create_pidfile(ngx_str_t *name, ngx_log_t *log);
124 void ngx_delete_pidfile(ngx_cycle_t *cycle);
125 ngx_int_t ngx_signal_process(ngx_cycle_t *cycle, char *sig);
126 void ngx_reopen_files(ngx_cycle_t *cycle, ngx_uid_t user);
127 char **ngx_set_environment(ngx_cycle_t *cycle, ngx_uint_t *last);
128 ngx_pid_t ngx_exec_new_binary(ngx_cycle_t *cycle, char *const *argv);
129 uint64_t ngx_get_cpu_affinity(ngx_uint_t n);
130 ngx_shm_zone_t *ngx_shared_memory_add(ngx_conf_t *cf, ngx_str_t *name,
131     size_t size, void *tag);
132
133
134 extern volatile ngx_cycle_t  *ngx_cycle;
135 extern ngx_array_t            ngx_old_cycles;
136 extern ngx_module_t           ngx_core_module;
137 extern ngx_uint_t             ngx_test_config;
138 extern ngx_uint_t             ngx_quiet_mode;
139 #if (NGX_THREADS)
140 extern ngx_tls_key_t          ngx_core_tls_key;
141 #endif
142
143
144 #endif /* _NGX_CYCLE_H_INCLUDED_ */
```

One Level Up Top Level

One Level Up Top Level

src/os/unix/ngx_shmem.h - nginx-1.7.10

Data types defined

- ngx_shm_t

Macros defined

- _NGX_SHMEM_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_SHMEM_H_INCLUDED_
9  #define _NGX_SHMEM_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct {
17      u_char      *addr;
18      size_t       size;
19      ngx_str_t    name;
20      ngx_log_t   *log;
21      ngx_uint_t   exists;   /* unsigned  exists:1;  */
22  } ngx_shm_t;
23
24
25  ngx_int_t ngx_shm_alloc(ngx_shm_t *shm);
26  void ngx_shm_free(ngx_shm_t *shm);
27
28
29  #endif /* _NGX_SHMEM_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_shmem.c - nginx-1.7.10

Functions defined

- ngx_shm_alloc
- ngx_shm_alloc
- ngx_shm_alloc
- ngx_shm_free
- ngx_shm_free
- ngx_shm_free

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 #if (NGX_HAVE_MAP_ANON)
13
14 ngx_int_t
15 ngx_shm_alloc(ngx_shm_t *shm)
16 {
17     shm->addr = (u_char *) mmap(NULL, shm->size,
18                                 PROT_READ|PROT_WRITE,
19                                 MAP_ANON|MAP_SHARED, -1, 0);
20
21     if (shm->addr == MAP_FAILED) {
22         ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
23                       "mmap(MAP_ANON|MAP_SHARED, %uz) failed", shm->size);
24         return NGX_ERROR;
25     }
26
27     return NGX_OK;
28 }
29
30
31 void
32 ngx_shm_free(ngx_shm_t *shm)
33 {
34     if (munmap((void *) shm->addr, shm->size) == -1) {
35         ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
36                       "munmap(%p, %uz) failed", shm->addr, shm->size);
37     }
38 }
39
40 #elif (NGX_HAVE_MAP_DEVZERO)
41
42 ngx_int_t
43 ngx_shm_alloc(ngx_shm_t *shm)
44 {
45     ngx_fd_t  fd;
46
47     fd = open("/dev/zero", O_RDWR);
48
49     if (fd == -1) {
50         ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
51                       "open(\"/dev/zero\") failed");
```

```
        return NGX_ERROR;
    }

    shm->addr = (u_char *) mmap(NULL, shm->size, PROT_READ|PROT_WRITE,
                               MAP_SHARED, fd, 0);

    if (shm->addr == MAP_FAILED) {
        ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
                      "mmap(/dev/zero, MAP_SHARED, %uz) failed", shm->size);
    }

    if (close(fd) == -1) {
        ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
                      "close(\"/dev/zero\") failed");
    }

    return (shm->addr == MAP_FAILED) ? NGX_ERROR : NGX_OK;
}


void
ngx_shm_free(ngx_shm_t *shm)
{
    if (munmap((void *) shm->addr, shm->size) == -1) {
        ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
                      "munmap(%p, %uz) failed", shm->addr, shm->size);
    }
}

#elif (NGX_HAVE_SYSVSHM)

#include <sys/ipc.h>
#include <sys/shm.h>


ngx_int_t
ngx_shm_alloc(ngx_shm_t *shm)
{
    int  id;

    id = shmget(IPC_PRIVATE, shm->size, (SHM_R|SHM_W|IPC_CREAT));

    if (id == -1) {
        ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
                      "shmget(%uz) failed", shm->size);
        return NGX_ERROR;
    }

     ngx_log_debug1(NGX_LOG_DEBUG_CORE, shm->log, 0, "shmget id: %d", id);

     shm->addr = shmat(id, NULL, 0);

     if (shm->addr == (void *) -1) {
         ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno, "shmat() failed");
     }

     if (shmctl(id, IPC_RMID, NULL) == -1) {
         ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
                       "shmctl(IPC_RMID) failed");
     }

     return (shm->addr == (void *) -1) ? NGX_ERROR : NGX_OK;
 }


 void
 ngx_shm_free(ngx_shm_t *shm)
 {
     if (shmdt(shm->addr) == -1) {
         ngx_log_error(NGX_LOG_ALERT, shm->log, ngx_errno,
                       "shmdt(%p) failed", shm->addr);
     }
 }

 #endif
```

One Level Up Top Level

src/os/unix/ngx_user.h - nginx-1.7.10

Data types defined

- ngx_gid_t
- ngx_uid_t

Macros defined

- _NGX_USER_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_USER_H_INCLUDED_
9  #define _NGX_USER_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef uid_t  ngx_uid_t;
17  typedef gid_t  ngx_gid_t;
18
19
20  ngx_int_t ngx_libc_crypt(ngx_pool_t *pool, u_char *key, u_char *salt,
21      u_char **encrypted);
22
23
24  #endif /* _NGX_USER_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_user.c - nginx-1.7.10

Functions defined

- ngx_libc_crypt
- ngx_libc_crypt

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13  * Solaris has thread-safe crypt()
14  * Linux has crypt_r(); "struct crypt_data" is more than 128K
15  * FreeBSD needs the mutex to protect crypt()
16  *
17  * TODO:
18  *     ngx_crypt_init() to init mutex
19  */
20
21
22  #if (NGX_CRYPT)
23
24  #if (NGX_HAVE_GNU_CRYPT_R)
25
26  ngx_int_t
27  ngx_libc_crypt(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
28  {
29      char               *value;
30      size_t              len;
31      struct crypt_data   cd;
32
33      cd.initialized = 0;
34  #ifdef __GLIBC__
35      /* work around the glibc bug */
36      cd.current_salt[0] = ~salt[0];
37  #endif
38
39      value = crypt_r((char *) key, (char *) salt, &cd);
40
41      if (value) {
42          len = ngx_strlen(value) + 1;
43
44          *encrypted = ngx_pnalloc(pool, len);
45          if (*encrypted == NULL) {
46              return NGX_ERROR;
47          }
48
49          ngx_memcpy(*encrypted, value, len);
50          return NGX_OK;
51      }
52
53      ngx_log_error(NGX_LOG_CRIT, pool->log, ngx_errno, "crypt_r() failed");
54
55      return NGX_ERROR;
56  }
57
58  #else
59
60  ngx_int_t
```

```
ngx_libc_crypt(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
{
    char       *value;
    size_t      len;
    ngx_err_t   err;

#if (NGX_THREADS && NGX_NONREENTRANT_CRYPT)

    /* crypt() is a time consuming function, so we only try to lock */

    if (ngx_mutex_trylock(ngx_crypt_mutex) != NGX_OK) {
        return NGX_AGAIN;
    }

#endif

    value = crypt((char *) key, (char *) salt);

    if (value) {
        len = ngx_strlen(value) + 1;

        *encrypted = ngx_pnalloc(pool, len);
        if (*encrypted == NULL) {
#if (NGX_THREADS && NGX_NONREENTRANT_CRYPT)
            ngx_mutex_unlock(ngx_crypt_mutex);
#endif
            return NGX_ERROR;
        }

        ngx_memcpy(*encrypted, value, len);
#if (NGX_THREADS && NGX_NONREENTRANT_CRYPT)
        ngx_mutex_unlock(ngx_crypt_mutex);
#endif
        return NGX_OK;
    }

    err = ngx_errno;

#if (NGX_THREADS && NGX_NONREENTRANT_CRYPT)
     ngx_mutex_unlock(ngx_crypt_mutex);
 #endif

     ngx_log_error(NGX_LOG_CRIT, pool->log, err, "crypt() failed");

     return NGX_ERROR;
 }

 #endif

 #endif /* NGX_CRYPT */
```

One Level Up Top Level

src/os/unix/ngx_pthread_thread.c - nginx-1.7.10

Global variables defined

- max_threads
- nthreads
- thr_attr

Functions defined

- ngx_cond_destroy
- ngx_cond_init
- ngx_cond_signal
- ngx_cond_wait
- ngx_create_thread
- ngx_init_threads
- ngx_mutex_destroy
- ngx_mutex_init
- ngx_mutex_lock
- ngx_mutex_trylock
- ngx_mutex_unlock

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  static ngx_uint_t   nthreads;
13  static ngx_uint_t   max_threads;
14
15
16  static pthread_attr_t  thr_attr;
17
18
19  ngx_err_t
20  ngx_create_thread(ngx_tid_t *tid, ngx_thread_value_t (*func)(void *arg),
21      void *arg, ngx_log_t *log)
22  {
23      int  err;
24
25      if (nthreads >= max_threads) {
26          ngx_log_error(NGX_LOG_CRIT, log, 0,
27                        "no more than %ui threads can be created", max_threads);
28          return NGX_ERROR;
29      }
```

```
30
31      err = pthread_create(tid, &thr_attr, func, arg);
32
33      if (err != 0) {
34          ngx_log_error(NGX_LOG_ALERT, log, err, "pthread_create() failed");
35          return err;
36      }
37
38      ngx_log_debug1(NGX_LOG_DEBUG_CORE, log, 0,
39                     "thread is created: " NGX_TID_T_FMT, *tid);
40
41      nthreads++;
42
43      return err;
44  }
45
46
47  ngx_int_t
48  ngx_init_threads(int n, size_t size, ngx_cycle_t *cycle)
49  {
50      int  err;
51
52      max_threads = n;
53
54      err = pthread_attr_init(&thr_attr);
55
56      if (err != 0) {
57          ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
58                        "pthread_attr_init() failed");
59          return NGX_ERROR;
60      }
61
62      err = pthread_attr_setstacksize(&thr_attr, size);
63
64      if (err != 0) {
65          ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
66                        "pthread_attr_setstacksize() failed");
67          return NGX_ERROR;
68      }
69
70      ngx_threaded = 1;
71
72      return NGX_OK;
73  }
74
75
76  ngx_mutex_t *
77  ngx_mutex_init(ngx_log_t *log, ngx_uint_t flags)
78  {
79      int           err;
80      ngx_mutex_t  *m;
81
82      m = ngx_alloc(sizeof(ngx_mutex_t), log);
83      if (m == NULL) {
84          return NULL;
85      }
86
87      m->log = log;
88
89      err = pthread_mutex_init(&m->mutex, NULL);
90
91      if (err != 0) {
92          ngx_log_error(NGX_LOG_ALERT, m->log, err,
93                        "pthread_mutex_init() failed");
94          return NULL;
95      }
96
97      return m;
98  }
99
100
101  void
102  ngx_mutex_destroy(ngx_mutex_t *m)
103  {
104      int  err;
105
```

```
    err = pthread_mutex_destroy(&m->mutex);

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, m->log, err,
                      "pthread_mutex_destroy(%p) failed", m);
    }

    ngx_free(m);
}


void
ngx_mutex_lock(ngx_mutex_t *m)
{
    int  err;

    if (!ngx_threaded) {
        return;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "lock mutex %p", m);

    err = pthread_mutex_lock(&m->mutex);

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, m->log, err,
                      "pthread_mutex_lock(%p) failed", m);
        ngx_abort();
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "mutex %p is locked", m);

    return;
}


ngx_int_t
ngx_mutex_trylock(ngx_mutex_t *m)
{
    int  err;

    if (!ngx_threaded) {
        return NGX_OK;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "try lock mutex %p", m);

    err = pthread_mutex_trylock(&m->mutex);

    if (err == NGX_EBUSY) {
        return NGX_AGAIN;
    }

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, m->log, err,
                      "pthread_mutex_trylock(%p) failed", m);
        ngx_abort();
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "mutex %p is locked", m);

    return NGX_OK;
}


void
ngx_mutex_unlock(ngx_mutex_t *m)
{
    int  err;

    if (!ngx_threaded) {
        return;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "unlock mutex %p", m);

```

```
    err = pthread_mutex_unlock(&m->mutex);

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, m->log, err,
                      "pthread_mutex_unlock(%p) failed", m);
        ngx_abort();
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "mutex %p is unlocked", m);

    return;
}


ngx_cond_t *
ngx_cond_init(ngx_log_t *log)
{
    int          err;
    ngx_cond_t  *cv;

    cv = ngx_alloc(sizeof(ngx_cond_t), log);
    if (cv == NULL) {
        return NULL;
    }

    cv->log = log;

    err = pthread_cond_init(&cv->cond, NULL);

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, cv->log, err,
                      "pthread_cond_init() failed");
        return NULL;
    }

    return cv;
}


void
ngx_cond_destroy(ngx_cond_t *cv)
{
    int  err;

    err = pthread_cond_destroy(&cv->cond);

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, cv->log, err,
                      "pthread_cond_destroy(%p) failed", cv);
    }

    ngx_free(cv);
}


ngx_int_t
ngx_cond_wait(ngx_cond_t *cv, ngx_mutex_t *m)
{
    int  err;

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, cv->log, 0, "cv %p wait", cv);

    err = pthread_cond_wait(&cv->cond, &m->mutex);

    if (err != 0) {
        ngx_log_error(NGX_LOG_ALERT, cv->log, err,
                      "pthread_cond_wait(%p) failed", cv);
        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, cv->log, 0, "cv %p is waked up", cv);

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0, "mutex %p is locked", m);

    return NGX_OK;
}
```

```
258
259
260 ngx_int_t
261 ngx_cond_signal(ngx_cond_t *cv)
262 {
263     int  err;
264
265     ngx_log_debug1(NGX_LOG_DEBUG_CORE, cv->log, 0, "cv %p to signal", cv);
266
267     err = pthread_cond_signal(&cv->cond);
268
269     if (err != 0) {
270         ngx_log_error(NGX_LOG_ALERT, cv->log, err,
271                       "pthread_cond_signal(%p) failed", cv);
272         return NGX_ERROR;
273     }
274
275     ngx_log_debug1(NGX_LOG_DEBUG_CORE, cv->log, 0, "cv %p is signaled", cv);
276
277     return NGX_OK;
278 }
```

One Level Up Top Level

src/os/unix/ngx_thread.h - nginx-1.7.10

Data types defined

- ngx_cond_t
- ngx_mutex_t
- ngx_thread_t
- ngx_thread_value_t
- ngx_tid_t
- ngx_tls_key_t

Macros defined

- NGX_MAX_THREADS
- NGX_MUTEX_LIGHT
- NGX_THREAD_BUSY
- NGX_THREAD_DONE
- NGX_THREAD_EXIT
- NGX_THREAD_FREE
- NGX_TID_T_FMT
- NGX_TID_T_FMT
- NGX_TID_T_FMT
- _NGX_THREAD_H_INCLUDED_
- ngx_cond_signal
- ngx_log_tid
- ngx_log_tid
- ngx_mutex_lock
- ngx_mutex_trylock
- ngx_mutex_unlock
- ngx_setthrtitle
- ngx_thread_get_tls
- ngx_thread_join
- ngx_thread_key_create
- ngx_thread_key_create_n
- ngx_thread_main

- ngx_thread_self
- ngx_thread_set_tls
- ngx_thread_set_tls_n
- ngx_thread_sigmask
- ngx_thread_sigmask_n
- ngx_thread_volatile
- ngx_thread_volatile

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_THREAD_H_INCLUDED_
9  #define _NGX_THREAD_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15  #if (NGX_THREADS)
16
17  #define NGX_MAX_THREADS      128
18
19  #if (NGX_USE_RFORK)
20  #include <ngx_freebsd_rfork_thread.h>
21
22
23  #else /* use pthreads */
24
25  #include <pthread.h>
26
27  typedef pthread_t                   ngx_tid_t;
28
29  #define ngx_thread_self()           pthread_self()
30  #define ngx_log_tid                 (int) ngx_thread_self()
31
32  #if (NGX_FREEBSD) && !(NGX_LINUXTHREADS)
33  #define NGX_TID_T_FMT               "%p"
34  #else
35  #define NGX_TID_T_FMT               "%d"
36  #endif
37
38
39  typedef pthread_key_t               ngx_tls_key_t;
40
41  #define ngx_thread_key_create(key)   pthread_key_create(key, NULL)
42  #define ngx_thread_key_create_n      "pthread_key_create()"
43  #define ngx_thread_set_tls           pthread_setspecific
44  #define ngx_thread_set_tls_n         "pthread_setspecific()"
45  #define ngx_thread_get_tls           pthread_getspecific
46
47
48  #define NGX_MUTEX_LIGHT     0
49
50  typedef struct {
51      pthread_mutex_t   mutex;
52      ngx_log_t        *log;
53  } ngx_mutex_t;
54
55  typedef struct {
56      pthread_cond_t    cond;
```

```
57      ngx_log_t        *log;
58  } ngx_cond_t;
59
60  #define ngx_thread_sigmask     pthread_sigmask
61  #define ngx_thread_sigmask_n  "pthread_sigmask()"
62
63  #define ngx_thread_join(t, p)  pthread_join(t, p)
64
65  #define ngx_setthrtitle(n)
66
67
68
69  ngx_int_t ngx_mutex_trylock(ngx_mutex_t *m);
70  void ngx_mutex_lock(ngx_mutex_t *m);
71  void ngx_mutex_unlock(ngx_mutex_t *m);
72
73  #endif
74
75
76  #define ngx_thread_volatile   volatile
77
78
79  typedef struct {
80      ngx_tid_t    tid;
81      ngx_cond_t  *cv;
82      ngx_uint_t   state;
83  } ngx_thread_t;
84
85  #define NGX_THREAD_FREE   1
86  #define NGX_THREAD_BUSY   2
87  #define NGX_THREAD_EXIT   3
88  #define NGX_THREAD_DONE   4
89
90  extern ngx_int_t              ngx_threads_n;
91  extern volatile ngx_thread_t  ngx_threads[NGX_MAX_THREADS];
92
93
94  typedef void *  ngx_thread_value_t;
95
96  ngx_int_t ngx_init_threads(int n, size_t size, ngx_cycle_t *cycle);
97  ngx_err_t ngx_create_thread(ngx_tid_t *tid,
98      ngx_thread_value_t (*func)(void *arg), void *arg, ngx_log_t *log);
99
100 ngx_mutex_t *ngx_mutex_init(ngx_log_t *log, ngx_uint_t flags);
101 void ngx_mutex_destroy(ngx_mutex_t *m);
102
103
104 ngx_cond_t *ngx_cond_init(ngx_log_t *log);
105 void ngx_cond_destroy(ngx_cond_t *cv);
106 ngx_int_t ngx_cond_wait(ngx_cond_t *cv, ngx_mutex_t *m);
107 ngx_int_t ngx_cond_signal(ngx_cond_t *cv);
108
109
110 #else /* !NGX_THREADS */
111
112 #define ngx_thread_volatile
113
114 #define ngx_log_tid           0
115 #define NGX_TID_T_FMT         "%d"
116
117 #define ngx_mutex_trylock(m)  NGX_OK
118 #define ngx_mutex_lock(m)
119 #define ngx_mutex_unlock(m)
120
121 #define ngx_cond_signal(cv)
122
123 #define ngx_thread_main()     1
124
125 #endif
126
127
128 #endif /* _NGX_THREAD_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_freebsd_rfork_thread.c - nginx-1.7.10

Global variables defined

- errno0
- errnos
- last_stack
- max_threads
- ngx_freebsd_kern_usrstack
- ngx_thread_stack_size
- ngx_tls
- nkeys
- nthreads
- rz_size
- tids
- usable_stack_size

Functions defined

- __error
- _spinlock
- _spinunlock
- ngx_cond_destroy
- ngx_cond_init
- ngx_cond_signal
- ngx_cond_wait
- ngx_create_thread
- ngx_init_threads
- ngx_mutex_destroy
- ngx_mutex_dolock
- ngx_mutex_init
- ngx_mutex_unlock
- ngx_thread_key_create
- ngx_thread_self
- ngx_thread_set_tls

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11  /*
12   * The threads implementation uses the rfork(RFPROC|RFTHREAD|RFMEM) syscall
13   * to create threads.  All threads use the stacks of the same size mmap()ed
14   * below the main stack.  Thus the current thread id is determined via
15   * the stack pointer value.
16   *
17   * The mutex implementation uses the ngx_atomic_cmp_set() operation
18   * to acquire a mutex and the SysV semaphore to wait on a mutex and to wake up
19   * the waiting threads.  The light mutex does not use semaphore, so after
20   * spinning in the lock the thread calls sched_yield().  However the light
21   * mutexes are intended to be used with the "trylock" operation only.
22   * The SysV semop() is a cheap syscall, particularly if it has little sembuf's
23   * and does not use SEM_UNDO.
24   *
25   * The condition variable implementation uses the signal #64.
26   * The signal handler is SIG_IGN so the kill() is a cheap syscall.
27   * The thread waits a signal in kevent().  The use of the EVFILT_SIGNAL
28   * is safe since FreeBSD 4.10-STABLE.
29   *
30   * This threads implementation currently works on i386 (486+) and amd64
31   * platforms only.
32   */
33
34
35  char                *ngx_freebsd_kern_usrstack;
36  size_t               ngx_thread_stack_size;
37
38
39  static size_t        rz_size;
40  static size_t        usable_stack_size;
41  static char         *last_stack;
42
43  static ngx_uint_t    nthreads;
44  static ngx_uint_t    max_threads;
45
46  static ngx_uint_t    nkeys;
47  static ngx_tid_t    *tids;      /* the threads tids array */
48  void               **ngx_tls;   /* the threads tls's array */
49
50  /* the thread-safe libc errno */
51
52  static int   errno0;   /* the main thread's errno */
53  static int  *errnos;   /* the threads errno's array */
54
55  int *
56  __error()
57  {
58      int  tid;
59
60      tid = ngx_gettid();
61
62      return tid ? &errnos[tid - 1] : &errno0;
63  }
64
65
66  /*
67   * __isthreaded enables the spinlocks in some libc functions, i.e. in malloc()
68   * and some other places.  Nevertheless we protect our malloc()/free() calls
69   * by own mutex that is more efficient than the spinlock.
70   *
71   * _spinlock() is a weak referenced stub in src/lib/libc/gen/_spinlock_stub.c
72   * that does nothing.
73   */
```

```
74
75  extern int  __isthreaded;
76
77  void
78  _spinlock(ngx_atomic_t *lock)
79  {
80      ngx_int_t  tries;
81
82      tries = 0;
83
84      for ( ;; ) {
85
86          if (*lock) {
87              if (ngx_ncpu > 1 && tries++ < 1000) {
88                  continue;
89              }
90
91              sched_yield();
92              tries = 0;
93
94          } else {
95              if (ngx_atomic_cmp_set(lock, 0, 1)) {
96                  return;
97              }
98          }
99      }
100 }
101
102
103 /*
104 * Before FreeBSD 5.1 _spinunlock() is a simple #define in
105 * src/lib/libc/include/spinlock.h that zeroes lock.
106 *
107 * Since FreeBSD 5.1 _spinunlock() is a weak referenced stub in
108 * src/lib/libc/gen/_spinlock_stub.c that does nothing.
109 */
110
111 #ifndef _spinunlock
112
113 void
114 _spinunlock(ngx_atomic_t *lock)
115 {
116     *lock = 0;
117 }
118
119 #endif
120
121
122 ngx_err_t
123 ngx_create_thread(ngx_tid_t *tid, ngx_thread_value_t (*func)(void *arg),
124     void *arg, ngx_log_t *log)
125 {
126     ngx_pid_t   id;
127     ngx_err_t   err;
128     char       *stack, *stack_top;
129
130     if (nthreads >= max_threads) {
131         ngx_log_error(NGX_LOG_CRIT, log, 0,
132                       "no more than %ui threads can be created", max_threads);
133         return NGX_ERROR;
134     }
135
136     last_stack -= ngx_thread_stack_size;
137
138     stack = mmap(last_stack, usable_stack_size, PROT_READ|PROT_WRITE,
139                  MAP_STACK, -1, 0);
140
141     if (stack == MAP_FAILED) {
142         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
143                       "mmap(%p:%uz, MAP_STACK) thread stack failed",
144                       last_stack, usable_stack_size);
145         return NGX_ERROR;
146     }
147
148     if (stack != last_stack) {
149         ngx_log_error(NGX_LOG_ALERT, log, 0,
```

```
                    "stack %p address was changed to %p", last_stack, stack);
        return NGX_ERROR;
    }

    stack_top = stack + usable_stack_size;

    ngx_log_debug2(NGX_LOG_DEBUG_CORE, log, 0,
                   "thread stack: %p-%p", stack, stack_top);

    ngx_set_errno(0);

    id = rfork_thread(RFPROC|RFTHREAD|RFMEM, stack_top,
                      (ngx_rfork_thread_func_pt) func, arg);

    err = ngx_errno;

    if (id == -1) {
        ngx_log_error(NGX_LOG_ALERT, log, err, "rfork() failed");

    } else {
        *tid = id;
        nthreads = (ngx_freebsd_kern_usrstack - stack_top)
                                                / ngx_thread_stack_size;
        tids[nthreads] = id;

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, log, 0, "rfork()ed thread: %P", id);
    }

    return err;
}


ngx_int_t
ngx_init_threads(int n, size_t size, ngx_cycle_t *cycle)
{
    char              *red_zone, *zone;
    size_t             len;
    ngx_int_t          i;
    struct sigaction   sa;

    max_threads = n + 1;

    for (i = 0; i < n; i++) {
        ngx_memzero(&sa, sizeof(struct sigaction));
        sa.sa_handler = SIG_IGN;
        sigemptyset(&sa.sa_mask);
        if (sigaction(NGX_CV_SIGNAL, &sa, NULL) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "sigaction(%d, SIG_IGN) failed", NGX_CV_SIGNAL);
            return NGX_ERROR;
        }
    }

    len = sizeof(ngx_freebsd_kern_usrstack);
    if (sysctlbyname("kern.usrstack", &ngx_freebsd_kern_usrstack, &len,
                                                        NULL, 0) == -1)
    {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "sysctlbyname(kern.usrstack) failed");
        return NGX_ERROR;
    }

    /* the main thread stack red zone */
    rz_size = ngx_pagesize;
    red_zone = ngx_freebsd_kern_usrstack - (size + rz_size);

    ngx_log_debug2(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                   "usrstack: %p red zone: %p",
                   ngx_freebsd_kern_usrstack, red_zone);

    zone = mmap(red_zone, rz_size, PROT_NONE, MAP_ANON, -1, 0);
    if (zone == MAP_FAILED) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "mmap(%p:%uz, PROT_NONE, MAP_ANON) red zone failed",
                      red_zone, rz_size);
        return NGX_ERROR;
```

```
    }

    if (zone != red_zone) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                      "red zone %p address was changed to %p", red_zone, zone);
        return NGX_ERROR;
    }

    /* create the thread errno' array */

    errnos = ngx_calloc(n * sizeof(int), cycle->log);
    if (errnos == NULL) {
        return NGX_ERROR;
    }

    /* create the thread tids array */

    tids = ngx_calloc((n + 1) * sizeof(ngx_tid_t), cycle->log);
    if (tids == NULL) {
        return NGX_ERROR;
    }

    tids[0] = ngx_pid;

    /* create the thread tls' array */

    ngx_tls = ngx_calloc(NGX_THREAD_KEYS_MAX * (n + 1) * sizeof(void *),
                         cycle->log);
    if (ngx_tls == NULL) {
        return NGX_ERROR;
    }

    nthreads = 1;

    last_stack = zone + rz_size;
    usable_stack_size = size;
    ngx_thread_stack_size = size + rz_size;

    /* allow the spinlock in libc malloc() */
    __isthreaded = 1;

    ngx_threaded = 1;

    return NGX_OK;
}


ngx_tid_t
ngx_thread_self(void)
{
    ngx_int_t  tid;

    tid = ngx_gettid();

    if (tids == NULL) {
        return ngx_pid;
    }

    return tids[tid];
}


ngx_err_t
ngx_thread_key_create(ngx_tls_key_t *key)
{
    if (nkeys >= NGX_THREAD_KEYS_MAX) {
        return NGX_ENOMEM;
    }

    *key = nkeys++;

    return 0;
}


ngx_err_t
```

```
ngx_thread_set_tls(ngx_tls_key_t key, void *value)
{
    if (key >= NGX_THREAD_KEYS_MAX) {
        return NGX_EINVAL;
    }

    ngx_tls[key * NGX_THREAD_KEYS_MAX + ngx_gettid()] = value;
    return 0;
}


ngx_mutex_t *
ngx_mutex_init(ngx_log_t *log, ngx_uint_t flags)
{
    ngx_mutex_t  *m;
    union semun   op;

    m = ngx_alloc(sizeof(ngx_mutex_t), log);
    if (m == NULL) {
        return NULL;
    }

    m->lock = 0;
    m->log = log;

    if (flags & NGX_MUTEX_LIGHT) {
        m->semid = -1;
        return m;
    }

    m->semid = semget(IPC_PRIVATE, 1, SEM_R|SEM_A);
    if (m->semid == -1) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno, "semget() failed");
        return NULL;
    }

    op.val = 0;

    if (semctl(m->semid, 0, SETVAL, op) == -1) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno, "semctl(SETVAL) failed");

        if (semctl(m->semid, 0, IPC_RMID) == -1) {
            ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                          "semctl(IPC_RMID) failed");
        }

        return NULL;
    }

    return m;
}


void
ngx_mutex_destroy(ngx_mutex_t *m)
{
    if (semctl(m->semid, 0, IPC_RMID) == -1) {
        ngx_log_error(NGX_LOG_ALERT, m->log, ngx_errno,
                      "semctl(IPC_RMID) failed");
    }

    ngx_free((void *) m);
}


ngx_int_t
ngx_mutex_dolock(ngx_mutex_t *m, ngx_int_t try)
{
    uint32_t       lock, old;
    ngx_uint_t     tries;
    struct sembuf  op;

    if (!ngx_threaded) {
        return NGX_OK;
    }

```

```
#if (NGX_DEBUG)
    if (try) {
        ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                       "try lock mutex %p lock:%XD", m, m->lock);
    } else {
        ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                       "lock mutex %p lock:%XD", m, m->lock);
    }
#endif

    old = m->lock;
    tries = 0;

    for ( ;; ) {
        if (old & NGX_MUTEX_LOCK_BUSY) {

            if (try) {
                return NGX_AGAIN;
            }

            if (ngx_ncpu > 1 && tries++ < 1000) {

                /* the spinlock is used only on the SMP system */

                old = m->lock;
                continue;
            }

            if (m->semid == -1) {
                sched_yield();

                tries = 0;
                old = m->lock;
                continue;
            }

            ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                           "mutex %p lock:%XD", m, m->lock);

            /*
             * The mutex is locked so we increase a number
             * of the threads that are waiting on the mutex
             */

            lock = old + 1;

            if ((lock & ~NGX_MUTEX_LOCK_BUSY) > nthreads) {
                ngx_log_error(NGX_LOG_ALERT, m->log, ngx_errno,
                              "%D threads wait for mutex %p, "
                              "while only %ui threads are available",
                              lock & ~NGX_MUTEX_LOCK_BUSY, m, nthreads);
                ngx_abort();
            }

            if (ngx_atomic_cmp_set(&m->lock, old, lock)) {

                ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                               "wait mutex %p lock:%XD", m, m->lock);

                /*
                 * The number of the waiting threads has been increased
                 * and we would wait on the SysV semaphore.
                 * A semaphore should wake up us more efficiently than
                 * a simple sched_yield() or usleep().
                 */

                op.sem_num = 0;
                op.sem_op = -1;
                op.sem_flg = 0;

                if (semop(m->semid, &op, 1) == -1) {
                    ngx_log_error(NGX_LOG_ALERT, m->log, ngx_errno,
                                  "semop() failed while waiting on mutex %p", m);
                    ngx_abort();
                }

```

```
                ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                               "mutex waked up %p lock:%XD", m, m->lock);

                tries = 0;
                old = m->lock;
                continue;
            }

            old = m->lock;

        } else {
            lock = old | NGX_MUTEX_LOCK_BUSY;

            if (ngx_atomic_cmp_set(&m->lock, old, lock)) {

                /* we locked the mutex */

                break;
            }

            old = m->lock;
        }

        if (tries++ > 1000) {

            ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                           "mutex %p is contested", m);

            /* the mutex is probably contested so we are giving up now */

            sched_yield();

            tries = 0;
            old = m->lock;
        }
    }

    ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                   "mutex %p is locked, lock:%XD", m, m->lock);

    return NGX_OK;
}


void
ngx_mutex_unlock(ngx_mutex_t *m)
{
    uint32_t       lock, old;
    struct sembuf  op;

    if (!ngx_threaded) {
        return;
    }

    old = m->lock;

    if (!(old & NGX_MUTEX_LOCK_BUSY)) {
        ngx_log_error(NGX_LOG_ALERT, m->log, 0,
                      "trying to unlock the free mutex %p", m);
        ngx_abort();
    }

    /* free the mutex */

#if 0
    ngx_log_debug2(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                   "unlock mutex %p lock:%XD", m, old);
#endif

    for ( ;; ) {
        lock = old & ~NGX_MUTEX_LOCK_BUSY;

        if (ngx_atomic_cmp_set(&m->lock, old, lock)) {
            break;
        }
```

```
        old = m->lock;
    }

    if (m->semid == -1) {
        ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                       "mutex %p is unlocked", m);

        return;
    }

    /* check whether we need to wake up a waiting thread */

    old = m->lock;

    for ( ;; ) {
        if (old & NGX_MUTEX_LOCK_BUSY) {

            /* the mutex is just locked by another thread */

            break;
        }

        if (old == 0) {
            break;
        }

        /* there are the waiting threads */

        lock = old - 1;

        if (ngx_atomic_cmp_set(&m->lock, old, lock)) {

            /* wake up the thread that waits on semaphore */

            ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                           "wake up mutex %p", m);

            op.sem_num = 0;
            op.sem_op = 1;
            op.sem_flg = 0;

            if (semop(m->semid, &op, 1) == -1) {
                ngx_log_error(NGX_LOG_ALERT, m->log, ngx_errno,
                              "semop() failed while waking up on mutex %p", m);
                ngx_abort();
            }

            break;
        }

        old = m->lock;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MUTEX, m->log, 0,
                   "mutex %p is unlocked", m);

    return;
}


ngx_cond_t *
ngx_cond_init(ngx_log_t *log)
{
    ngx_cond_t  *cv;

    cv = ngx_alloc(sizeof(ngx_cond_t), log);
    if (cv == NULL) {
        return NULL;
    }

    cv->signo = NGX_CV_SIGNAL;
    cv->tid = -1;
    cv->log = log;
    cv->kq = -1;

    return cv;
```

```
}


void
ngx_cond_destroy(ngx_cond_t *cv)
{
    if (close(cv->kq) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cv->log, ngx_errno,
                      "kqueue close() failed");
    }

    ngx_free(cv);
}


ngx_int_t
ngx_cond_wait(ngx_cond_t *cv, ngx_mutex_t *m)
{
    int              n;
    ngx_err_t        err;
    struct kevent    kev;
    struct timespec  ts;

    if (cv->kq == -1) {

        /*
         * We have to add the EVFILT_SIGNAL filter in the rfork()ed thread.
         * Otherwise the thread would not get a signal event.
         *
         * However, we have not to open the kqueue in the thread,
         * it is simply handy do it together.
         */

        cv->kq = kqueue();
        if (cv->kq == -1) {
            ngx_log_error(NGX_LOG_ALERT, cv->log, ngx_errno, "kqueue() failed");
            return NGX_ERROR;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, cv->log, 0,
                       "cv kq:%d signo:%d", cv->kq, cv->signo);

        kev.ident = cv->signo;
        kev.filter = EVFILT_SIGNAL;
        kev.flags = EV_ADD;
        kev.fflags = 0;
        kev.data = 0;
        kev.udata = NULL;

        ts.tv_sec = 0;
        ts.tv_nsec = 0;

        if (kevent(cv->kq, &kev, 1, NULL, 0, &ts) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cv->log, ngx_errno, "kevent() failed");
            return NGX_ERROR;
        }

        cv->tid = ngx_thread_self();
    }

    ngx_mutex_unlock(m);

    ngx_log_debug3(NGX_LOG_DEBUG_CORE, cv->log, 0,
                   "cv %p wait, kq:%d, signo:%d", cv, cv->kq, cv->signo);

    for ( ;; ) {
        n = kevent(cv->kq, NULL, 0, &kev, 1, NULL);

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, cv->log, 0,
                       "cv %p kevent: %d", cv, n);

        if (n == -1) {
            err = ngx_errno;
            ngx_log_error((err == NGX_EINTR) ? NGX_LOG_INFO : NGX_LOG_ALERT,
                          cv->log, ngx_errno,
                          "kevent() failed while waiting condition variable %p",
```

```
                          cv);

            if (err == NGX_EINTR) {
                break;
            }

            return NGX_ERROR;
        }

        if (n == 0) {
            ngx_log_error(NGX_LOG_ALERT, cv->log, 0,
                          "kevent() returned no events "
                          "while waiting condition variable %p",
                          cv);
            continue;
        }

        if (kev.filter != EVFILT_SIGNAL) {
            ngx_log_error(NGX_LOG_ALERT, cv->log, 0,
                          "kevent() returned unexpected events: %d "
                          "while waiting condition variable %p",
                          kev.filter, cv);
            continue;
        }

        if (kev.ident != (uintptr_t) cv->signo) {
            ngx_log_error(NGX_LOG_ALERT, cv->log, 0,
                          "kevent() returned unexpected signal: %d ",
                          "while waiting condition variable %p",
                          kev.ident, cv);
            continue;
        }

        break;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, cv->log, 0, "cv %p is waked up", cv);

    ngx_mutex_lock(m);

    return NGX_OK;
}


ngx_int_t
ngx_cond_signal(ngx_cond_t *cv)
{
    ngx_err_t  err;

    ngx_log_debug3(NGX_LOG_DEBUG_CORE, cv->log, 0,
                   "cv %p to signal %P %d",
                   cv, cv->tid, cv->signo);

    if (cv->tid == -1) {
        return NGX_OK;
    }

    if (kill(cv->tid, cv->signo) == -1) {

        err = ngx_errno;

        ngx_log_error(NGX_LOG_ALERT, cv->log, err,
                      "kill() failed while signaling condition variable %p", cv);

        if (err == NGX_ESRCH) {
            cv->tid = -1;
        }

        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, cv->log, 0, "cv %p is signaled", cv);

    return NGX_OK;
}
```

src/os/unix/ngx_gcc_atomic_sparc64.h - nginx-1.7.10

Functions defined

- ngx_atomic_cmp_set
- ngx_atomic_fetch_add

Macros defined

- NGX_CASA
- NGX_CASA
- ngx_cpu_pause
- ngx_memory_barrier
- ngx_memory_barrier

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  /*
9  * "casa   [r1] 0x80, r2, r0"  and
10 * "casxa  [r1] 0x80, r2, r0"  do the following:
11 *
12 *     if ([r1] == r2) {
13 *         swap(r0, [r1]);
14 *     } else {
15 *         r0 = [r1];
16 *     }
17 *
18 * so "r0 == r2" means that the operation was successful.
19 *
20 *
21 * The "r" means the general register.
22 * The "+r" means the general register used for both input and output.
23 */
24
25
26 #if (NGX_PTR_SIZE == 4)
27 #define NGX_CASA  "casa"
28 #else
29 #define NGX_CASA  "casxa"
30 #endif
31
32
33 static ngx_inline ngx_atomic_uint_t
34 ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
35     ngx_atomic_uint_t set)
36 {
37     __asm__ volatile (
38
39     NGX_CASA " [%1] 0x80, %2, %0"
40
41     : "+r" (set) : "r" (lock), "r" (old) : "memory");
42
43     return (set == old);
44 }
45
```

```
46
47  static ngx_inline ngx_atomic_int_t
48  ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
49  {
50      ngx_atomic_uint_t  old, res;
51
52      old = *value;
53
54      for ( ;; ) {
55
56          res = old + add;
57
58          __asm__ volatile (
59
60          NGX_CASA " [%1] 0x80, %2, %0"
61
62          : "+r" (res) : "r" (value), "r" (old) : "memory");
63
64          if (res == old) {
65              return res;
66          }
67
68          old = res;
69      }
70  }
71
72
73  #if (NGX_SMP)
74  #define ngx_memory_barrier()                                                \
75              __asm__ volatile (                                              \
76              "membar #LoadLoad | #LoadStore | #StoreStore | #StoreLoad"      \
77              ::: "memory")
78  #else
79  #define ngx_memory_barrier()   __asm__ volatile ("" ::: "memory")
80  #endif
81
82  #define ngx_cpu_pause()
```

One Level Up Top Level

src/os/unix/ngx_freebsd_rfork_thread.h - nginx-1.7.10

Data types defined

- ngx_cond_t
- ngx_mutex_t
- ngx_rfork_thread_func_pt
- ngx_tid_t
- ngx_tls_key_t

Functions defined

- ngx_gettid
- ngx_thread_get_tls

Macros defined

- NGX_CV_SIGNAL
- NGX_MUTEX_LIGHT
- NGX_MUTEX_LOCK_BUSY
- NGX_THREAD_KEYS_MAX
- NGX_TID_T_FMT
- _NGX_FREEBSD_RFORK_THREAD_H_INCLUDED_
- ngx_log_pid
- ngx_log_tid
- ngx_mutex_lock
- ngx_mutex_trylock
- ngx_setthrtitle
- ngx_thread_join
- ngx_thread_key_create_n
- ngx_thread_set_tls_n
- ngx_thread_sigmask
- ngx_thread_sigmask_n

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
```

```
6
7
8  #ifndef _NGX_FREEBSD_RFORK_THREAD_H_INCLUDED_
9  #define _NGX_FREEBSD_RFORK_THREAD_H_INCLUDED_
10
11
12 #include <sys/ipc.h>
13 #include <sys/sem.h>
14 #include <sched.h>
15
16 typedef pid_t  ngx_tid_t;
17
18 #define ngx_log_pid    ngx_thread_self()
19 #define ngx_log_tid    0
20
21 #define NGX_TID_T_FMT  "%P"
22
23
24 #define NGX_MUTEX_LIGHT      1
25
26 #define NGX_MUTEX_LOCK_BUSY  0x80000000
27
28 typedef volatile struct {
29     ngx_atomic_t  lock;
30     ngx_log_t    *log;
31     int           semid;
32 } ngx_mutex_t;
33
34
35 #define NGX_CV_SIGNAL        64
36
37 typedef struct {
38     int           signo;
39     int           kq;
40     ngx_tid_t     tid;
41     ngx_log_t    *log;
42 } ngx_cond_t;
43
44
45 #define ngx_thread_sigmask(how, set, oset)                          \
46             (sigprocmask(how, set, oset) == -1) ? ngx_errno : 0
47
48 #define ngx_thread_sigmask_n   "sigprocmask()"
49
50 #define ngx_thread_join(t, p)
51
52 #define ngx_setthrtitle(n)     setproctitle(n)
53
54
55 extern char    *ngx_freebsd_kern_usrstack;
56 extern size_t   ngx_thread_stack_size;
57
58
59 static ngx_inline ngx_int_t
60 ngx_gettid(void)
61 {
62     char  *sp;
63
64     if (ngx_thread_stack_size == 0) {
65         return 0;
66     }
67
68 #if ( __i386__ )
69
70     __asm__ volatile ("mov %%esp, %0" : "=q" (sp));
71
72 #elif ( __amd64__ )
73
74     __asm__ volatile ("mov %%rsp, %0" : "=q" (sp));
75
76 #else
77
78 #error "rfork()ed threads are not supported on this platform"
79
80 #endif
81
```

```
82      return (ngx_freebsd_kern_usrstack - sp) / ngx_thread_stack_size;
83  }
84
85
86  ngx_tid_t ngx_thread_self(void);
87
88
89  typedef ngx_uint_t               ngx_tls_key_t;
90
91  #define NGX_THREAD_KEYS_MAX      16
92
93  extern void    **ngx_tls;
94
95  ngx_err_t ngx_thread_key_create(ngx_tls_key_t *key);
96  #define ngx_thread_key_create_n  "the tls key creation"
97
98  ngx_err_t ngx_thread_set_tls(ngx_tls_key_t key, void *value);
99  #define ngx_thread_set_tls_n     "the tls key setting"
100
101
102  static void *
103  ngx_thread_get_tls(ngx_tls_key_t key)
104  {
105      if (key >= NGX_THREAD_KEYS_MAX) {
106          return NULL;
107      }
108
109      return ngx_tls[key * NGX_THREAD_KEYS_MAX + ngx_gettid()];
110  }
111
112
113  #define ngx_mutex_trylock(m)  ngx_mutex_dolock(m, 1)
114  #define ngx_mutex_lock(m)     (void) ngx_mutex_dolock(m, 0)
115  ngx_int_t ngx_mutex_dolock(ngx_mutex_t *m, ngx_int_t try);
116  void ngx_mutex_unlock(ngx_mutex_t *m);
117
118
119  typedef int (*ngx_rfork_thread_func_pt)(void *arg);
120
121
122  #endif /* _NGX_FREEBSD_RFORK_THREAD_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_posix_init.c - nginx-1.7.10

Global variables defined

- ngx_inherited_nonblocking
- ngx_max_sockets
- ngx_ncpu
- ngx_os_io
- ngx_tcp_nodelay_and_tcp_nopush
- rlmt

Functions defined

- ngx_os_init
- ngx_os_status

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <nginx.h>
11
12
13  ngx_int_t   ngx_ncpu;
14  ngx_int_t   ngx_max_sockets;
15  ngx_uint_t  ngx_inherited_nonblocking;
16  ngx_uint_t  ngx_tcp_nodelay_and_tcp_nopush;
17
18
19  struct rlimit  rlmt;
20
21
22  ngx_os_io_t ngx_os_io = {
23      ngx_unix_recv,
24      ngx_readv_chain,
25      ngx_udp_unix_recv,
26      ngx_unix_send,
27      ngx_writev_chain,
28      0
29  };
30
31
32  ngx_int_t
33  ngx_os_init(ngx_log_t *log)
34  {
35      ngx_uint_t  n;
36
37  #if (NGX_HAVE_OS_SPECIFIC_INIT)
38      if (ngx_os_specific_init(log) != NGX_OK) {
39          return NGX_ERROR;
40      }
41  #endif
42
43      if (ngx_init_setproctitle(log) != NGX_OK) {
```

```
        return NGX_ERROR;
    }

    ngx_pagesize = getpagesize();
    ngx_cacheline_size = NGX_CPU_CACHE_LINE;

    for (n = ngx_pagesize; n >>= 1; ngx_pagesize_shift++) { /* void */ }

#if (NGX_HAVE_SC_NPROCESSORS_ONLN)
    if (ngx_ncpu == 0) {
        ngx_ncpu = sysconf(_SC_NPROCESSORS_ONLN);
    }
#endif

    if (ngx_ncpu < 1) {
        ngx_ncpu = 1;
    }

    ngx_cpuinfo();

    if (getrlimit(RLIMIT_NOFILE, &rlmt) == -1) {
        ngx_log_error(NGX_LOG_ALERT, log, errno,
                      "getrlimit(RLIMIT_NOFILE) failed)");
        return NGX_ERROR;
    }

    ngx_max_sockets = (ngx_int_t) rlmt.rlim_cur;

#if (NGX_HAVE_INHERITED_NONBLOCK || NGX_HAVE_ACCEPT4)
    ngx_inherited_nonblocking = 1;
#else
    ngx_inherited_nonblocking = 0;
#endif

    srandom(ngx_time());

    return NGX_OK;
}


void
ngx_os_status(ngx_log_t *log)
{
    ngx_log_error(NGX_LOG_NOTICE, log, 0, NGINX_VER_BUILD);

#ifdef NGX_COMPILER
    ngx_log_error(NGX_LOG_NOTICE, log, 0, "built by " NGX_COMPILER);
#endif

#if (NGX_HAVE_OS_SPECIFIC_INIT)
    ngx_os_specific_status(log);
#endif

    ngx_log_error(NGX_LOG_NOTICE, log, 0,
                  "getrlimit(RLIMIT_NOFILE): %r:%r",
                  rlmt.rlim_cur, rlmt.rlim_max);
 }


 #if 0

 ngx_int_t
 ngx_posix_post_conf_init(ngx_log_t *log)
 {
     ngx_fd_t  pp[2];

     if (pipe(pp) == -1) {
         ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "pipe() failed");
         return NGX_ERROR;
     }

     if (dup2(pp[1], STDERR_FILENO) == -1) {
         ngx_log_error(NGX_LOG_EMERG, log, errno, "dup2(STDERR) failed");
         return NGX_ERROR;
     }

```

```
    if (pp[1] > STDERR_FILENO) {
        if (close(pp[1]) == -1) {
            ngx_log_error(NGX_LOG_EMERG, log, errno, "close() failed");
            return NGX_ERROR;
        }
    }

    return NGX_OK;
}

#endif
```

One Level Up Top Level

src/os/unix/ngx_os.h - nginx-1.7.10

Data types defined

- ngx_iovec_t
- ngx_os_io_t
- ngx_recv_chain_pt
- ngx_recv_pt
- ngx_send_chain_pt
- ngx_send_pt

Macros defined

- NGX_IOVS_PREALLOCATE
- NGX_IOVS_PREALLOCATE
- NGX_IO_SENDFILE
- _NGX_OS_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_OS_H_INCLUDED_
9  #define _NGX_OS_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #define NGX_IO_SENDFILE    1
17
18
19  typedef ssize_t (*ngx_recv_pt)(ngx_connection_t *c, u_char *buf, size_t size);
20  typedef ssize_t (*ngx_recv_chain_pt)(ngx_connection_t *c, ngx_chain_t *in,
21      off_t limit);
22  typedef ssize_t (*ngx_send_pt)(ngx_connection_t *c, u_char *buf, size_t size);
23  typedef ngx_chain_t *(*ngx_send_chain_pt)(ngx_connection_t *c, ngx_chain_t *in,
24      off_t limit);
25
26  typedef struct {
27      ngx_recv_pt        recv;
28      ngx_recv_chain_pt  recv_chain;
29      ngx_recv_pt        udp_recv;
30      ngx_send_pt        send;
31      ngx_send_chain_pt  send_chain;
32      ngx_uint_t         flags;
33  } ngx_os_io_t;
34
35
36  ngx_int_t ngx_os_init(ngx_log_t *log);
37  void ngx_os_status(ngx_log_t *log);
38  ngx_int_t ngx_os_specific_init(ngx_log_t *log);
```

```
39  void ngx_os_specific_status(ngx_log_t *log);
40  ngx_int_t ngx_daemon(ngx_log_t *log);
41  ngx_int_t ngx_os_signal_process(ngx_cycle_t *cycle, char *sig, ngx_int_t pid);
42
43
44  ssize_t ngx_unix_recv(ngx_connection_t *c, u_char *buf, size_t size);
45  ssize_t ngx_readv_chain(ngx_connection_t *c, ngx_chain_t *entry, off_t limit);
46  ssize_t ngx_udp_unix_recv(ngx_connection_t *c, u_char *buf, size_t size);
47  ssize_t ngx_unix_send(ngx_connection_t *c, u_char *buf, size_t size);
48  ngx_chain_t *ngx_writev_chain(ngx_connection_t *c, ngx_chain_t *in,
49      off_t limit);
50
51  #if (NGX_HAVE_AIO)
52  ssize_t ngx_aio_read(ngx_connection_t *c, u_char *buf, size_t size);
53  ssize_t ngx_aio_read_chain(ngx_connection_t *c, ngx_chain_t *cl, off_t limit);
54  ssize_t ngx_aio_write(ngx_connection_t *c, u_char *buf, size_t size);
55  ngx_chain_t *ngx_aio_write_chain(ngx_connection_t *c, ngx_chain_t *in,
56      off_t limit);
57  #endif
58
59
60  #if (IOV_MAX > 64)
61  #define NGX_IOVS_PREALLOCATE  64
62  #else
63  #define NGX_IOVS_PREALLOCATE  IOV_MAX
64  #endif
65
66
67  typedef struct {
68      struct iovec  *iovs;
69      ngx_uint_t     count;
70      size_t         size;
71      ngx_uint_t     nalloc;
72  } ngx_iovec_t;
73
74  ngx_chain_t *ngx_output_chain_to_iovec(ngx_iovec_t *vec, ngx_chain_t *in,
75      size_t limit, ngx_log_t *log);
76
77
78  ssize_t ngx_writev(ngx_connection_t *c, ngx_iovec_t *vec);
79
80
81  extern ngx_os_io_t  ngx_os_io;
82  extern ngx_int_t    ngx_ncpu;
83  extern ngx_int_t    ngx_max_sockets;
84  extern ngx_uint_t   ngx_inherited_nonblocking;
85  extern ngx_uint_t   ngx_tcp_nodelay_and_tcp_nopush;
86
87
88  #if (NGX_FREEBSD)
89  #include <ngx_freebsd.h>
90
91
92  #elif (NGX_LINUX)
93  #include <ngx_linux.h>
94
95
96  #elif (NGX_SOLARIS)
97  #include <ngx_solaris.h>
98
99
100 #elif (NGX_DARWIN)
101 #include <ngx_darwin.h>
102 #endif
103
104
105 #endif /* _NGX_OS_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_freebsd_init.c - nginx-1.7.10

Global variables defined

- ngx_debug_malloc
- ngx_freebsd_hw_ncpu
- ngx_freebsd_io
- ngx_freebsd_kern_ipc_somaxconn
- ngx_freebsd_kern_osreldate
- ngx_freebsd_kern_osrelease
- ngx_freebsd_kern_ostype
- ngx_freebsd_machdep_hlt_logical_cpus
- ngx_freebsd_net_inet_tcp_sendspace
- ngx_freebsd_sendfile_nbytes_bug
- ngx_freebsd_use_tcp_nopush
- sysctls

Data types defined

- sysctl_t

Functions defined

- ngx_debug_init
- ngx_os_specific_init
- ngx_os_specific_status

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /* FreeBSD 3.0 at least */
13  char    ngx_freebsd_kern_ostype[16];
14  char    ngx_freebsd_kern_osrelease[128];
15  int     ngx_freebsd_kern_osreldate;
16  int     ngx_freebsd_hw_ncpu;
17  int     ngx_freebsd_kern_ipc_somaxconn;
18  u_long  ngx_freebsd_net_inet_tcp_sendspace;
19
20  /* FreeBSD 4.9 */
21  int     ngx_freebsd_machdep_hlt_logical_cpus;
```

```
22
23
24  ngx_uint_t  ngx_freebsd_sendfile_nbytes_bug;
25  ngx_uint_t  ngx_freebsd_use_tcp_nopush;
26
27  ngx_uint_t  ngx_debug_malloc;
28
29
30  static ngx_os_io_t ngx_freebsd_io = {
31      ngx_unix_recv,
32      ngx_readv_chain,
33      ngx_udp_unix_recv,
34      ngx_unix_send,
35  #if (NGX_HAVE_SENDFILE)
36      ngx_freebsd_sendfile_chain,
37      NGX_IO_SENDFILE
38  #else
39      ngx_writev_chain,
40      0
41  #endif
42  };
43
44
45  typedef struct {
46      char        *name;
47      void        *value;
48      size_t       size;
49      ngx_uint_t   exists;
50  } sysctl_t;
51
52
53  sysctl_t sysctls[] = {
54      { "hw.ncpu",
55        &ngx_freebsd_hw_ncpu,
56        sizeof(ngx_freebsd_hw_ncpu), 0 },
57
58      { "machdep.hlt_logical_cpus",
59        &ngx_freebsd_machdep_hlt_logical_cpus,
60        sizeof(ngx_freebsd_machdep_hlt_logical_cpus), 0 },
61
62      { "net.inet.tcp.sendspace",
63        &ngx_freebsd_net_inet_tcp_sendspace,
64        sizeof(ngx_freebsd_net_inet_tcp_sendspace), 0 },
65
66      { "kern.ipc.somaxconn",
67        &ngx_freebsd_kern_ipc_somaxconn,
68        sizeof(ngx_freebsd_kern_ipc_somaxconn), 0 },
69
70      { NULL, NULL, 0, 0 }
71  };
72
73
74  void
75  ngx_debug_init(void)
76  {
77  #if (NGX_DEBUG_MALLOC)
78
79  #if __FreeBSD_version >= 500014 && __FreeBSD_version < 1000011
80      _malloc_options = "J";
81  #elif __FreeBSD_version < 500014
82      malloc_options = "J";
83  #endif
84
85      ngx_debug_malloc = 1;
86
87  #else
88      char  *mo;
89
90      mo = getenv("MALLOC_OPTIONS");
91
92      if (mo && ngx_strchr(mo, 'J')) {
93          ngx_debug_malloc = 1;
94      }
95  #endif
96  }
97
```

```
98
99  ngx_int_t
100 ngx_os_specific_init(ngx_log_t *log)
101 {
102     int         version;
103     size_t      size;
104     ngx_err_t   err;
105     ngx_uint_t  i;
106
107     size = sizeof(ngx_freebsd_kern_ostype);
108     if (sysctlbyname("kern.ostype",
109                      ngx_freebsd_kern_ostype, &size, NULL, 0) == -1) {
110         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
111                       "sysctlbyname(kern.ostype) failed");
112
113         if (ngx_errno != NGX_ENOMEM) {
114             return NGX_ERROR;
115         }
116
117         ngx_freebsd_kern_ostype[size - 1] = '\0';
118     }
119
120     size = sizeof(ngx_freebsd_kern_osrelease);
121     if (sysctlbyname("kern.osrelease",
122                      ngx_freebsd_kern_osrelease, &size, NULL, 0) == -1) {
123         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
124                       "sysctlbyname(kern.osrelease) failed");
125
126         if (ngx_errno != NGX_ENOMEM) {
127             return NGX_ERROR;
128         }
129
130         ngx_freebsd_kern_osrelease[size - 1] = '\0';
131     }
132
133
134     size = sizeof(int);
135     if (sysctlbyname("kern.osreldate",
136                      &ngx_freebsd_kern_osreldate, &size, NULL, 0) == -1) {
137         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
138                       "sysctlbyname(kern.osreldate) failed");
139         return NGX_ERROR;
140     }
141
142     version = ngx_freebsd_kern_osreldate;
143
144
145 #if (NGX_HAVE_SENDFILE)
146
147     /*
148      * The determination of the sendfile() "nbytes bug" is complex enough.
149      * There are two sendfile() syscalls: a new #393 has no bug while
150      * an old #336 has the bug in some versions and has not in others.
151      * Besides libc_r wrapper also emulates the bug in some versions.
152      * There is no way to say exactly if syscall #336 in FreeBSD circa 4.6
153      * has the bug.  We use the algorithm that is correct at least for
154      * RELEASEs and for syscalls only (not libc_r wrapper).
155      *
156      * 4.6.1-RELEASE and below have the bug
157      * 4.6.2-RELEASE and above have the new syscall
158      *
159      * We detect the new sendfile() syscall available at the compile time
160      * to allow an old binary to run correctly on an updated FreeBSD system.
161      */
162
163 #if (__FreeBSD__ == 4 && __FreeBSD_version >= 460102) \
164     || __FreeBSD_version == 460002 || __FreeBSD_version >= 500039
165
166     /* a new syscall without the bug */
167
168     ngx_freebsd_sendfile_nbytes_bug = 0;
169
170 #else
171
172     /* an old syscall that may have the bug */
173
```

```
    ngx_freebsd_sendfile_nbytes_bug = 1;

#endif

#endif /* NGX_HAVE_SENDFILE */


    if ((version < 500000 && version >= 440003) || version >= 500017) {
        ngx_freebsd_use_tcp_nopush = 1;
    }


    for (i = 0; sysctls[i].name; i++) {
        size = sysctls[i].size;

        if (sysctlbyname(sysctls[i].name, sysctls[i].value, &size, NULL, 0)
            == 0)
        {
            sysctls[i].exists = 1;
            continue;
        }

        err = ngx_errno;

        if (err == NGX_ENOENT) {
            continue;
        }

        ngx_log_error(NGX_LOG_ALERT, log, err,
                      "sysctlbyname(%s) failed", sysctls[i].name);
        return NGX_ERROR;
    }

    if (ngx_freebsd_machdep_hlt_logical_cpus) {
        ngx_ncpu = ngx_freebsd_hw_ncpu / 2;

    } else {
        ngx_ncpu = ngx_freebsd_hw_ncpu;
    }

    if (version < 600008 && ngx_freebsd_kern_ipc_somaxconn > 32767) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "sysctl kern.ipc.somaxconn must be less than 32768");
        return NGX_ERROR;
    }

    ngx_tcp_nodelay_and_tcp_nopush = 1;

    ngx_os_io = ngx_freebsd_io;

    return NGX_OK;
}


void
ngx_os_specific_status(ngx_log_t *log)
{
    u_long      value;
    ngx_uint_t  i;

    ngx_log_error(NGX_LOG_NOTICE, log, 0, "OS: %s %s",
                  ngx_freebsd_kern_ostype, ngx_freebsd_kern_osrelease);

#ifdef __DragonFly_version
    ngx_log_error(NGX_LOG_NOTICE, log, 0,
                  "kern.osreldate: %d, built on %d",
                  ngx_freebsd_kern_osreldate, __DragonFly_version);
#else
    ngx_log_error(NGX_LOG_NOTICE, log, 0,
                  "kern.osreldate: %d, built on %d",
                  ngx_freebsd_kern_osreldate, __FreeBSD_version);
#endif

    for (i = 0; sysctls[i].name; i++) {
        if (sysctls[i].exists) {
            if (sysctls[i].size == sizeof(long)) {
```

```
250                 value = *(long *) sysctls[i].value;
251
252             } else {
253                 value = *(int *) sysctls[i].value;
254             }
255
256             ngx_log_error(NGX_LOG_NOTICE, log, 0, "%s: %l",
257                           sysctls[i].name, value);
258         }
259     }
260 }
```

One Level Up Top Level

src/os/unix/ngx_recv.c - nginx-1.7.10

Functions defined

- ngx_unix_recv
- ngx_unix_recv

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #if (NGX_HAVE_KQUEUE)
14
15  ssize_t
16  ngx_unix_recv(ngx_connection_t *c, u_char *buf, size_t size)
17  {
18      ssize_t       n;
19      ngx_err_t     err;
20      ngx_event_t  *rev;
21
22      rev = c->read;
23
24      if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
25          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
26                         "recv: eof:%d, avail:%d, err:%d",
27                         rev->pending_eof, rev->available, rev->kq_errno);
28
29          if (rev->available == 0) {
30              if (rev->pending_eof) {
31                  rev->ready = 0;
32                  rev->eof = 1;
33
34                  if (rev->kq_errno) {
35                      rev->error = 1;
36                      ngx_set_socket_errno(rev->kq_errno);
37
38                      return ngx_connection_error(c, rev->kq_errno,
39                                 "kevent() reported about an closed connection");
40                  }
41
42                  return 0;
43
44              } else {
45                  rev->ready = 0;
46                  return NGX_AGAIN;
47              }
48          }
49      }
50
51      do {
52          n = recv(c->fd, buf, size, 0);
53
54          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
55                         "recv: fd:%d %d of %d", c->fd, n, size);
56
57          if (n >= 0) {
58              if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
59                  rev->available -= n;
60
```

```
                /*
                 * rev->available may be negative here because some additional
                 * bytes may be received between kevent() and recv()
                 */

                if (rev->available <= 0) {
                    if (!rev->pending_eof) {
                        rev->ready = 0;
                    }

                    if (rev->available < 0) {
                        rev->available = 0;
                    }
                }

                if (n == 0) {

                    /*
                     * on FreeBSD recv() may return 0 on closed socket
                     * even if kqueue reported about available data
                     */

                    rev->ready = 0;
                    rev->eof = 1;
                    rev->available = 0;
                }

                return n;
            }

            if ((size_t) n < size
                && !(ngx_event_flags & NGX_USE_GREEDY_EVENT))
            {
                rev->ready = 0;
            }

            if (n == 0) {
                rev->eof = 1;
            }

             return n;
        }

        err = ngx_socket_errno;

        if (err == NGX_EAGAIN || err == NGX_EINTR) {
            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
                           "recv() not ready");
            n = NGX_AGAIN;

        } else {
            n = ngx_connection_error(c, err, "recv() failed");
            break;
        }

    } while (err == NGX_EINTR);

    rev->ready = 0;

    if (n == NGX_ERROR) {
        rev->error = 1;
    }

    return n;
}

#else /* ! NGX_HAVE_KQUEUE */

ssize_t
ngx_unix_recv(ngx_connection_t *c, u_char *buf, size_t size)
{
    ssize_t       n;
    ngx_err_t     err;
    ngx_event_t  *rev;

    rev = c->read;
```

```
137
138     do {
139         n = recv(c->fd, buf, size, 0);
140
141         ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
142                        "recv: fd:%d %d of %d", c->fd, n, size);
143
144         if (n == 0) {
145             rev->ready = 0;
146             rev->eof = 1;
147             return n;
148
149         } else if (n > 0) {
150
151             if ((size_t) n < size
152                 && !(ngx_event_flags & NGX_USE_GREEDY_EVENT))
153             {
154                 rev->ready = 0;
155             }
156
157             return n;
158         }
159
160         err = ngx_socket_errno;
161
162         if (err == NGX_EAGAIN || err == NGX_EINTR) {
163             ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
164                            "recv() not ready");
165             n = NGX_AGAIN;
166
167         } else {
168             n = ngx_connection_error(c, err, "recv() failed");
169             break;
170         }
171
172     } while (err == NGX_EINTR);
173
174     rev->ready = 0;
175
176     if (n == NGX_ERROR) {
177         rev->error = 1;
178     }
179
180     return n;
181 }
182
183 #endif /* NGX_HAVE_KQUEUE */
```

One Level Up Top Level

src/core/ngx_connection.c - nginx-1.7.10

Global variables defined

- ngx_io

Functions defined

- ngx_close_connection
- ngx_close_listening_sockets
- ngx_configure_listening_sockets
- ngx_connection_error
- ngx_connection_local_sockaddr
- ngx_create_listening
- ngx_drain_connections
- ngx_free_connection
- ngx_get_connection
- ngx_open_listening_sockets
- ngx_reusable_connection
- ngx_set_inherited_sockets

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ngx_os_io_t  ngx_io;
14
15
16  static void ngx_drain_connections(void);
17
18
19  ngx_listening_t *
20  ngx_create_listening(ngx_conf_t *cf, void *sockaddr, socklen_t socklen)
21  {
22      size_t            len;
23      ngx_listening_t  *ls;
24      struct sockaddr  *sa;
25      u_char            text[NGX_SOCKADDR_STRLEN];
26
27      ls = ngx_array_push(&cf->cycle->listening);
28      if (ls == NULL) {
29          return NULL;
30      }
31
32      ngx_memzero(ls, sizeof(ngx_listening_t));
```

```
33
34      sa = ngx_palloc(cf->pool, socklen);
35      if (sa == NULL) {
36          return NULL;
37      }
38
39      ngx_memcpy(sa, sockaddr, socklen);
40
41      ls->sockaddr = sa;
42      ls->socklen = socklen;
43
44      len = ngx_sock_ntop(sa, socklen, text, NGX_SOCKADDR_STRLEN, 1);
45      ls->addr_text.len = len;
46
47      switch (ls->sockaddr->sa_family) {
48  #if (NGX_HAVE_INET6)
49      case AF_INET6:
50           ls->addr_text_max_len = NGX_INET6_ADDRSTRLEN;
51           break;
52  #endif
53  #if (NGX_HAVE_UNIX_DOMAIN)
54      case AF_UNIX:
55           ls->addr_text_max_len = NGX_UNIX_ADDRSTRLEN;
56           len++;
57           break;
58  #endif
59      case AF_INET:
60           ls->addr_text_max_len = NGX_INET_ADDRSTRLEN;
61           break;
62      default:
63           ls->addr_text_max_len = NGX_SOCKADDR_STRLEN;
64           break;
65      }
66
67      ls->addr_text.data = ngx_pnalloc(cf->pool, len);
68      if (ls->addr_text.data == NULL) {
69          return NULL;
70      }
71
72      ngx_memcpy(ls->addr_text.data, text, len);
73
74      ls->fd = (ngx_socket_t) -1;
75      ls->type = SOCK_STREAM;
76
77      ls->backlog = NGX_LISTEN_BACKLOG;
78      ls->rcvbuf = -1;
79      ls->sndbuf = -1;
80
81  #if (NGX_HAVE_SETFIB)
82      ls->setfib = -1;
83  #endif
84
85  #if (NGX_HAVE_TCP_FASTOPEN)
86      ls->fastopen = -1;
87  #endif
88
89      return ls;
90  }
91
92
93  ngx_int_t
94  ngx_set_inherited_sockets(ngx_cycle_t *cycle)
95  {
96      size_t                     len;
97      ngx_uint_t                 i;
98      ngx_listening_t           *ls;
99      socklen_t                  olen;
100 #if (NGX_HAVE_DEFERRED_ACCEPT || NGX_HAVE_TCP_FASTOPEN)
101     ngx_err_t                  err;
102 #endif
103 #if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
104     struct accept_filter_arg   af;
105 #endif
106 #if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
107     int                        timeout;
108 #endif
```

```
109
110     ls = cycle->listening.elts;
111     for (i = 0; i < cycle->listening.nelts; i++) {
112
113         ls[i].sockaddr = ngx_palloc(cycle->pool, NGX_SOCKADDRLEN);
114         if (ls[i].sockaddr == NULL) {
115             return NGX_ERROR;
116         }
117
118         ls[i].socklen = NGX_SOCKADDRLEN;
119         if (getsockname(ls[i].fd, ls[i].sockaddr, &ls[i].socklen) == -1) {
120             ngx_log_error(NGX_LOG_CRIT, cycle->log, ngx_socket_errno,
121                           "getsockname() of the inherited "
122                           "socket #%d failed", ls[i].fd);
123             ls[i].ignore = 1;
124             continue;
125         }
126
127         switch (ls[i].sockaddr->sa_family) {
128
129 #if (NGX_HAVE_INET6)
130         case AF_INET6:
131              ls[i].addr_text_max_len = NGX_INET6_ADDRSTRLEN;
132              len = NGX_INET6_ADDRSTRLEN + sizeof("[]:65535") - 1;
133              break;
134 #endif
135
136 #if (NGX_HAVE_UNIX_DOMAIN)
137         case AF_UNIX:
138              ls[i].addr_text_max_len = NGX_UNIX_ADDRSTRLEN;
139              len = NGX_UNIX_ADDRSTRLEN;
140              break;
141 #endif
142
143         case AF_INET:
144              ls[i].addr_text_max_len = NGX_INET_ADDRSTRLEN;
145              len = NGX_INET_ADDRSTRLEN + sizeof(":65535") - 1;
146              break;
147
148         default:
149             ngx_log_error(NGX_LOG_CRIT, cycle->log, ngx_socket_errno,
150                           "the inherited socket #%d has "
151                           "an unsupported protocol family", ls[i].fd);
152             ls[i].ignore = 1;
153             continue;
154         }
155
156         ls[i].addr_text.data = ngx_pnalloc(cycle->pool, len);
157         if (ls[i].addr_text.data == NULL) {
158             return NGX_ERROR;
159         }
160
161         len = ngx_sock_ntop(ls[i].sockaddr, ls[i].socklen,
162                             ls[i].addr_text.data, len, 1);
163         if (len == 0) {
164             return NGX_ERROR;
165         }
166
167         ls[i].addr_text.len = len;
168
169         ls[i].backlog = NGX_LISTEN_BACKLOG;
170
171         olen = sizeof(int);
172
173         if (getsockopt(ls[i].fd, SOL_SOCKET, SO_RCVBUF, (void *) &ls[i].rcvbuf,
174                        &olen)
175             == -1)
176         {
177             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
178                           "getsockopt(SO_RCVBUF) %V failed, ignored",
179                           &ls[i].addr_text);
180
181             ls[i].rcvbuf = -1;
182         }
183
184         olen = sizeof(int);
```

```
185
186         if (getsockopt(ls[i].fd, SOL_SOCKET, SO_SNDBUF, (void *) &ls[i].sndbuf,
187                        &olen)
188             == -1)
189         {
190             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
191                           "getsockopt(SO_SNDBUF) %V failed, ignored",
192                           &ls[i].addr_text);
193
194             ls[i].sndbuf = -1;
195         }
196
197 #if 0
198         /* SO_SETFIB is currently a set only option */
199
200 #if (NGX_HAVE_SETFIB)
201
202         olen = sizeof(int);
203
204         if (getsockopt(ls[i].fd, SOL_SOCKET, SO_SETFIB,
205                        (void *) &ls[i].setfib, &olen)
206             == -1)
207         {
208             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
209                           "getsockopt(SO_SETFIB) %V failed, ignored",
210                           &ls[i].addr_text);
211
212             ls[i].setfib = -1;
213         }
214
215 #endif
216 #endif
217
218 #if (NGX_HAVE_TCP_FASTOPEN)
219
220         olen = sizeof(int);
221
222         if (getsockopt(ls[i].fd, IPPROTO_TCP, TCP_FASTOPEN,
223                        (void *) &ls[i].fastopen, &olen)
224             == -1)
225         {
226             err = ngx_socket_errno;
227
228             if (err != NGX_EOPNOTSUPP && err != NGX_ENOPROTOOPT) {
229                 ngx_log_error(NGX_LOG_NOTICE, cycle->log, err,
230                               "getsockopt(TCP_FASTOPEN) %V failed, ignored",
231                               &ls[i].addr_text);
232             }
233
234             ls[i].fastopen = -1;
235         }
236
237 #endif
238
239 #if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
240
241         ngx_memzero(&af, sizeof(struct accept_filter_arg));
242         olen = sizeof(struct accept_filter_arg);
243
244         if (getsockopt(ls[i].fd, SOL_SOCKET, SO_ACCEPTFILTER, &af, &olen)
245             == -1)
246         {
247             err = ngx_socket_errno;
248
249             if (err == NGX_EINVAL) {
250                 continue;
251             }
252
253             ngx_log_error(NGX_LOG_NOTICE, cycle->log, err,
254                           "getsockopt(SO_ACCEPTFILTER) for %V failed, ignored",
255                           &ls[i].addr_text);
256             continue;
257         }
258
259         if (olen < sizeof(struct accept_filter_arg) || af.af_name[0] == '\0') {
260             continue;
```

```
261         }
262
263         ls[i].accept_filter = ngx_palloc(cycle->pool, 16);
264         if (ls[i].accept_filter == NULL) {
265             return NGX_ERROR;
266         }
267
268         (void) ngx_cpystrn((u_char *) ls[i].accept_filter,
269                            (u_char *) af.af_name, 16);
270 #endif
271
272 #if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
273
274         timeout = 0;
275         olen = sizeof(int);
276
277         if (getsockopt(ls[i].fd, IPPROTO_TCP, TCP_DEFER_ACCEPT, &timeout, &olen)
278             == -1)
279         {
280             err = ngx_socket_errno;
281
282             if (err == NGX_EOPNOTSUPP) {
283                 continue;
284             }
285
286             ngx_log_error(NGX_LOG_NOTICE, cycle->log, err,
287                           "getsockopt(TCP_DEFER_ACCEPT) for %V failed, ignored",
288                           &ls[i].addr_text);
289             continue;
290         }
291
292         if (olen < sizeof(int) || timeout == 0) {
293             continue;
294         }
295
296         ls[i].deferred_accept = 1;
297 #endif
298     }
299
300     return NGX_OK;
301 }
302
303
304 ngx_int_t
305 ngx_open_listening_sockets(ngx_cycle_t *cycle)
306 {
307     int               reuseaddr;
308     ngx_uint_t        i, tries, failed;
309     ngx_err_t         err;
310     ngx_log_t        *log;
311     ngx_socket_t      s;
312     ngx_listening_t  *ls;
313
314     reuseaddr = 1;
315 #if (NGX_SUPPRESS_WARN)
316     failed = 0;
317 #endif
318
319     log = cycle->log;
320
321     /* TODO: configurable try number */
322
323     for (tries = 5; tries; tries--) {
324         failed = 0;
325
326         /* for each listening socket */
327
328         ls = cycle->listening.elts;
329         for (i = 0; i < cycle->listening.nelts; i++) {
330
331             if (ls[i].ignore) {
332                 continue;
333             }
334
335             if (ls[i].fd != (ngx_socket_t) -1) {
336                 continue;
```

```
337             }
338
339             if (ls[i].inherited) {
340
341                 /* TODO: close on exit */
342                 /* TODO: nonblocking */
343                 /* TODO: deferred accept */
344
345                 continue;
346             }
347
348             s = ngx_socket(ls[i].sockaddr->sa_family, ls[i].type, 0);
349
350             if (s == (ngx_socket_t) -1) {
351                 ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
352                               ngx_socket_n " %V failed", &ls[i].addr_text);
353                 return NGX_ERROR;
354             }
355
356             if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR,
357                            (const void *) &reuseaddr, sizeof(int))
358                 == -1)
359             {
360                 ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
361                               "setsockopt(SO_REUSEADDR) %V failed",
362                               &ls[i].addr_text);
363
364                 if (ngx_close_socket(s) == -1) {
365                     ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
366                                   ngx_close_socket_n " %V failed",
367                                   &ls[i].addr_text);
368                 }
369
370                 return NGX_ERROR;
371             }
372
373 #if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
374
375             if (ls[i].sockaddr->sa_family == AF_INET6) {
376                 int  ipv6only;
377
378                 ipv6only = ls[i].ipv6only;
379
380                 if (setsockopt(s, IPPROTO_IPV6, IPV6_V6ONLY,
381                                (const void *) &ipv6only, sizeof(int))
382                     == -1)
383                 {
384                     ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
385                                   "setsockopt(IPV6_V6ONLY) %V failed, ignored",
386                                   &ls[i].addr_text);
387                 }
388             }
389 #endif
390             /* TODO: close on exit */
391
392             if (!(ngx_event_flags & NGX_USE_AIO_EVENT)) {
393                 if (ngx_nonblocking(s) == -1) {
394                     ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
395                                   ngx_nonblocking_n " %V failed",
396                                   &ls[i].addr_text);
397
398                     if (ngx_close_socket(s) == -1) {
399                         ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
400                                       ngx_close_socket_n " %V failed",
401                                       &ls[i].addr_text);
402                     }
403
404                     return NGX_ERROR;
405                 }
406             }
407
408             ngx_log_debug2(NGX_LOG_DEBUG_CORE, log, 0,
409                            "bind() %V #%d ", &ls[i].addr_text, s);
410
411             if (bind(s, ls[i].sockaddr, ls[i].socklen) == -1) {
412                 err = ngx_socket_errno;
```

```
413
414                 if (err != NGX_EADDRINUSE || !ngx_test_config) {
415                     ngx_log_error(NGX_LOG_EMERG, log, err,
416                                   "bind() to %V failed", &ls[i].addr_text);
417                 }
418
419                 if (ngx_close_socket(s) == -1) {
420                     ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
421                                   ngx_close_socket_n " %V failed",
422                                   &ls[i].addr_text);
423                 }
424
425                 if (err != NGX_EADDRINUSE) {
426                     return NGX_ERROR;
427                 }
428
429                 if (!ngx_test_config) {
430                     failed = 1;
431                 }
432
433                 continue;
434             }
435
436 #if (NGX_HAVE_UNIX_DOMAIN)
437
438             if (ls[i].sockaddr->sa_family == AF_UNIX) {
439                 mode_t   mode;
440                 u_char  *name;
441
442                 name = ls[i].addr_text.data + sizeof("unix:") - 1;
443                 mode = (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
444
445                 if (chmod((char *) name, mode) == -1) {
446                     ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
447                                   "chmod() \"%s\" failed", name);
448                 }
449
450                 if (ngx_test_config) {
451                     if (ngx_delete_file(name) == NGX_FILE_ERROR) {
452                         ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
453                                       ngx_delete_file_n " %s failed", name);
454                     }
455                 }
456             }
457 #endif
458
459             if (listen(s, ls[i].backlog) == -1) {
460                 ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
461                               "listen() to %V, backlog %d failed",
462                               &ls[i].addr_text, ls[i].backlog);
463
464                 if (ngx_close_socket(s) == -1) {
465                     ngx_log_error(NGX_LOG_EMERG, log, ngx_socket_errno,
466                                   ngx_close_socket_n " %V failed",
467                                   &ls[i].addr_text);
468                 }
469
470                 return NGX_ERROR;
471             }
472
473             ls[i].listen = 1;
474
475             ls[i].fd = s;
476         }
477
478         if (!failed) {
479             break;
480         }
481
482         /* TODO: delay configurable */
483
484         ngx_log_error(NGX_LOG_NOTICE, log, 0,
485                       "try again to bind() after 500ms");
486
487         ngx_msleep(500);
488     }
```

```
489
490     if (failed) {
491         ngx_log_error(NGX_LOG_EMERG, log, 0, "still could not bind()");
492         return NGX_ERROR;
493     }
494
495     return NGX_OK;
496 }
497
498
499 void
500 ngx_configure_listening_sockets(ngx_cycle_t *cycle)
501 {
502     int                      value;
503     ngx_uint_t               i;
504     ngx_listening_t         *ls;
505
506 #if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
507     struct accept_filter_arg   af;
508 #endif
509
510     ls = cycle->listening.elts;
511     for (i = 0; i < cycle->listening.nelts; i++) {
512
513         ls[i].log = *ls[i].logp;
514
515         if (ls[i].rcvbuf != -1) {
516             if (setsockopt(ls[i].fd, SOL_SOCKET, SO_RCVBUF,
517                            (const void *) &ls[i].rcvbuf, sizeof(int))
518                 == -1)
519             {
520                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
521                               "setsockopt(SO_RCVBUF, %d) %V failed, ignored",
522                               ls[i].rcvbuf, &ls[i].addr_text);
523             }
524         }
525
526         if (ls[i].sndbuf != -1) {
527             if (setsockopt(ls[i].fd, SOL_SOCKET, SO_SNDBUF,
528                            (const void *) &ls[i].sndbuf, sizeof(int))
529                 == -1)
530             {
531                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
532                               "setsockopt(SO_SNDBUF, %d) %V failed, ignored",
533                               ls[i].sndbuf, &ls[i].addr_text);
534             }
535         }
536
537         if (ls[i].keepalive) {
538             value = (ls[i].keepalive == 1) ? 1 : 0;
539
540             if (setsockopt(ls[i].fd, SOL_SOCKET, SO_KEEPALIVE,
541                            (const void *) &value, sizeof(int))
542                 == -1)
543             {
544                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
545                               "setsockopt(SO_KEEPALIVE, %d) %V failed, ignored",
546                               value, &ls[i].addr_text);
547             }
548         }
549
550 #if (NGX_HAVE_KEEPALIVE_TUNABLE)
551
552         if (ls[i].keepidle) {
553             value = ls[i].keepidle;
554
555 #if (NGX_KEEPALIVE_FACTOR)
556             value *= NGX_KEEPALIVE_FACTOR;
557 #endif
558
559             if (setsockopt(ls[i].fd, IPPROTO_TCP, TCP_KEEPIDLE,
560                            (const void *) &value, sizeof(int))
561                 == -1)
562             {
563                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
564                               "setsockopt(TCP_KEEPIDLE, %d) %V failed, ignored",
```

```
                                value, &ls[i].addr_text);
            }
        }

        if (ls[i].keepintvl) {
            value = ls[i].keepintvl;

#if (NGX_KEEPALIVE_FACTOR)
            value *= NGX_KEEPALIVE_FACTOR;
#endif

            if (setsockopt(ls[i].fd, IPPROTO_TCP, TCP_KEEPINTVL,
                           (const void *) &value, sizeof(int))
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
                              "setsockopt(TCP_KEEPINTVL, %d) %V failed, ignored",
                              value, &ls[i].addr_text);
            }
        }

        if (ls[i].keepcnt) {
            if (setsockopt(ls[i].fd, IPPROTO_TCP, TCP_KEEPCNT,
                           (const void *) &ls[i].keepcnt, sizeof(int))
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
                              "setsockopt(TCP_KEEPCNT, %d) %V failed, ignored",
                              ls[i].keepcnt, &ls[i].addr_text);
            }
        }

#endif

#if (NGX_HAVE_SETFIB)
        if (ls[i].setfib != -1) {
            if (setsockopt(ls[i].fd, SOL_SOCKET, SO_SETFIB,
                           (const void *) &ls[i].setfib, sizeof(int))
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
                              "setsockopt(SO_SETFIB, %d) %V failed, ignored",
                              ls[i].setfib, &ls[i].addr_text);
            }
        }
#endif

#if (NGX_HAVE_TCP_FASTOPEN)
        if (ls[i].fastopen != -1) {
            if (setsockopt(ls[i].fd, IPPROTO_TCP, TCP_FASTOPEN,
                           (const void *) &ls[i].fastopen, sizeof(int))
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
                              "setsockopt(TCP_FASTOPEN, %d) %V failed, ignored",
                              ls[i].fastopen, &ls[i].addr_text);
            }
        }
#endif

#if 0
        if (1) {
            int tcp_nodelay = 1;

            if (setsockopt(ls[i].fd, IPPROTO_TCP, TCP_NODELAY,
                           (const void *) &tcp_nodelay, sizeof(int))
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
                              "setsockopt(TCP_NODELAY) %V failed, ignored",
                              &ls[i].addr_text);
            }
        }
#endif

        if (ls[i].listen) {
```

```
641
642             /* change backlog via listen() */
643
644             if (listen(ls[i].fd, ls[i].backlog) == -1) {
645                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
646                               "listen() to %V, backlog %d failed, ignored",
647                               &ls[i].addr_text, ls[i].backlog);
648             }
649         }
650
651         /*
652          * setting deferred mode should be last operation on socket,
653          * because code may prematurely continue cycle on failure
654          */
655
656 #if (NGX_HAVE_DEFERRED_ACCEPT)
657
658 #ifdef SO_ACCEPTFILTER
659
660         if (ls[i].delete_deferred) {
661             if (setsockopt(ls[i].fd, SOL_SOCKET, SO_ACCEPTFILTER, NULL, 0)
662                 == -1)
663             {
664                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
665                               "setsockopt(SO_ACCEPTFILTER, NULL) "
666                               "for %V failed, ignored",
667                               &ls[i].addr_text);
668
669                 if (ls[i].accept_filter) {
670                     ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
671                                   "could not change the accept filter "
672                                   "to \"%s\" for %V, ignored",
673                                   ls[i].accept_filter, &ls[i].addr_text);
674                 }
675
676                 continue;
677             }
678
679             ls[i].deferred_accept = 0;
680         }
681
682         if (ls[i].add_deferred) {
683             ngx_memzero(&af, sizeof(struct accept_filter_arg));
684             (void) ngx_cpystrn((u_char *) af.af_name,
685                                (u_char *) ls[i].accept_filter, 16);
686
687             if (setsockopt(ls[i].fd, SOL_SOCKET, SO_ACCEPTFILTER,
688                            &af, sizeof(struct accept_filter_arg))
689                 == -1)
690             {
691                 ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
692                               "setsockopt(SO_ACCEPTFILTER, \"%s\") "
693                               "for %V failed, ignored",
694                               ls[i].accept_filter, &ls[i].addr_text);
695                 continue;
696             }
697
698             ls[i].deferred_accept = 1;
699         }
700
701 #endif
702
703 #ifdef TCP_DEFER_ACCEPT
704
705         if (ls[i].add_deferred || ls[i].delete_deferred) {
706
707             if (ls[i].add_deferred) {
708                 /*
709                  * There is no way to find out how long a connection was
710                  * in queue (and a connection may bypass deferred queue at all
711                  * if syncookies were used), hence we use 1 second timeout
712                  * here.
713                  */
714                 value = 1;
715
716             } else {
```

```
                value = 0;
            }

            if (setsockopt(ls[i].fd, IPPROTO_TCP, TCP_DEFER_ACCEPT,
                           &value, sizeof(int))
                == -1)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_socket_errno,
                              "setsockopt(TCP_DEFER_ACCEPT, %d) for %V failed, "
                              "ignored",
                              value, &ls[i].addr_text);

                continue;
            }
        }

        if (ls[i].add_deferred) {
            ls[i].deferred_accept = 1;
        }

#endif

#endif /* NGX_HAVE_DEFERRED_ACCEPT */
    }

    return;
}


void
ngx_close_listening_sockets(ngx_cycle_t *cycle)
{
    ngx_uint_t         i;
    ngx_listening_t   *ls;
    ngx_connection_t  *c;

    if (ngx_event_flags & NGX_USE_IOCP_EVENT) {
        return;
    }

    ngx_accept_mutex_held = 0;
    ngx_use_accept_mutex = 0;

    ls = cycle->listening.elts;
    for (i = 0; i < cycle->listening.nelts; i++) {

        c = ls[i].connection;

        if (c) {
            if (c->read->active) {
                if (ngx_event_flags & NGX_USE_RTSIG_EVENT) {
                    ngx_del_conn(c, NGX_CLOSE_EVENT);

                } else if (ngx_event_flags & NGX_USE_EPOLL_EVENT) {

                    /*
                     * it seems that Linux-2.6.x OpenVZ sends events
                     * for closed shared listening sockets unless
                     * the events was explicitly deleted
                     */

                    ngx_del_event(c->read, NGX_READ_EVENT, 0);

                } else {
                    ngx_del_event(c->read, NGX_READ_EVENT, NGX_CLOSE_EVENT);
                }
            }

            ngx_free_connection(c);

            c->fd = (ngx_socket_t) -1;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                       "close listening %V #%d ", &ls[i].addr_text, ls[i].fd);

```

```
        if (ngx_close_socket(ls[i].fd) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_socket_errno,
                          ngx_close_socket_n " %V failed", &ls[i].addr_text);
        }

#if (NGX_HAVE_UNIX_DOMAIN)

        if (ls[i].sockaddr->sa_family == AF_UNIX
            && ngx_process <= NGX_PROCESS_MASTER
            && ngx_new_binary == 0)
        {
            u_char *name = ls[i].addr_text.data + sizeof("unix:") - 1;

            if (ngx_delete_file(name) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_socket_errno,
                              ngx_delete_file_n " %s failed", name);
            }
        }

#endif

        ls[i].fd = (ngx_socket_t) -1;
    }

    cycle->listening.nelts = 0;
}


ngx_connection_t *
ngx_get_connection(ngx_socket_t s, ngx_log_t *log)
{
    ngx_uint_t         instance;
    ngx_event_t       *rev, *wev;
    ngx_connection_t  *c;

    /* disable warning: Win32 SOCKET is u_int while UNIX socket is int */

    if (ngx_cycle->files && (ngx_uint_t) s >= ngx_cycle->files_n) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "the new socket has number %d, "
                      "but only %ui files are available",
                      s, ngx_cycle->files_n);
        return NULL;
    }

    /* ngx_mutex_lock */

    c = ngx_cycle->free_connections;

    if (c == NULL) {
        ngx_drain_connections();
        c = ngx_cycle->free_connections;
    }

    if (c == NULL) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "%ui worker_connections are not enough",
                      ngx_cycle->connection_n);

        /* ngx_mutex_unlock */

        return NULL;
    }

    ngx_cycle->free_connections = c->data;
    ngx_cycle->free_connection_n--;

    /* ngx_mutex_unlock */

    if (ngx_cycle->files) {
        ngx_cycle->files[s] = c;
    }

    rev = c->read;
    wev = c->write;

```

```
    ngx_memzero(c, sizeof(ngx_connection_t));

    c->read = rev;
    c->write = wev;
    c->fd = s;
    c->log = log;

    instance = rev->instance;

    ngx_memzero(rev, sizeof(ngx_event_t));
    ngx_memzero(wev, sizeof(ngx_event_t));

    rev->instance = !instance;
    wev->instance = !instance;

    rev->index = NGX_INVALID_INDEX;
    wev->index = NGX_INVALID_INDEX;

    rev->data = c;
    wev->data = c;

    wev->write = 1;

    return c;
}


void
ngx_free_connection(ngx_connection_t *c)
{
    /* ngx_mutex_lock */

    c->data = ngx_cycle->free_connections;
    ngx_cycle->free_connections = c;
    ngx_cycle->free_connection_n++;

    /* ngx_mutex_unlock */

    if (ngx_cycle->files) {
        ngx_cycle->files[c->fd] = NULL;
    }
}


void
ngx_close_connection(ngx_connection_t *c)
{
    ngx_err_t     err;
    ngx_uint_t    log_error, level;
    ngx_socket_t  fd;

    if (c->fd == (ngx_socket_t) -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, 0, "connection already closed");
        return;
    }

    if (c->read->timer_set) {
        ngx_del_timer(c->read);
    }

    if (c->write->timer_set) {
        ngx_del_timer(c->write);
    }

    if (ngx_del_conn) {
        ngx_del_conn(c, NGX_CLOSE_EVENT);

    } else {
        if (c->read->active || c->read->disabled) {
            ngx_del_event(c->read, NGX_READ_EVENT, NGX_CLOSE_EVENT);
        }

        if (c->write->active || c->write->disabled) {
            ngx_del_event(c->write, NGX_WRITE_EVENT, NGX_CLOSE_EVENT);
        }
    }
```

```
945
946 #if (NGX_THREADS)
947
948     /*
949      * we have to clean the connection information before the closing
950      * because another thread may reopen the same file descriptor
951      * before we clean the connection
952      */
953
954     ngx_unlock(&c->lock);
955
956 #endif
957
958     if (c->read->posted) {
959         ngx_delete_posted_event(c->read);
960     }
961
962     if (c->write->posted) {
963         ngx_delete_posted_event(c->write);
964     }
965
966     c->read->closed = 1;
967     c->write->closed = 1;
968
969     ngx_reusable_connection(c, 0);
970
971     log_error = c->log_error;
972
973     ngx_free_connection(c);
974
975     fd = c->fd;
976     c->fd = (ngx_socket_t) -1;
977
978     if (ngx_close_socket(fd) == -1) {
979
980         err = ngx_socket_errno;
981
982         if (err == NGX_ECONNRESET || err == NGX_ENOTCONN) {
983
984             switch (log_error) {
985
986             case NGX_ERROR_INFO:
987                 level = NGX_LOG_INFO;
988                 break;
989
990             case NGX_ERROR_ERR:
991                 level = NGX_LOG_ERR;
992                 break;
993
994             default:
995                 level = NGX_LOG_CRIT;
996             }
997
998         } else {
999             level = NGX_LOG_CRIT;
1000         }
1001
1002         /* we use ngx_cycle->log because c->log was in c->pool */
1003
1004         ngx_log_error(level, ngx_cycle->log, err,
1005                       ngx_close_socket_n " %d failed", fd);
1006     }
1007 }
1008
1009
1010 void
1011 ngx_reusable_connection(ngx_connection_t *c, ngx_uint_t reusable)
1012 {
1013     ngx_log_debug1(NGX_LOG_DEBUG_CORE, c->log, 0,
1014                    "reusable connection: %ui", reusable);
1015
1016     if (c->reusable) {
1017         ngx_queue_remove(&c->queue);
1018
1019 #if (NGX_STAT_STUB)
1020         (void) ngx_atomic_fetch_add(ngx_stat_waiting, -1);
```

```
#endif
    }

    c->reusable = reusable;

    if (reusable) {
        /* need cast as ngx_cycle is volatile */

        ngx_queue_insert_head(
            (ngx_queue_t *) &ngx_cycle->reusable_connections_queue, &c->queue);

#if (NGX_STAT_STUB)
        (void) ngx_atomic_fetch_add(ngx_stat_waiting, 1);
#endif
    }
}


static void
ngx_drain_connections(void)
{
    ngx_int_t          i;
    ngx_queue_t       *q;
    ngx_connection_t  *c;

    for (i = 0; i < 32; i++) {
        if (ngx_queue_empty(&ngx_cycle->reusable_connections_queue)) {
            break;
        }

        q = ngx_queue_last(&ngx_cycle->reusable_connections_queue);
        c = ngx_queue_data(q, ngx_connection_t, queue);

        ngx_log_debug0(NGX_LOG_DEBUG_CORE, c->log, 0,
                       "reusing connection");

        c->close = 1;
        c->read->handler(c->read);
    }
}


ngx_int_t
ngx_connection_local_sockaddr(ngx_connection_t *c, ngx_str_t *s,
    ngx_uint_t port)
{
    socklen_t             len;
    ngx_uint_t            addr;
    u_char                sa[NGX_SOCKADDRLEN];
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    ngx_uint_t            i;
    struct sockaddr_in6  *sin6;
#endif

    if (c->local_socklen == 0) {
        return NGX_ERROR;
    }

    switch (c->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) c->local_sockaddr;

        for (addr = 0, i = 0; addr == 0 && i < 16; i++) {
            addr |= sin6->sin6_addr.s6_addr[i];
        }

        break;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
    case AF_UNIX:
        addr = 1;
        break;
```

```
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) c->local_sockaddr;
        addr = sin->sin_addr.s_addr;
        break;
    }

    if (addr == 0) {

        len = NGX_SOCKADDRLEN;

        if (getsockname(c->fd, (struct sockaddr *) &sa, &len) == -1) {
            ngx_connection_error(c, ngx_socket_errno, "getsockname() failed");
            return NGX_ERROR;
        }

        c->local_sockaddr = ngx_palloc(c->pool, len);
        if (c->local_sockaddr == NULL) {
            return NGX_ERROR;
        }

        ngx_memcpy(c->local_sockaddr, &sa, len);

        c->local_socklen = len;
    }

    if (s == NULL) {
        return NGX_OK;
    }

    s->len = ngx_sock_ntop(c->local_sockaddr, c->local_socklen,
                           s->data, s->len, port);

    return NGX_OK;
}


ngx_int_t
ngx_connection_error(ngx_connection_t *c, ngx_err_t err, char *text)
{
    ngx_uint_t  level;

    /* Winsock may return NGX_ECONNABORTED instead of NGX_ECONNRESET */

    if ((err == NGX_ECONNRESET
#if (NGX_WIN32)
         || err == NGX_ECONNABORTED
#endif
        ) && c->log_error == NGX_ERROR_IGNORE_ECONNRESET)
    {
        return 0;
    }

#if (NGX_SOLARIS)
    if (err == NGX_EINVAL && c->log_error == NGX_ERROR_IGNORE_EINVAL) {
        return 0;
    }
#endif

    if (err == 0
        || err == NGX_ECONNRESET
#if (NGX_WIN32)
        || err == NGX_ECONNABORTED
#else
        || err == NGX_EPIPE
#endif
        || err == NGX_ENOTCONN
        || err == NGX_ETIMEDOUT
        || err == NGX_ECONNREFUSED
        || err == NGX_ENETDOWN
        || err == NGX_ENETUNREACH
        || err == NGX_EHOSTDOWN
        || err == NGX_EHOSTUNREACH)
    {
        switch (c->log_error) {
```

```
1173
1174            case NGX_ERROR_IGNORE_EINVAL:
1175            case NGX_ERROR_IGNORE_ECONNRESET:
1176            case NGX_ERROR_INFO:
1177                level = NGX_LOG_INFO;
1178                break;
1179
1180            default:
1181                level = NGX_LOG_ERR;
1182            }
1183
1184        } else {
1185            level = NGX_LOG_ALERT;
1186        }
1187
1188        ngx_log_error(level, c->log, err, text);
1189
1190        return NGX_ERROR;
1191    }
```

One Level Up Top Level

src/core/ngx_connection.h - nginx-1.7.10

Data types defined

- ngx_connection_log_error_e
- ngx_connection_s
- ngx_connection_tcp_nodelay_e
- ngx_connection_tcp_nopush_e
- ngx_listening_s
- ngx_listening_t

Macros defined

- NGX_LOWLEVEL_BUFFERED
- NGX_SPDY_BUFFERED
- NGX_SSL_BUFFERED
- _NGX_CONNECTION_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_CONNECTION_H_INCLUDED_
9  #define _NGX_CONNECTION_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct ngx_listening_s  ngx_listening_t;
17
18  struct ngx_listening_s {
19      ngx_socket_t        fd;
20
21      struct sockaddr    *sockaddr;
22      socklen_t           socklen;    /* size of sockaddr */
23      size_t              addr_text_max_len;
24      ngx_str_t           addr_text;
25
26      int                 type;
27
28      int                 backlog;
29      int                 rcvbuf;
30      int                 sndbuf;
31  #if (NGX_HAVE_KEEPALIVE_TUNABLE)
32      int                 keepidle;
33      int                 keepintvl;
34      int                 keepcnt;
35  #endif
36
37      /* handler of accepted connection */
38      ngx_connection_handler_pt   handler;
```

```
    void               *servers;  /* array of ngx_http_in_addr_t, for example */

    ngx_log_t           log;
    ngx_log_t          *logp;

    size_t              pool_size;
    /* should be here because of the AcceptEx() preread */
    size_t              post_accept_buffer_size;
    /* should be here because of the deferred accept */
    ngx_msec_t          post_accept_timeout;

    ngx_listening_t    *previous;
    ngx_connection_t   *connection;

    unsigned            open:1;
    unsigned            remain:1;
    unsigned            ignore:1;

    unsigned            bound:1;       /* already bound */
    unsigned            inherited:1;   /* inherited from previous process */
    unsigned            nonblocking_accept:1;
    unsigned            listen:1;
    unsigned            nonblocking:1;
    unsigned            shared:1;    /* shared between threads or processes */
    unsigned            addr_ntop:1;

#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    unsigned            ipv6only:1;
#endif
    unsigned            keepalive:2;

#if (NGX_HAVE_DEFERRED_ACCEPT)
    unsigned            deferred_accept:1;
    unsigned            delete_deferred:1;
    unsigned            add_deferred:1;
#ifdef SO_ACCEPTFILTER
    char               *accept_filter;
#endif
#endif
#if (NGX_HAVE_SETFIB)
    int                 setfib;
#endif

#if (NGX_HAVE_TCP_FASTOPEN)
    int                 fastopen;
#endif

};


typedef enum {
     NGX_ERROR_ALERT = 0,
     NGX_ERROR_ERR,
     NGX_ERROR_INFO,
     NGX_ERROR_IGNORE_ECONNRESET,
     NGX_ERROR_IGNORE_EINVAL
} ngx_connection_log_error_e;


typedef enum {
      NGX_TCP_NODELAY_UNSET = 0,
      NGX_TCP_NODELAY_SET,
      NGX_TCP_NODELAY_DISABLED
 } ngx_connection_tcp_nodelay_e;


 typedef enum {
      NGX_TCP_NOPUSH_UNSET = 0,
      NGX_TCP_NOPUSH_SET,
      NGX_TCP_NOPUSH_DISABLED
 } ngx_connection_tcp_nopush_e;


 #define NGX_LOWLEVEL_BUFFERED  0x0f
 #define NGX_SSL_BUFFERED       0x01
```

```
#define NGX_SPDY_BUFFERED      0x02


struct ngx_connection_s {
    void               *data;
    ngx_event_t        *read;
    ngx_event_t        *write;

    ngx_socket_t        fd;

    ngx_recv_pt         recv;
    ngx_send_pt         send;
    ngx_recv_chain_pt   recv_chain;
    ngx_send_chain_pt   send_chain;

    ngx_listening_t    *listening;

    off_t               sent;

    ngx_log_t          *log;

    ngx_pool_t         *pool;

    struct sockaddr    *sockaddr;
    socklen_t           socklen;
    ngx_str_t           addr_text;

    ngx_str_t           proxy_protocol_addr;

#if (NGX_SSL)
    ngx_ssl_connection_t  *ssl;
#endif

    struct sockaddr    *local_sockaddr;
    socklen_t           local_socklen;

    ngx_buf_t          *buffer;

    ngx_queue_t         queue;

    ngx_atomic_uint_t   number;

    ngx_uint_t          requests;

    unsigned            buffered:8;

    unsigned            log_error:3;     /* ngx_connection_log_error_e */

    unsigned            unexpected_eof:1;
    unsigned            timedout:1;
    unsigned            error:1;
    unsigned            destroyed:1;

    unsigned            idle:1;
    unsigned            reusable:1;
    unsigned            close:1;

    unsigned            sendfile:1;
    unsigned            sndlowat:1;
    unsigned            tcp_nodelay:2;   /* ngx_connection_tcp_nodelay_e */
    unsigned            tcp_nopush:2;    /* ngx_connection_tcp_nopush_e */

    unsigned            need_last_buf:1;

#if (NGX_HAVE_IOCP)
    unsigned            accept_context_updated:1;
#endif

#if (NGX_HAVE_AIO_SENDFILE)
    unsigned            aio_sendfile:1;
    unsigned            busy_count:2;
    ngx_buf_t          *busy_sendfile;
#endif

#if (NGX_THREADS)
    ngx_atomic_t        lock;
```

```
#endif
};


ngx_listening_t *ngx_create_listening(ngx_conf_t *cf, void *sockaddr,
    socklen_t socklen);
ngx_int_t ngx_set_inherited_sockets(ngx_cycle_t *cycle);
ngx_int_t ngx_open_listening_sockets(ngx_cycle_t *cycle);
void ngx_configure_listening_sockets(ngx_cycle_t *cycle);
void ngx_close_listening_sockets(ngx_cycle_t *cycle);
void ngx_close_connection(ngx_connection_t *c);
ngx_int_t ngx_connection_local_sockaddr(ngx_connection_t *c, ngx_str_t *s,
    ngx_uint_t port);
ngx_int_t ngx_connection_error(ngx_connection_t *c, ngx_err_t err, char *text);

ngx_connection_t *ngx_get_connection(ngx_socket_t s, ngx_log_t *log);
void ngx_free_connection(ngx_connection_t *c);

void ngx_reusable_connection(ngx_connection_t *c, ngx_uint_t reusable);

#endif /* _NGX_CONNECTION_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_socket.h - nginx-1.7.10

Data types defined

- ngx_socket_t

Macros defined

- NGX_WRITE_SHUTDOWN
- _NGX_SOCKET_H_INCLUDED_
- ngx_blocking
- ngx_blocking_n
- ngx_blocking_n
- ngx_close_socket
- ngx_close_socket_n
- ngx_nonblocking
- ngx_nonblocking_n
- ngx_nonblocking_n
- ngx_shutdown_socket
- ngx_shutdown_socket_n
- ngx_socket
- ngx_socket_n
- ngx_tcp_nopush_n
- ngx_tcp_nopush_n
- ngx_tcp_push_n
- ngx_tcp_push_n

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_SOCKET_H_INCLUDED_
9  #define _NGX_SOCKET_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13
14
15  #define NGX_WRITE_SHUTDOWN SHUT_WR
16
17  typedef int  ngx_socket_t;
18
```

```
19 #define ngx_socket          socket
20 #define ngx_socket_n        "socket()"
21
22
23 #if (NGX_HAVE_FIONBIO)
24
25 int ngx_nonblocking(ngx_socket_t s);
26 int ngx_blocking(ngx_socket_t s);
27
28 #define ngx_nonblocking_n   "ioctl(FIONBIO)"
29 #define ngx_blocking_n      "ioctl(!FIONBIO)"
30
31 #else
32
33 #define ngx_nonblocking(s)  fcntl(s, F_SETFL, fcntl(s, F_GETFL) | O_NONBLOCK)
34 #define ngx_nonblocking_n   "fcntl(O_NONBLOCK)"
35
36 #define ngx_blocking(s)     fcntl(s, F_SETFL, fcntl(s, F_GETFL) & ~O_NONBLOCK)
37 #define ngx_blocking_n      "fcntl(!O_NONBLOCK)"
38
39 #endif
40
41 int ngx_tcp_nopush(ngx_socket_t s);
42 int ngx_tcp_push(ngx_socket_t s);
43
44 #if (NGX_LINUX)
45
46 #define ngx_tcp_nopush_n   "setsockopt(TCP_CORK)"
47 #define ngx_tcp_push_n     "setsockopt(!TCP_CORK)"
48
49 #else
50
51 #define ngx_tcp_nopush_n   "setsockopt(TCP_NOPUSH)"
52 #define ngx_tcp_push_n     "setsockopt(!TCP_NOPUSH)"
53
54 #endif
55
56
57 #define ngx_shutdown_socket    shutdown
58 #define ngx_shutdown_socket_n  "shutdown()"
59
60 #define ngx_close_socket    close
61 #define ngx_close_socket_n  "close() socket"
62
63
64 #endif /* _NGX_SOCKET_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_socket.c - nginx-1.7.10

Functions defined

- ngx_blocking
- ngx_nonblocking
- ngx_tcp_nopush
- ngx_tcp_nopush
- ngx_tcp_nopush
- ngx_tcp_push
- ngx_tcp_push
- ngx_tcp_push

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13   * ioctl(FIONBIO) sets a non-blocking mode with the single syscall
14   * while fcntl(F_SETFL, O_NONBLOCK) needs to learn the current state
15   * using fcntl(F_GETFL).
16   *
17   * ioctl() and fcntl() are syscalls at least in FreeBSD 2.x, Linux 2.2
18   * and Solaris 7.
19   *
20   * ioctl() in Linux 2.4 and 2.6 uses BKL, however, fcntl(F_SETFL) uses it too.
21   */
22
23
24  #if (NGX_HAVE_FIONBIO)
25
26  int
27  ngx_nonblocking(ngx_socket_t s)
28  {
29      int  nb;
30
31      nb = 1;
32
33      return ioctl(s, FIONBIO, &nb);
34  }
35
36
37  int
38  ngx_blocking(ngx_socket_t s)
39  {
40      int  nb;
41
42      nb = 0;
43
44      return ioctl(s, FIONBIO, &nb);
45  }
46
47  #endif
```

```
48
49
50  #if (NGX_FREEBSD)
51
52  int
53  ngx_tcp_nopush(ngx_socket_t s)
54  {
55      int  tcp_nopush;
56
57      tcp_nopush = 1;
58
59      return setsockopt(s, IPPROTO_TCP, TCP_NOPUSH,
60                        (const void *) &tcp_nopush, sizeof(int));
61  }
62
63
64  int
65  ngx_tcp_push(ngx_socket_t s)
66  {
67      int  tcp_nopush;
68
69      tcp_nopush = 0;
70
71      return setsockopt(s, IPPROTO_TCP, TCP_NOPUSH,
72                        (const void *) &tcp_nopush, sizeof(int));
73  }
74
75  #elif (NGX_LINUX)
76
77
78  int
79  ngx_tcp_nopush(ngx_socket_t s)
80  {
81      int  cork;
82
83      cork = 1;
84
85      return setsockopt(s, IPPROTO_TCP, TCP_CORK,
86                        (const void *) &cork, sizeof(int));
87  }
88
89
90  int
91  ngx_tcp_push(ngx_socket_t s)
92  {
93      int  cork;
94
95      cork = 0;
96
97      return setsockopt(s, IPPROTO_TCP, TCP_CORK,
98                        (const void *) &cork, sizeof(int));
99  }
100
101  #else
102
103  int
104  ngx_tcp_nopush(ngx_socket_t s)
105  {
106      return 0;
107  }
108
109
110  int
111  ngx_tcp_push(ngx_socket_t s)
112  {
113      return 0;
114  }
115
116  #endif
```

One Level Up Top Level

src/http/ngx_http_core_module.h - nginx-1.7.10

Data types defined

- ngx_http_addr_conf_s
- ngx_http_conf_addr_t
- ngx_http_conf_port_t
- ngx_http_core_loc_conf_s
- ngx_http_core_loc_conf_t
- ngx_http_core_main_conf_t
- ngx_http_core_srv_conf_t
- ngx_http_err_page_t
- ngx_http_in6_addr_t
- ngx_http_in_addr_t
- ngx_http_listen_opt_t
- ngx_http_location_queue_t
- ngx_http_location_tree_node_s
- ngx_http_location_tree_node_t
- ngx_http_output_body_filter_pt
- ngx_http_output_header_filter_pt
- ngx_http_phase_engine_t
- ngx_http_phase_handler_pt
- ngx_http_phase_handler_s
- ngx_http_phase_handler_t
- ngx_http_phase_t
- ngx_http_phases
- ngx_http_port_t
- ngx_http_server_name_t
- ngx_http_try_file_t
- ngx_http_virtual_names_t

Macros defined

- NGX_HTTP_AIO_OFF
- NGX_HTTP_AIO_ON

- NGX_HTTP_AIO_SENDFILE
- NGX_HTTP_GZIP_PROXIED_ANY
- NGX_HTTP_GZIP_PROXIED_AUTH
- NGX_HTTP_GZIP_PROXIED_EXPIRED
- NGX_HTTP_GZIP_PROXIED_NO_CACHE
- NGX_HTTP_GZIP_PROXIED_NO_ETAG
- NGX_HTTP_GZIP_PROXIED_NO_LM
- NGX_HTTP_GZIP_PROXIED_NO_STORE
- NGX_HTTP_GZIP_PROXIED_OFF
- NGX_HTTP_GZIP_PROXIED_PRIVATE
- NGX_HTTP_IMS_BEFORE
- NGX_HTTP_IMS_EXACT
- NGX_HTTP_IMS_OFF
- NGX_HTTP_KEEPALIVE_DISABLE_MSIE6
- NGX_HTTP_KEEPALIVE_DISABLE_NONE
- NGX_HTTP_KEEPALIVE_DISABLE_SAFARI
- NGX_HTTP_LINGERING_ALWAYS
- NGX_HTTP_LINGERING_OFF
- NGX_HTTP_LINGERING_ON
- NGX_HTTP_SATISFY_ALL
- NGX_HTTP_SATISFY_ANY
- _NGX_HTTP_CORE_H_INCLUDED_
- ngx_http_clear_accept_ranges
- ngx_http_clear_content_length
- ngx_http_clear_etag
- ngx_http_clear_last_modified
- ngx_http_clear_location

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_CORE_H_INCLUDED_
9  #define _NGX_HTTP_CORE_H_INCLUDED_
10
11
```

```
#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_http.h>


#define NGX_HTTP_GZIP_PROXIED_OFF       0x0002
#define NGX_HTTP_GZIP_PROXIED_EXPIRED   0x0004
#define NGX_HTTP_GZIP_PROXIED_NO_CACHE  0x0008
#define NGX_HTTP_GZIP_PROXIED_NO_STORE  0x0010
#define NGX_HTTP_GZIP_PROXIED_PRIVATE   0x0020
#define NGX_HTTP_GZIP_PROXIED_NO_LM     0x0040
#define NGX_HTTP_GZIP_PROXIED_NO_ETAG   0x0080
#define NGX_HTTP_GZIP_PROXIED_AUTH      0x0100
#define NGX_HTTP_GZIP_PROXIED_ANY       0x0200


#define NGX_HTTP_AIO_OFF                0
#define NGX_HTTP_AIO_ON                 1
#define NGX_HTTP_AIO_SENDFILE           2


#define NGX_HTTP_SATISFY_ALL            0
#define NGX_HTTP_SATISFY_ANY            1


#define NGX_HTTP_LINGERING_OFF          0
#define NGX_HTTP_LINGERING_ON           1
#define NGX_HTTP_LINGERING_ALWAYS       2


#define NGX_HTTP_IMS_OFF                0
#define NGX_HTTP_IMS_EXACT              1
#define NGX_HTTP_IMS_BEFORE             2


#define NGX_HTTP_KEEPALIVE_DISABLE_NONE    0x0002
#define NGX_HTTP_KEEPALIVE_DISABLE_MSIE6   0x0004
#define NGX_HTTP_KEEPALIVE_DISABLE_SAFARI  0x0008


typedef struct ngx_http_location_tree_node_s  ngx_http_location_tree_node_t;
typedef struct ngx_http_core_loc_conf_s  ngx_http_core_loc_conf_t;


typedef struct {
    union {
        struct sockaddr        sockaddr;
        struct sockaddr_in     sockaddr_in;
#if (NGX_HAVE_INET6)
        struct sockaddr_in6    sockaddr_in6;
#endif
#if (NGX_HAVE_UNIX_DOMAIN)
        struct sockaddr_un     sockaddr_un;
#endif
        u_char                 sockaddr_data[NGX_SOCKADDRLEN];
    } u;

    socklen_t                  socklen;

    unsigned                   set:1;
    unsigned                   default_server:1;
    unsigned                   bind:1;
    unsigned                   wildcard:1;
#if (NGX_HTTP_SSL)
    unsigned                   ssl:1;
#endif
#if (NGX_HTTP_SPDY)
    unsigned                   spdy:1;
#endif
#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    unsigned                   ipv6only:1;
#endif
    unsigned                   so_keepalive:2;
    unsigned                   proxy_protocol:1;

    int                        backlog;
```

```
    int                        rcvbuf;
    int                        sndbuf;
#if (NGX_HAVE_SETFIB)
    int                        setfib;
#endif
#if (NGX_HAVE_TCP_FASTOPEN)
    int                        fastopen;
#endif
#if (NGX_HAVE_KEEPALIVE_TUNABLE)
    int                        tcp_keepidle;
    int                        tcp_keepintvl;
    int                        tcp_keepcnt;
#endif

#if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
    char                       *accept_filter;
#endif
#if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
    ngx_uint_t                 deferred_accept;
#endif

    u_char                     addr[NGX_SOCKADDR_STRLEN + 1];
} ngx_http_listen_opt_t;


typedef enum {
    NGX_HTTP_POST_READ_PHASE = 0,

    NGX_HTTP_SERVER_REWRITE_PHASE,

    NGX_HTTP_FIND_CONFIG_PHASE,
    NGX_HTTP_REWRITE_PHASE,
    NGX_HTTP_POST_REWRITE_PHASE,

    NGX_HTTP_PREACCESS_PHASE,

    NGX_HTTP_ACCESS_PHASE,
    NGX_HTTP_POST_ACCESS_PHASE,

    NGX_HTTP_TRY_FILES_PHASE,
    NGX_HTTP_CONTENT_PHASE,

    NGX_HTTP_LOG_PHASE
} ngx_http_phases;

typedef struct ngx_http_phase_handler_s  ngx_http_phase_handler_t;

typedef ngx_int_t (*ngx_http_phase_handler_pt)(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);

struct ngx_http_phase_handler_s {
    ngx_http_phase_handler_pt  checker;
    ngx_http_handler_pt        handler;
    ngx_uint_t                 next;
};


typedef struct {
    ngx_http_phase_handler_t  *handlers;
    ngx_uint_t                 server_rewrite_index;
    ngx_uint_t                 location_rewrite_index;
} ngx_http_phase_engine_t;


typedef struct {
    ngx_array_t                handlers;
} ngx_http_phase_t;


typedef struct {
    ngx_array_t                servers;         /* ngx_http_core_srv_conf_t */

    ngx_http_phase_engine_t    phase_engine;

    ngx_hash_t                 headers_in_hash;

```

```
    ngx_hash_t                 variables_hash;

    ngx_array_t                variables;       /* ngx_http_variable_t */
    ngx_uint_t                 ncaptures;

    ngx_uint_t                 server_names_hash_max_size;
    ngx_uint_t                 server_names_hash_bucket_size;

    ngx_uint_t                 variables_hash_max_size;
    ngx_uint_t                 variables_hash_bucket_size;

    ngx_hash_keys_arrays_t    *variables_keys;

    ngx_array_t               *ports;

    ngx_uint_t                 try_files;       /* unsigned  try_files:1 */

    ngx_http_phase_t           phases[NGX_HTTP_LOG_PHASE + 1];
} ngx_http_core_main_conf_t;


typedef struct {
    /* array of the ngx_http_server_name_t, "server_name" directive */
    ngx_array_t                server_names;

    /* server ctx */
    ngx_http_conf_ctx_t        *ctx;

    ngx_str_t                  server_name;

    size_t                     connection_pool_size;
    size_t                     request_pool_size;
    size_t                     client_header_buffer_size;

    ngx_bufs_t                 large_client_header_buffers;

    ngx_msec_t                 client_header_timeout;

    ngx_flag_t                 ignore_invalid_headers;
    ngx_flag_t                 merge_slashes;
    ngx_flag_t                 underscores_in_headers;

    unsigned                   listen:1;
#if (NGX_PCRE)
    unsigned                   captures:1;
#endif

    ngx_http_core_loc_conf_t  **named_locations;
} ngx_http_core_srv_conf_t;


/* list of structures to find core_srv_conf quickly at run time */


typedef struct {
#if (NGX_PCRE)
    ngx_http_regex_t          *regex;
#endif
    ngx_http_core_srv_conf_t  *server;   /* virtual name server conf */
    ngx_str_t                  name;
} ngx_http_server_name_t;


typedef struct {
     ngx_hash_combined_t       names;

     ngx_uint_t                nregex;
     ngx_http_server_name_t   *regex;
} ngx_http_virtual_names_t;


struct ngx_http_addr_conf_s {
    /* the default server configuration for this address:port */
    ngx_http_core_srv_conf_t  *default_server;

    ngx_http_virtual_names_t  *virtual_names;
```

```
240
241 #if (NGX_HTTP_SSL)
242     unsigned                  ssl:1;
243 #endif
244 #if (NGX_HTTP_SPDY)
245     unsigned                  spdy:1;
246 #endif
247     unsigned                  proxy_protocol:1;
248 };
249
250
251 typedef struct {
252     in_addr_t                 addr;
253     ngx_http_addr_conf_t      conf;
254 } ngx_http_in_addr_t;
255
256
257 #if (NGX_HAVE_INET6)
258
259 typedef struct {
260     struct in6_addr           addr6;
261     ngx_http_addr_conf_t      conf;
262 } ngx_http_in6_addr_t;
263
264 #endif
265
266
267 typedef struct {
268     /* ngx_http_in_addr_t or ngx_http_in6_addr_t */
269     void                      *addrs;
270     ngx_uint_t                naddrs;
271 } ngx_http_port_t;
272
273
274 typedef struct {
275     ngx_int_t                 family;
276     in_port_t                 port;
277     ngx_array_t               addrs;     /* array of ngx_http_conf_addr_t */
278 } ngx_http_conf_port_t;
279
280
281 typedef struct {
282     ngx_http_listen_opt_t     opt;
283
284     ngx_hash_t                hash;
285     ngx_hash_wildcard_t       *wc_head;
286     ngx_hash_wildcard_t       *wc_tail;
287
288 #if (NGX_PCRE)
289     ngx_uint_t                nregex;
290     ngx_http_server_name_t    *regex;
291 #endif
292
293     /* the default server configuration for this address:port */
294     ngx_http_core_srv_conf_t  *default_server;
295     ngx_array_t               servers;  /* array of ngx_http_core_srv_conf_t */
296 } ngx_http_conf_addr_t;
297
298
299 typedef struct {
300     ngx_int_t                 status;
301     ngx_int_t                 overwrite;
302     ngx_http_complex_value_t  value;
303     ngx_str_t                 args;
304 } ngx_http_err_page_t;
305
306
307 typedef struct {
308     ngx_array_t               *lengths;
309     ngx_array_t               *values;
310     ngx_str_t                 name;
311
312     unsigned                  code:10;
313     unsigned                  test_dir:1;
314 } ngx_http_try_file_t;
315
```

```
struct ngx_http_core_loc_conf_s {
    ngx_str_t     name;          /* location name */

#if (NGX_PCRE)
    ngx_http_regex_t  *regex;
#endif

    unsigned      noname:1;   /* "if () {}" block or limit_except */
    unsigned      lmt_excpt:1;
    unsigned      named:1;

    unsigned      exact_match:1;
    unsigned      noregex:1;

    unsigned      auto_redirect:1;
#if (NGX_HTTP_GZIP)
    unsigned      gzip_disable_msie6:2;
#if (NGX_HTTP_DEGRADATION)
    unsigned      gzip_disable_degradation:2;
#endif
#endif

    ngx_http_location_tree_node_t   *static_locations;
#if (NGX_PCRE)
    ngx_http_core_loc_conf_t       **regex_locations;
#endif

    /* pointer to the modules' loc_conf */
    void        **loc_conf;

    uint32_t      limit_except;
    void        **limit_except_loc_conf;

    ngx_http_handler_pt  handler;

    /* location name length for inclusive location with inherited alias */
    size_t        alias;
    ngx_str_t     root;                    /* root, alias */
    ngx_str_t     post_action;

    ngx_array_t  *root_lengths;
    ngx_array_t  *root_values;

    ngx_array_t  *types;
    ngx_hash_t    types_hash;
    ngx_str_t     default_type;

    off_t         client_max_body_size;    /* client_max_body_size */
    off_t         directio;                /* directio */
    off_t         directio_alignment;      /* directio_alignment */

    size_t        client_body_buffer_size; /* client_body_buffer_size */
    size_t        send_lowat;              /* send_lowat */
    size_t        postpone_output;         /* postpone_output */
    size_t        limit_rate;              /* limit_rate */
    size_t        limit_rate_after;        /* limit_rate_after */
    size_t        sendfile_max_chunk;      /* sendfile_max_chunk */
    size_t        read_ahead;              /* read_ahead */

    ngx_msec_t    client_body_timeout;     /* client_body_timeout */
    ngx_msec_t    send_timeout;            /* send_timeout */
    ngx_msec_t    keepalive_timeout;       /* keepalive_timeout */
    ngx_msec_t    lingering_time;          /* lingering_time */
    ngx_msec_t    lingering_timeout;       /* lingering_timeout */
    ngx_msec_t    resolver_timeout;        /* resolver_timeout */

    ngx_resolver_t  *resolver;             /* resolver */

    time_t        keepalive_header;        /* keepalive_timeout */

    ngx_uint_t    keepalive_requests;      /* keepalive_requests */
    ngx_uint_t    keepalive_disable;       /* keepalive_disable */
    ngx_uint_t    satisfy;                 /* satisfy */
    ngx_uint_t    lingering_close;         /* lingering_close */
    ngx_uint_t    if_modified_since;       /* if_modified_since */
```

```
    ngx_uint_t    max_ranges;              /* max_ranges */
    ngx_uint_t    client_body_in_file_only; /* client_body_in_file_only */

    ngx_flag_t    client_body_in_single_buffer;
                                           /* client_body_in_singe_buffer */
    ngx_flag_t    internal;                /* internal */
    ngx_flag_t    sendfile;                /* sendfile */
#if (NGX_HAVE_FILE_AIO)
    ngx_flag_t    aio;                     /* aio */
#endif
    ngx_flag_t    tcp_nopush;              /* tcp_nopush */
    ngx_flag_t    tcp_nodelay;             /* tcp_nodelay */
    ngx_flag_t    reset_timedout_connection; /* reset_timedout_connection */
    ngx_flag_t    server_name_in_redirect; /* server_name_in_redirect */
    ngx_flag_t    port_in_redirect;        /* port_in_redirect */
    ngx_flag_t    msie_padding;            /* msie_padding */
    ngx_flag_t    msie_refresh;            /* msie_refresh */
    ngx_flag_t    log_not_found;           /* log_not_found */
    ngx_flag_t    log_subrequest;          /* log_subrequest */
    ngx_flag_t    recursive_error_pages;   /* recursive_error_pages */
    ngx_flag_t    server_tokens;           /* server_tokens */
    ngx_flag_t    chunked_transfer_encoding; /* chunked_transfer_encoding */
    ngx_flag_t    etag;                    /* etag */

#if (NGX_HTTP_GZIP)
    ngx_flag_t    gzip_vary;               /* gzip_vary */

    ngx_uint_t    gzip_http_version;       /* gzip_http_version */
    ngx_uint_t    gzip_proxied;            /* gzip_proxied */

#if (NGX_PCRE)
    ngx_array_t  *gzip_disable;            /* gzip_disable */
#endif
#endif

#if (NGX_HAVE_OPENAT)
    ngx_uint_t    disable_symlinks;        /* disable_symlinks */
    ngx_http_complex_value_t  *disable_symlinks_from;
#endif

    ngx_array_t  *error_pages;             /* error_page */
    ngx_http_try_file_t    *try_files;     /* try_files */

    ngx_path_t   *client_body_temp_path;   /* client_body_temp_path */

    ngx_open_file_cache_t  *open_file_cache;
    time_t        open_file_cache_valid;
    ngx_uint_t    open_file_cache_min_uses;
    ngx_flag_t    open_file_cache_errors;
    ngx_flag_t    open_file_cache_events;

    ngx_log_t    *error_log;

    ngx_uint_t    types_hash_max_size;
    ngx_uint_t    types_hash_bucket_size;

    ngx_queue_t  *locations;

#if 0
    ngx_http_core_loc_conf_t  *prev_location;
#endif
};


typedef struct {
    ngx_queue_t                      queue;
    ngx_http_core_loc_conf_t        *exact;
    ngx_http_core_loc_conf_t        *inclusive;
    ngx_str_t                       *name;
    u_char                          *file_name;
    ngx_uint_t                       line;
    ngx_queue_t                      list;
} ngx_http_location_queue_t;


struct ngx_http_location_tree_node_s {
```

```
    ngx_http_location_tree_node_t   *left;
    ngx_http_location_tree_node_t   *right;
    ngx_http_location_tree_node_t   *tree;

    ngx_http_core_loc_conf_t        *exact;
    ngx_http_core_loc_conf_t        *inclusive;

    u_char                           auto_redirect;
    u_char                           len;
    u_char                           name[1];
};


void ngx_http_core_run_phases(ngx_http_request_t *r);
ngx_int_t ngx_http_core_generic_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_rewrite_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_find_config_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_post_rewrite_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_access_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_post_access_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_try_files_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);
ngx_int_t ngx_http_core_content_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph);


void *ngx_http_test_content_type(ngx_http_request_t *r, ngx_hash_t *types_hash);
ngx_int_t ngx_http_set_content_type(ngx_http_request_t *r);
void ngx_http_set_exten(ngx_http_request_t *r);
ngx_int_t ngx_http_set_etag(ngx_http_request_t *r);
void ngx_http_weak_etag(ngx_http_request_t *r);
ngx_int_t ngx_http_send_response(ngx_http_request_t *r, ngx_uint_t status,
    ngx_str_t *ct, ngx_http_complex_value_t *cv);
u_char *ngx_http_map_uri_to_path(ngx_http_request_t *r, ngx_str_t *name,
    size_t *root_length, size_t reserved);
ngx_int_t ngx_http_auth_basic_user(ngx_http_request_t *r);
#if (NGX_HTTP_GZIP)
ngx_int_t ngx_http_gzip_ok(ngx_http_request_t *r);
#endif


ngx_int_t ngx_http_subrequest(ngx_http_request_t *r,
    ngx_str_t *uri, ngx_str_t *args, ngx_http_request_t **sr,
    ngx_http_post_subrequest_t *psr, ngx_uint_t flags);
ngx_int_t ngx_http_internal_redirect(ngx_http_request_t *r,
    ngx_str_t *uri, ngx_str_t *args);
ngx_int_t ngx_http_named_location(ngx_http_request_t *r, ngx_str_t *name);


ngx_http_cleanup_t *ngx_http_cleanup_add(ngx_http_request_t *r, size_t size);


typedef ngx_int_t (*ngx_http_output_header_filter_pt)(ngx_http_request_t *r);
typedef ngx_int_t (*ngx_http_output_body_filter_pt)
    (ngx_http_request_t *r, ngx_chain_t *chain);


ngx_int_t ngx_http_output_filter(ngx_http_request_t *r, ngx_chain_t *chain);
ngx_int_t ngx_http_write_filter(ngx_http_request_t *r, ngx_chain_t *chain);


ngx_int_t ngx_http_set_disable_symlinks(ngx_http_request_t *r,
    ngx_http_core_loc_conf_t *clcf, ngx_str_t *path, ngx_open_file_info_t *of);

ngx_int_t ngx_http_get_forwarded_addr(ngx_http_request_t *r, ngx_addr_t *addr,
    ngx_array_t *headers, ngx_str_t *value, ngx_array_t *proxies,
    int recursive);


extern ngx_module_t  ngx_http_core_module;
```

```
544
545 extern ngx_uint_t ngx_http_max_module;
546
547 extern ngx_str_t  ngx_http_core_get_method;
548
549
550 #define ngx_http_clear_content_length(r)                                     \
551                                                                              \
552     r->headers_out.content_length_n = -1;                                    \
553     if (r->headers_out.content_length) {                                     \
554         r->headers_out.content_length->hash = 0;                             \
555         r->headers_out.content_length = NULL;                                \
556     }
557
558 #define ngx_http_clear_accept_ranges(r)                                      \
559                                                                              \
560     r->allow_ranges = 0;                                                     \
561     if (r->headers_out.accept_ranges) {                                      \
562         r->headers_out.accept_ranges->hash = 0;                              \
563         r->headers_out.accept_ranges = NULL;                                 \
564     }
565
566 #define ngx_http_clear_last_modified(r)                                      \
567                                                                              \
568     r->headers_out.last_modified_time = -1;                                  \
569     if (r->headers_out.last_modified) {                                      \
570         r->headers_out.last_modified->hash = 0;                              \
571         r->headers_out.last_modified = NULL;                                 \
572     }
573
574 #define ngx_http_clear_location(r)                                           \
575                                                                              \
576     if (r->headers_out.location) {                                           \
577         r->headers_out.location->hash = 0;                                   \
578         r->headers_out.location = NULL;                                      \
579     }
580
581 #define ngx_http_clear_etag(r)                                               \
582                                                                              \
583     if (r->headers_out.etag) {                                               \
584         r->headers_out.etag->hash = 0;                                       \
585         r->headers_out.etag = NULL;                                          \
586     }
587
588
589 #endif /* _NGX_HTTP_CORE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_inet.h - nginx-1.7.10

Data types defined

- ngx_addr_t
- ngx_cidr_t
- ngx_in6_cidr_t
- ngx_in_cidr_t
- ngx_url_t

Macros defined

- NGX_INET6_ADDRSTRLEN
- NGX_INET_ADDRSTRLEN
- NGX_SOCKADDRLEN
- NGX_SOCKADDRLEN
- NGX_SOCKADDR_STRLEN
- NGX_SOCKADDR_STRLEN
- NGX_UNIX_ADDRSTRLEN
- _NGX_INET_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_INET_H_INCLUDED_
9  #define _NGX_INET_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  /*
17  * TODO: autoconfigure NGX_SOCKADDRLEN and NGX_SOCKADDR_STRLEN as
18  *       sizeof(struct sockaddr_storage)
19  *       sizeof(struct sockaddr_un)
20  *       sizeof(struct sockaddr_in6)
21  *       sizeof(struct sockaddr_in)
22  */
23
24  #define NGX_INET_ADDRSTRLEN   (sizeof("255.255.255.255") - 1)
25  #define NGX_INET6_ADDRSTRLEN                                                 \
26      (sizeof("ffff:ffff:ffff:ffff:ffff:ffff:255.255.255.255") - 1)
27  #define NGX_UNIX_ADDRSTRLEN                                                  \
28      (sizeof(struct sockaddr_un) - offsetof(struct sockaddr_un, sun_path))
29
30  #if (NGX_HAVE_UNIX_DOMAIN)
31  #define NGX_SOCKADDR_STRLEN   (sizeof("unix:") - 1 + NGX_UNIX_ADDRSTRLEN)
32  #else
```

```
#define NGX_SOCKADDR_STRLEN   (NGX_INET6_ADDRSTRLEN + sizeof("[]:65535") - 1)
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
#define NGX_SOCKADDRLEN       sizeof(struct sockaddr_un)
#else
#define NGX_SOCKADDRLEN       512
#endif


typedef struct {
    in_addr_t                 addr;
    in_addr_t                 mask;
} ngx_in_cidr_t;


#if (NGX_HAVE_INET6)

typedef struct {
    struct in6_addr           addr;
    struct in6_addr           mask;
} ngx_in6_cidr_t;

#endif


typedef struct {
    ngx_uint_t                family;
    union {
        ngx_in_cidr_t         in;
#if (NGX_HAVE_INET6)
        ngx_in6_cidr_t        in6;
#endif
    } u;
} ngx_cidr_t;


typedef struct {
    struct sockaddr          *sockaddr;
    socklen_t                 socklen;
    ngx_str_t                 name;
} ngx_addr_t;


typedef struct {
    ngx_str_t                 url;
    ngx_str_t                 host;
    ngx_str_t                 port_text;
    ngx_str_t                 uri;

    in_port_t                 port;
    in_port_t                 default_port;
    int                       family;

    unsigned                  listen:1;
    unsigned                  uri_part:1;
    unsigned                  no_resolve:1;
    unsigned                  one_addr:1;  /* compatibility */

    unsigned                  no_port:1;
    unsigned                  wildcard:1;

    socklen_t                 socklen;
    u_char                    sockaddr[NGX_SOCKADDRLEN];

    ngx_addr_t               *addrs;
    ngx_uint_t                naddrs;

     char                     *err;
} ngx_url_t;


in_addr_t ngx_inet_addr(u_char *text, size_t len);
#if (NGX_HAVE_INET6)
ngx_int_t ngx_inet6_addr(u_char *p, size_t len, u_char *addr);
size_t ngx_inet6_ntop(u_char *p, u_char *text, size_t len);
```

```
#endif
size_t ngx_sock_ntop(struct sockaddr *sa, socklen_t socklen, u_char *text,
    size_t len, ngx_uint_t port);
size_t ngx_inet_ntop(int family, void *addr, u_char *text, size_t len);
ngx_int_t ngx_ptocidr(ngx_str_t *text, ngx_cidr_t *cidr);
ngx_int_t ngx_parse_addr(ngx_pool_t *pool, ngx_addr_t *addr, u_char *text,
    size_t len);
ngx_int_t ngx_parse_url(ngx_pool_t *pool, ngx_url_t *u);
ngx_int_t ngx_inet_resolve_host(ngx_pool_t *pool, ngx_url_t *u);
ngx_int_t ngx_cmp_sockaddr(struct sockaddr *sa1, socklen_t slen1,
    struct sockaddr *sa2, socklen_t slen2, ngx_uint_t cmp_port);


#endif /* _NGX_INET_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_inet.c - nginx-1.7.10

Functions defined

- ngx_cmp_sockaddr
- ngx_inet6_addr
- ngx_inet6_ntop
- ngx_inet_addr
- ngx_inet_ntop
- ngx_inet_resolve_host
- ngx_inet_resolve_host
- ngx_parse_addr
- ngx_parse_inet6_url
- ngx_parse_inet_url
- ngx_parse_unix_domain_url
- ngx_parse_url
- ngx_ptocidr
- ngx_sock_ntop

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  static ngx_int_t ngx_parse_unix_domain_url(ngx_pool_t *pool, ngx_url_t *u);
13  static ngx_int_t ngx_parse_inet_url(ngx_pool_t *pool, ngx_url_t *u);
14  static ngx_int_t ngx_parse_inet6_url(ngx_pool_t *pool, ngx_url_t *u);
15
16
17  in_addr_t
18  ngx_inet_addr(u_char *text, size_t len)
19  {
20      u_char      *p, c;
21      in_addr_t    addr;
22      ngx_uint_t   octet, n;
23
24      addr = 0;
25      octet = 0;
26      n = 0;
27
28      for (p = text; p < text + len; p++) {
29
30          c = *p;
31
32          if (c >= '0' && c <= '9') {
33              octet = octet * 10 + (c - '0');
```

```
            continue;
        }

        if (c == '.' && octet < 256) {
            addr = (addr << 8) + octet;
            octet = 0;
            n++;
            continue;
        }

        return INADDR_NONE;
    }

    if (n == 3 && octet < 256) {
        addr = (addr << 8) + octet;
        return htonl(addr);
    }

    return INADDR_NONE;
}


#if (NGX_HAVE_INET6)

ngx_int_t
ngx_inet6_addr(u_char *p, size_t len, u_char *addr)
{
    u_char      c, *zero, *digit, *s, *d;
    size_t      len4;
    ngx_uint_t  n, nibbles, word;

    if (len == 0) {
        return NGX_ERROR;
    }

    zero = NULL;
    digit = NULL;
    len4 = 0;
    nibbles = 0;
    word = 0;
    n = 8;

    if (p[0] == ':') {
        p++;
        len--;
    }

    for (/* void */; len; len--) {
        c = *p++;

        if (c == ':') {
            if (nibbles) {
                digit = p;
                len4 = len;
                *addr++ = (u_char) (word >> 8);
                *addr++ = (u_char) (word & 0xff);

                if (--n) {
                    nibbles = 0;
                    word = 0;
                    continue;
                }

            } else {
                if (zero == NULL) {
                    digit = p;
                     len4 = len;
                     zero = addr;
                     continue;
                 }
             }

             return NGX_ERROR;
         }

         if (c == '.' && nibbles) {
```

```
            if (n < 2 || digit == NULL) {
                return NGX_ERROR;
            }

            word = ngx_inet_addr(digit, len4 - 1);
            if (word == INADDR_NONE) {
                return NGX_ERROR;
            }

            word = ntohl(word);
            *addr++ = (u_char) ((word >> 24) & 0xff);
            *addr++ = (u_char) ((word >> 16) & 0xff);
            n--;
            break;
        }

        if (++nibbles > 4) {
            return NGX_ERROR;
        }

        if (c >= '0' && c <= '9') {
            word = word * 16 + (c - '0');
            continue;
        }

        c |= 0x20;

        if (c >= 'a' && c <= 'f') {
            word = word * 16 + (c - 'a') + 10;
            continue;
        }

        return NGX_ERROR;
    }

    if (nibbles == 0 && zero == NULL) {
        return NGX_ERROR;
    }

    *addr++ = (u_char) (word >> 8);
    *addr++ = (u_char) (word & 0xff);

    if (--n) {
        if (zero) {
            n *= 2;
            s = addr - 1;
            d = s + n;
            while (s >= zero) {
                *d-- = *s--;
            }
            ngx_memzero(zero, n);
            return NGX_OK;
        }

    } else {
        if (zero == NULL) {
            return NGX_OK;
        }
    }

    return NGX_ERROR;
}

#endif


size_t
ngx_sock_ntop(struct sockaddr *sa, socklen_t socklen, u_char *text, size_t len,
    ngx_uint_t port)
{
    u_char               *p;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    size_t                n;
    struct sockaddr_in6  *sin6;
#endif
```

```c
#if (NGX_HAVE_UNIX_DOMAIN)
    struct sockaddr_un   *saun;
#endif

    switch (sa->sa_family) {

    case AF_INET:

        sin = (struct sockaddr_in *) sa;
        p = (u_char *) &sin->sin_addr;

        if (port) {
            p = ngx_snprintf(text, len, "%ud.%ud.%ud.%ud:%d",
                             p[0], p[1], p[2], p[3], ntohs(sin->sin_port));
        } else {
            p = ngx_snprintf(text, len, "%ud.%ud.%ud.%ud",
                             p[0], p[1], p[2], p[3]);
        }

        return (p - text);

#if (NGX_HAVE_INET6)

    case AF_INET6:

        sin6 = (struct sockaddr_in6 *) sa;

        n = 0;

        if (port) {
            text[n++] = '[';
        }

        n = ngx_inet6_ntop(sin6->sin6_addr.s6_addr, &text[n], len);

        if (port) {
            n = ngx_sprintf(&text[1 + n], "]:%d",
                            ntohs(sin6->sin6_port)) - text;
        }

        return n;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)

    case AF_UNIX:
        saun = (struct sockaddr_un *) sa;

        /* on Linux sockaddr might not include sun_path at all */

        if (socklen <= (socklen_t) offsetof(struct sockaddr_un, sun_path)) {
            p = ngx_snprintf(text, len, "unix:%Z");

        } else {
            p = ngx_snprintf(text, len, "unix:%s%Z", saun->sun_path);
        }

        /* we do not include trailing zero in address length */

        return (p - text - 1);

#endif

    default:
        return 0;
    }
}


size_t
ngx_inet_ntop(int family, void *addr, u_char *text, size_t len)
{
    u_char  *p;

    switch (family) {

```

```
    case AF_INET:

        p = addr;

        return ngx_snprintf(text, len, "%ud.%ud.%ud.%ud",
                            p[0], p[1], p[2], p[3])
               - text;

#if (NGX_HAVE_INET6)

    case AF_INET6:
        return ngx_inet6_ntop(addr, text, len);

#endif

    default:
        return 0;
    }
}


#if (NGX_HAVE_INET6)

size_t
ngx_inet6_ntop(u_char *p, u_char *text, size_t len)
{
    u_char      *dst;
    size_t       max, n;
    ngx_uint_t   i, zero, last;

    if (len < NGX_INET6_ADDRSTRLEN) {
        return 0;
    }

    zero = (ngx_uint_t) -1;
    last = (ngx_uint_t) -1;
    max = 1;
    n = 0;

    for (i = 0; i < 16; i += 2) {

        if (p[i] || p[i + 1]) {

            if (max < n) {
                zero = last;
                max = n;
            }

            n = 0;
            continue;
        }

        if (n++ == 0) {
            last = i;
        }
    }

    if (max < n) {
        zero = last;
        max = n;
    }

    dst = text;
    n = 16;

    if (zero == 0) {

        if ((max == 5 && p[10] == 0xff && p[11] == 0xff)
            || (max == 6)
            || (max == 7 && p[14] != 0 && p[15] != 1))
        {
            n = 12;
        }

        *dst++ = ':';
    }
```

```
    for (i = 0; i < n; i += 2) {

        if (i == zero) {
            *dst++ = ':';
            i += (max - 1) * 2;
            continue;
        }

        dst = ngx_sprintf(dst, "%uxi", p[i] * 256 + p[i + 1]);

        if (i < 14) {
            *dst++ = ':';
        }
    }

    if (n == 12) {
        dst = ngx_sprintf(dst, "%ud.%ud.%ud.%ud", p[12], p[13], p[14], p[15]);
    }

    return dst - text;
}

#endif


ngx_int_t
ngx_ptocidr(ngx_str_t *text, ngx_cidr_t *cidr)
{
    u_char      *addr, *mask, *last;
    size_t       len;
    ngx_int_t    shift;
#if (NGX_HAVE_INET6)
    ngx_int_t    rc;
    ngx_uint_t   s, i;
#endif

    addr = text->data;
    last = addr + text->len;

    mask = ngx_strlchr(addr, last, '/');
    len = (mask ? mask : last) - addr;

    cidr->u.in.addr = ngx_inet_addr(addr, len);

    if (cidr->u.in.addr != INADDR_NONE) {
        cidr->family = AF_INET;

        if (mask == NULL) {
            cidr->u.in.mask = 0xffffffff;
            return NGX_OK;
        }

#if (NGX_HAVE_INET6)
    } else if (ngx_inet6_addr(addr, len, cidr->u.in6.addr.s6_addr) == NGX_OK) {
        cidr->family = AF_INET6;

        if (mask == NULL) {
            ngx_memset(cidr->u.in6.mask.s6_addr, 0xff, 16);
            return NGX_OK;
        }

#endif
    } else {
        return NGX_ERROR;
    }

    mask++;

    shift = ngx_atoi(mask, last - mask);
    if (shift == NGX_ERROR) {
        return NGX_ERROR;
    }

    switch (cidr->family) {

```

```
#if (NGX_HAVE_INET6)
    case AF_INET6:
        if (shift > 128) {
            return NGX_ERROR;
        }

        addr = cidr->u.in6.addr.s6_addr;
        mask = cidr->u.in6.mask.s6_addr;
        rc = NGX_OK;

        for (i = 0; i < 16; i++) {

            s = (shift > 8) ? 8 : shift;
            shift -= s;

            mask[i] = (u_char) (0xffu << (8 - s));

            if (addr[i] != (addr[i] & mask[i])) {
                rc = NGX_DONE;
                addr[i] &= mask[i];
            }
        }

        return rc;
#endif

    default: /* AF_INET */
        if (shift > 32) {
            return NGX_ERROR;
        }

        if (shift) {
            cidr->u.in.mask = htonl((uint32_t) (0xffffffffu << (32 - shift)));

        } else {
            /* x86 compilers use a shl instruction that shifts by modulo 32 */
            cidr->u.in.mask = 0;
        }

        if (cidr->u.in.addr == (cidr->u.in.addr & cidr->u.in.mask)) {
            return NGX_OK;
        }

        cidr->u.in.addr &= cidr->u.in.mask;

        return NGX_DONE;
    }
}


ngx_int_t
ngx_parse_addr(ngx_pool_t *pool, ngx_addr_t *addr, u_char *text, size_t len)
{
    in_addr_t            inaddr;
    ngx_uint_t           family;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct in6_addr      inaddr6;
    struct sockaddr_in6  *sin6;

    /*
     * prevent MSVC8 warning:
     *    potentially uninitialized local variable 'inaddr6' used
     */
    ngx_memzero(&inaddr6, sizeof(struct in6_addr));
#endif

    inaddr = ngx_inet_addr(text, len);

    if (inaddr != INADDR_NONE) {
        family = AF_INET;
        len = sizeof(struct sockaddr_in);

#if (NGX_HAVE_INET6)
    } else if (ngx_inet6_addr(text, len, inaddr6.s6_addr) == NGX_OK) {
        family = AF_INET6;
```

```
        len = sizeof(struct sockaddr_in6);

#endif
    } else {
        return NGX_DECLINED;
    }

    addr->sockaddr = ngx_pcalloc(pool, len);
    if (addr->sockaddr == NULL) {
        return NGX_ERROR;
    }

    addr->sockaddr->sa_family = (u_char) family;
    addr->socklen = len;

    switch (family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) addr->sockaddr;
        ngx_memcpy(sin6->sin6_addr.s6_addr, inaddr6.s6_addr, 16);
        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) addr->sockaddr;
        sin->sin_addr.s_addr = inaddr;
        break;
    }

    return NGX_OK;
}


ngx_int_t
ngx_parse_url(ngx_pool_t *pool, ngx_url_t *u)
{
    u_char  *p;

    p = u->url.data;

    if (ngx_strncasecmp(p, (u_char *) "unix:", 5) == 0) {
        return ngx_parse_unix_domain_url(pool, u);
    }

    if (p[0] == '[') {
        return ngx_parse_inet6_url(pool, u);
    }

    return ngx_parse_inet_url(pool, u);
}


static ngx_int_t
ngx_parse_unix_domain_url(ngx_pool_t *pool, ngx_url_t *u)
{
#if (NGX_HAVE_UNIX_DOMAIN)
    u_char              *path, *uri, *last;
    size_t               len;
    struct sockaddr_un  *saun;

    len = u->url.len;
    path = u->url.data;

    path += 5;
    len -= 5;

    if (u->uri_part) {

        last = path + len;
        uri = ngx_strlchr(path, last, ':');

        if (uri) {
            len = uri - path;
            uri++;
            u->uri.len = last - uri;
```

```
            u->uri.data = uri;
        }
    }

    if (len == 0) {
        u->err = "no path in the unix domain socket";
        return NGX_ERROR;
    }

    u->host.len = len++;
    u->host.data = path;

    if (len > sizeof(saun->sun_path)) {
        u->err = "too long path in the unix domain socket";
        return NGX_ERROR;
    }

    u->socklen = sizeof(struct sockaddr_un);
    saun = (struct sockaddr_un *) &u->sockaddr;
    saun->sun_family = AF_UNIX;
    (void) ngx_cpystrn((u_char *) saun->sun_path, path, len);

    u->addrs = ngx_pcalloc(pool, sizeof(ngx_addr_t));
    if (u->addrs == NULL) {
        return NGX_ERROR;
    }

    saun = ngx_pcalloc(pool, sizeof(struct sockaddr_un));
    if (saun == NULL) {
        return NGX_ERROR;
    }

    u->family = AF_UNIX;
    u->naddrs = 1;

    saun->sun_family = AF_UNIX;
    (void) ngx_cpystrn((u_char *) saun->sun_path, path, len);

    u->addrs[0].sockaddr = (struct sockaddr *) saun;
    u->addrs[0].socklen = sizeof(struct sockaddr_un);
    u->addrs[0].name.len = len + 4;
    u->addrs[0].name.data = u->url.data;

    return NGX_OK;

#else

    u->err = "the unix domain sockets are not supported on this platform";

    return NGX_ERROR;

#endif
}


static ngx_int_t
ngx_parse_inet_url(ngx_pool_t *pool, ngx_url_t *u)
{
    u_char              *p, *host, *port, *last, *uri, *args;
    size_t               len;
    ngx_int_t            n;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
#endif

    u->socklen = sizeof(struct sockaddr_in);
    sin = (struct sockaddr_in *) &u->sockaddr;
    sin->sin_family = AF_INET;

    u->family = AF_INET;

    host = u->url.data;

    last = host + u->url.len;

```

```
    port = ngx_strlchr(host, last, ':');

    uri = ngx_strlchr(host, last, '/');

    args = ngx_strlchr(host, last, '?');

    if (args) {
        if (uri == NULL || args < uri) {
            uri = args;
        }
    }

    if (uri) {
        if (u->listen || !u->uri_part) {
            u->err = "invalid host";
            return NGX_ERROR;
        }

        u->uri.len = last - uri;
        u->uri.data = uri;

        last = uri;

        if (uri < port) {
            port = NULL;
        }
    }

    if (port) {
        port++;

        len = last - port;

        n = ngx_atoi(port, len);

        if (n < 1 || n > 65535) {
            u->err = "invalid port";
            return NGX_ERROR;
        }

        u->port = (in_port_t) n;
        sin->sin_port = htons((in_port_t) n);

        u->port_text.len = len;
        u->port_text.data = port;

        last = port - 1;

    } else {
        if (uri == NULL) {

            if (u->listen) {

                /* test value as port only */

                n = ngx_atoi(host, last - host);

                if (n != NGX_ERROR) {

                    if (n < 1 || n > 65535) {
                        u->err = "invalid port";
                        return NGX_ERROR;
                    }

                    u->port = (in_port_t) n;
                    sin->sin_port = htons((in_port_t) n);

                    u->port_text.len = last - host;
                    u->port_text.data = host;

                    u->wildcard = 1;

                    return NGX_OK;
                }
            }
        }
```

```
        u->no_port = 1;
        u->port = u->default_port;
        sin->sin_port = htons(u->default_port);
    }

    len = last - host;

    if (len == 0) {
        u->err = "no host";
        return NGX_ERROR;
    }

    u->host.len = len;
    u->host.data = host;

    if (u->listen && len == 1 && *host == '*') {
        sin->sin_addr.s_addr = INADDR_ANY;
        u->wildcard = 1;
        return NGX_OK;
    }

    sin->sin_addr.s_addr = ngx_inet_addr(host, len);

    if (sin->sin_addr.s_addr != INADDR_NONE) {

        if (sin->sin_addr.s_addr == INADDR_ANY) {
            u->wildcard = 1;
        }

        u->naddrs = 1;

        u->addrs = ngx_pcalloc(pool, sizeof(ngx_addr_t));
        if (u->addrs == NULL) {
            return NGX_ERROR;
        }

        sin = ngx_pcalloc(pool, sizeof(struct sockaddr_in));
        if (sin == NULL) {
            return NGX_ERROR;
        }

        ngx_memcpy(sin, u->sockaddr, sizeof(struct sockaddr_in));

        u->addrs[0].sockaddr = (struct sockaddr *) sin;
        u->addrs[0].socklen = sizeof(struct sockaddr_in);

        p = ngx_pnalloc(pool, u->host.len + sizeof(":65535") - 1);
        if (p == NULL) {
            return NGX_ERROR;
        }

        u->addrs[0].name.len = ngx_sprintf(p, "%V:%d",
                                           &u->host, u->port) - p;
        u->addrs[0].name.data = p;

        return NGX_OK;
    }

    if (u->no_resolve) {
        return NGX_OK;
    }

    if (ngx_inet_resolve_host(pool, u) != NGX_OK) {
        return NGX_ERROR;
    }

    u->family = u->addrs[0].sockaddr->sa_family;
    u->socklen = u->addrs[0].socklen;
    ngx_memcpy(u->sockaddr, u->addrs[0].sockaddr, u->addrs[0].socklen);

    switch (u->family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) &u->sockaddr;
```

```
794
795         if (IN6_IS_ADDR_UNSPECIFIED(&sin6->sin6_addr)) {
796             u->wildcard = 1;
797         }
798
799         break;
800 #endif
801
802     default: /* AF_INET */
803         sin = (struct sockaddr_in *) &u->sockaddr;
804
805         if (sin->sin_addr.s_addr == INADDR_ANY) {
806             u->wildcard = 1;
807         }
808
809         break;
810     }
811
812     return NGX_OK;
813 }
814
815
816 static ngx_int_t
817 ngx_parse_inet6_url(ngx_pool_t *pool, ngx_url_t *u)
818 {
819 #if (NGX_HAVE_INET6)
820     u_char               *p, *host, *port, *last, *uri;
821     size_t                len;
822     ngx_int_t             n;
823     struct sockaddr_in6  *sin6;
824
825     u->socklen = sizeof(struct sockaddr_in6);
826     sin6 = (struct sockaddr_in6 *) &u->sockaddr;
827     sin6->sin6_family = AF_INET6;
828
829     host = u->url.data + 1;
830
831     last = u->url.data + u->url.len;
832
833     p = ngx_strlchr(host, last, ']');
834
835     if (p == NULL) {
836         u->err = "invalid host";
837         return NGX_ERROR;
838     }
839
840     if (last - p) {
841
842         port = p + 1;
843
844         uri = ngx_strlchr(port, last, '/');
845
846         if (uri) {
847             if (u->listen || !u->uri_part) {
848                 u->err = "invalid host";
849                 return NGX_ERROR;
850             }
851
852             u->uri.len = last - uri;
853             u->uri.data = uri;
854
855             last = uri;
856         }
857
858         if (*port == ':') {
859             port++;
860
861             len = last - port;
862
863             n = ngx_atoi(port, len);
864
865             if (n < 1 || n > 65535) {
866                 u->err = "invalid port";
867                 return NGX_ERROR;
868             }
869
```

```
            u->port = (in_port_t) n;
            sin6->sin6_port = htons((in_port_t) n);

            u->port_text.len = len;
            u->port_text.data = port;

        } else {
            u->no_port = 1;
            u->port = u->default_port;
            sin6->sin6_port = htons(u->default_port);
        }
    }

    len = p - host;

    if (len == 0) {
        u->err = "no host";
        return NGX_ERROR;
    }

    u->host.len = len + 2;
    u->host.data = host - 1;

    if (ngx_inet6_addr(host, len, sin6->sin6_addr.s6_addr) != NGX_OK) {
        u->err = "invalid IPv6 address";
        return NGX_ERROR;
    }

    if (IN6_IS_ADDR_UNSPECIFIED(&sin6->sin6_addr)) {
        u->wildcard = 1;
    }

    u->family = AF_INET6;
    u->naddrs = 1;

    u->addrs = ngx_pcalloc(pool, sizeof(ngx_addr_t));
    if (u->addrs == NULL) {
        return NGX_ERROR;
    }

    sin6 = ngx_pcalloc(pool, sizeof(struct sockaddr_in6));
    if (sin6 == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(sin6, u->sockaddr, sizeof(struct sockaddr_in6));

    u->addrs[0].sockaddr = (struct sockaddr *) sin6;
    u->addrs[0].socklen = sizeof(struct sockaddr_in6);

    p = ngx_pnalloc(pool, u->host.len + sizeof(":65535") - 1);
    if (p == NULL) {
        return NGX_ERROR;
    }

    u->addrs[0].name.len = ngx_sprintf(p, "%V:%d",
                                       &u->host, u->port) - p;
    u->addrs[0].name.data = p;

    return NGX_OK;

#else

    u->err = "the INET6 sockets are not supported on this platform";

    return NGX_ERROR;

#endif
}


#if (NGX_HAVE_GETADDRINFO && NGX_HAVE_INET6)

ngx_int_t
ngx_inet_resolve_host(ngx_pool_t *pool, ngx_url_t *u)
{
```

```
    u_char               *p, *host;
    size_t                len;
    in_port_t             port;
    ngx_uint_t            i;
    struct addrinfo       hints, *res, *rp;
    struct sockaddr_in   *sin;
    struct sockaddr_in6  *sin6;

    port = htons(u->port);

    host = ngx_alloc(u->host.len + 1, pool->log);
    if (host == NULL) {
        return NGX_ERROR;
    }

    (void) ngx_cpystrn(host, u->host.data, u->host.len + 1);

    ngx_memzero(&hints, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
#ifdef AI_ADDRCONFIG
    hints.ai_flags = AI_ADDRCONFIG;
#endif

    if (getaddrinfo((char *) host, NULL, &hints, &res) != 0) {
        u->err = "host not found";
        ngx_free(host);
        return NGX_ERROR;
    }

    ngx_free(host);

    for (i = 0, rp = res; rp != NULL; rp = rp->ai_next) {

        switch (rp->ai_family) {

        case AF_INET:
        case AF_INET6:
            break;

        default:
            continue;
        }

        i++;
    }

    if (i == 0) {
        u->err = "host not found";
        goto failed;
    }

    /* MP: ngx_shared_palloc() */

     u->addrs = ngx_pcalloc(pool, i * sizeof(ngx_addr_t));
     if (u->addrs == NULL) {
         goto failed;
     }

     u->naddrs = i;

     i = 0;

     /* AF_INET addresses first */

     for (rp = res; rp != NULL; rp = rp->ai_next) {

         if (rp->ai_family != AF_INET) {
             continue;
         }

         sin = ngx_pcalloc(pool, rp->ai_addrlen);
         if (sin == NULL) {
             goto failed;
         }

```

```
1022            ngx_memcpy(sin, rp->ai_addr, rp->ai_addrlen);
1023
1024            sin->sin_port = port;
1025
1026            u->addrs[i].sockaddr = (struct sockaddr *) sin;
1027            u->addrs[i].socklen = rp->ai_addrlen;
1028
1029            len = NGX_INET_ADDRSTRLEN + sizeof(":65535") - 1;
1030
1031            p = ngx_pnalloc(pool, len);
1032            if (p == NULL) {
1033                goto failed;
1034            }
1035
1036            len = ngx_sock_ntop((struct sockaddr *) sin, rp->ai_addrlen, p, len, 1);
1037
1038            u->addrs[i].name.len = len;
1039            u->addrs[i].name.data = p;
1040
1041            i++;
1042        }
1043
1044        for (rp = res; rp != NULL; rp = rp->ai_next) {
1045
1046            if (rp->ai_family != AF_INET6) {
1047                continue;
1048            }
1049
1050            sin6 = ngx_pcalloc(pool, rp->ai_addrlen);
1051            if (sin6 == NULL) {
1052                goto failed;
1053            }
1054
1055            ngx_memcpy(sin6, rp->ai_addr, rp->ai_addrlen);
1056
1057            sin6->sin6_port = port;
1058
1059            u->addrs[i].sockaddr = (struct sockaddr *) sin6;
1060            u->addrs[i].socklen = rp->ai_addrlen;
1061
1062            len = NGX_INET6_ADDRSTRLEN + sizeof("[]:65535") - 1;
1063
1064            p = ngx_pnalloc(pool, len);
1065            if (p == NULL) {
1066                goto failed;
1067            }
1068
1069            len = ngx_sock_ntop((struct sockaddr *) sin6, rp->ai_addrlen, p,
1070                                len, 1);
1071
1072            u->addrs[i].name.len = len;
1073            u->addrs[i].name.data = p;
1074
1075            i++;
1076        }
1077
1078        freeaddrinfo(res);
1079        return NGX_OK;
1080
1081    failed:
1082
1083        freeaddrinfo(res);
1084        return NGX_ERROR;
1085    }
1086
1087    #else /* !NGX_HAVE_GETADDRINFO || !NGX_HAVE_INET6 */
1088
1089    ngx_int_t
1090    ngx_inet_resolve_host(ngx_pool_t *pool, ngx_url_t *u)
1091    {
1092        u_char              *p, *host;
1093        size_t               len;
1094        in_port_t            port;
1095        in_addr_t            in_addr;
1096        ngx_uint_t           i;
1097        struct hostent      *h;
```

```
1098      struct sockaddr_in  *sin;
1099
1100      /* AF_INET only */
1101
1102      port = htons(u->port);
1103
1104      in_addr = ngx_inet_addr(u->host.data, u->host.len);
1105
1106      if (in_addr == INADDR_NONE) {
1107          host = ngx_alloc(u->host.len + 1, pool->log);
1108          if (host == NULL) {
1109              return NGX_ERROR;
1110          }
1111
1112          (void) ngx_cpystrn(host, u->host.data, u->host.len + 1);
1113
1114          h = gethostbyname((char *) host);
1115
1116          ngx_free(host);
1117
1118          if (h == NULL || h->h_addr_list[0] == NULL) {
1119              u->err = "host not found";
1120              return NGX_ERROR;
1121          }
1122
1123          for (i = 0; h->h_addr_list[i] != NULL; i++) { /* void */ }
1124
1125          /* MP: ngx_shared_palloc() */
1126
1127          u->addrs = ngx_pcalloc(pool, i * sizeof(ngx_addr_t));
1128          if (u->addrs == NULL) {
1129              return NGX_ERROR;
1130          }
1131
1132          u->naddrs = i;
1133
1134          for (i = 0; i < u->naddrs; i++) {
1135
1136              sin = ngx_pcalloc(pool, sizeof(struct sockaddr_in));
1137              if (sin == NULL) {
1138                  return NGX_ERROR;
1139              }
1140
1141              sin->sin_family = AF_INET;
1142              sin->sin_port = port;
1143              sin->sin_addr.s_addr = *(in_addr_t *) (h->h_addr_list[i]);
1144
1145              u->addrs[i].sockaddr = (struct sockaddr *) sin;
1146              u->addrs[i].socklen = sizeof(struct sockaddr_in);
1147
1148              len = NGX_INET_ADDRSTRLEN + sizeof(":65535") - 1;
1149
1150              p = ngx_pnalloc(pool, len);
1151              if (p == NULL) {
1152                  return NGX_ERROR;
1153              }
1154
1155              len = ngx_sock_ntop((struct sockaddr *) sin,
1156                                  sizeof(struct sockaddr_in), p, len, 1);
1157
1158              u->addrs[i].name.len = len;
1159              u->addrs[i].name.data = p;
1160          }
1161
1162      } else {
1163
1164          /* MP: ngx_shared_palloc() */
1165
1166          u->addrs = ngx_pcalloc(pool, sizeof(ngx_addr_t));
1167          if (u->addrs == NULL) {
1168              return NGX_ERROR;
1169          }
1170
1171          sin = ngx_pcalloc(pool, sizeof(struct sockaddr_in));
1172          if (sin == NULL) {
1173              return NGX_ERROR;
```

```
        }

        u->naddrs = 1;

        sin->sin_family = AF_INET;
        sin->sin_port = port;
        sin->sin_addr.s_addr = in_addr;

        u->addrs[0].sockaddr = (struct sockaddr *) sin;
        u->addrs[0].socklen = sizeof(struct sockaddr_in);

        p = ngx_pnalloc(pool, u->host.len + sizeof(":65535") - 1);
        if (p == NULL) {
            return NGX_ERROR;
        }

        u->addrs[0].name.len = ngx_sprintf(p, "%V:%d",
                                           &u->host, ntohs(port)) - p;
        u->addrs[0].name.data = p;
    }

    return NGX_OK;
}

#endif /* NGX_HAVE_GETADDRINFO && NGX_HAVE_INET6 */


ngx_int_t
ngx_cmp_sockaddr(struct sockaddr *sa1, socklen_t slen1,
    struct sockaddr *sa2, socklen_t slen2, ngx_uint_t cmp_port)
{
    struct sockaddr_in   *sin1, *sin2;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin61, *sin62;
#endif
#if (NGX_HAVE_UNIX_DOMAIN)
    struct sockaddr_un   *saun1, *saun2;
#endif

    if (sa1->sa_family != sa2->sa_family) {
        return NGX_DECLINED;
    }

    switch (sa1->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:

        sin61 = (struct sockaddr_in6 *) sa1;
        sin62 = (struct sockaddr_in6 *) sa2;

        if (cmp_port && sin61->sin6_port != sin62->sin6_port) {
            return NGX_DECLINED;
        }

        if (ngx_memcmp(&sin61->sin6_addr, &sin62->sin6_addr, 16) != 0) {
            return NGX_DECLINED;
        }

        break;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
    case AF_UNIX:

       /* TODO length */

       saun1 = (struct sockaddr_un *) sa1;
       saun2 = (struct sockaddr_un *) sa2;

       if (ngx_memcmp(&saun1->sun_path, &saun2->sun_path,
                      sizeof(saun1->sun_path))
           != 0)
       {
           return NGX_DECLINED;
       }
```

```
1250
1251        break;
1252 #endif
1253
1254     default: /* AF_INET */
1255
1256         sin1 = (struct sockaddr_in *) sa1;
1257         sin2 = (struct sockaddr_in *) sa2;
1258
1259         if (cmp_port && sin1->sin_port != sin2->sin_port) {
1260             return NGX_DECLINED;
1261         }
1262
1263         if (sin1->sin_addr.s_addr != sin2->sin_addr.s_addr) {
1264             return NGX_DECLINED;
1265         }
1266
1267         break;
1268     }
1269
1270     return NGX_OK;
1271 }
```

One Level Up Top Level

src/http/ngx_http_variables.h - nginx-1.7.10

Data types defined

- ngx_http_get_variable_pt
- ngx_http_map_regex_t
- ngx_http_map_t
- ngx_http_regex_t
- ngx_http_regex_variable_t
- ngx_http_set_variable_pt
- ngx_http_variable_s
- ngx_http_variable_t
- ngx_http_variable_value_t

Macros defined

- NGX_HTTP_VAR_CHANGEABLE
- NGX_HTTP_VAR_INDEXED
- NGX_HTTP_VAR_NOCACHEABLE
- NGX_HTTP_VAR_NOHASH
- _NGX_HTTP_VARIABLES_H_INCLUDED_
- ngx_http_variable

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_VARIABLES_H_INCLUDED_
9  #define _NGX_HTTP_VARIABLES_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
17  typedef ngx_variable_value_t  ngx_http_variable_value_t;
18
19  #define ngx_http_variable(v)     { sizeof(v) - 1, 1, 0, 0, 0, (u_char *) v }
20
21  typedef struct ngx_http_variable_s  ngx_http_variable_t;
22
23  typedef void (*ngx_http_set_variable_pt) (ngx_http_request_t *r,
24      ngx_http_variable_value_t *v, uintptr_t data);
25  typedef ngx_int_t (*ngx_http_get_variable_pt) (ngx_http_request_t *r,
26      ngx_http_variable_value_t *v, uintptr_t data);
27
```

```
28
29  #define NGX_HTTP_VAR_CHANGEABLE   1
30  #define NGX_HTTP_VAR_NOCACHEABLE  2
31  #define NGX_HTTP_VAR_INDEXED      4
32  #define NGX_HTTP_VAR_NOHASH       8
33
34
35  struct ngx_http_variable_s {
36      ngx_str_t                     name;   /* must be first to build the hash */
37      ngx_http_set_variable_pt      set_handler;
38      ngx_http_get_variable_pt      get_handler;
39      uintptr_t                     data;
40      ngx_uint_t                    flags;
41      ngx_uint_t                    index;
42  };
43
44
45  ngx_http_variable_t *ngx_http_add_variable(ngx_conf_t *cf, ngx_str_t *name,
46      ngx_uint_t flags);
47  ngx_int_t ngx_http_get_variable_index(ngx_conf_t *cf, ngx_str_t *name);
48  ngx_http_variable_value_t *ngx_http_get_indexed_variable(ngx_http_request_t *r,
49      ngx_uint_t index);
50  ngx_http_variable_value_t *ngx_http_get_flushed_variable(ngx_http_request_t *r,
51      ngx_uint_t index);
52
53  ngx_http_variable_value_t *ngx_http_get_variable(ngx_http_request_t *r,
54      ngx_str_t *name, ngx_uint_t key);
55
56  ngx_int_t ngx_http_variable_unknown_header(ngx_http_variable_value_t *v,
57      ngx_str_t *var, ngx_list_part_t *part, size_t prefix);
58
59
60  #if (NGX_PCRE)
61
62  typedef struct {
63      ngx_uint_t                    capture;
64      ngx_int_t                     index;
65  } ngx_http_regex_variable_t;
66
67
68  typedef struct {
69      ngx_regex_t                  *regex;
70      ngx_uint_t                    ncaptures;
71      ngx_http_regex_variable_t    *variables;
72      ngx_uint_t                    nvariables;
73      ngx_str_t                     name;
74  } ngx_http_regex_t;
75
76
77  typedef struct {
78      ngx_http_regex_t             *regex;
79      void                         *value;
80  } ngx_http_map_regex_t;
81
82
83  ngx_http_regex_t *ngx_http_regex_compile(ngx_conf_t *cf,
84      ngx_regex_compile_t *rc);
85  ngx_int_t ngx_http_regex_exec(ngx_http_request_t *r, ngx_http_regex_t *re,
86      ngx_str_t *s);
87
88  #endif
89
90
91  typedef struct {
92      ngx_hash_combined_t           hash;
93  #if (NGX_PCRE)
94      ngx_http_map_regex_t         *regex;
95      ngx_uint_t                    nregex;
96  #endif
97  } ngx_http_map_t;
98
99
100  void *ngx_http_map_find(ngx_http_request_t *r, ngx_http_map_t *map,
101      ngx_str_t *match);
102
103
```

```
104 ngx_int_t ngx_http_variables_add_core_vars(ngx_conf_t *cf);
105 ngx_int_t ngx_http_variables_init_vars(ngx_conf_t *cf);
106
107
108 extern ngx_http_variable_value_t  ngx_http_variable_null_value;
109 extern ngx_http_variable_value_t  ngx_http_variable_true_value;
110
111
112 #endif /* _NGX_HTTP_VARIABLES_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_variables.c - nginx-1.7.10

Global variables defined

- ngx_http_core_variables
- ngx_http_variable_null_value
- ngx_http_variable_true_value

Functions defined

- ngx_http_add_variable
- ngx_http_get_flushed_variable
- ngx_http_get_indexed_variable
- ngx_http_get_variable
- ngx_http_get_variable_index
- ngx_http_map_find
- ngx_http_regex_compile
- ngx_http_regex_exec
- ngx_http_variable_argument
- ngx_http_variable_binary_remote_addr
- ngx_http_variable_body_bytes_sent
- ngx_http_variable_bytes_sent
- ngx_http_variable_connection
- ngx_http_variable_connection_requests
- ngx_http_variable_content_length
- ngx_http_variable_cookie
- ngx_http_variable_cookies
- ngx_http_variable_document_root
- ngx_http_variable_header
- ngx_http_variable_headers
- ngx_http_variable_headers_internal
- ngx_http_variable_host
- ngx_http_variable_hostname
- ngx_http_variable_https
- ngx_http_variable_is_args

- ngx_http_variable_msec
- ngx_http_variable_nginx_version
- ngx_http_variable_not_found
- ngx_http_variable_pid
- ngx_http_variable_pipe
- ngx_http_variable_proxy_protocol_addr
- ngx_http_variable_realpath_root
- ngx_http_variable_remote_addr
- ngx_http_variable_remote_port
- ngx_http_variable_remote_user
- ngx_http_variable_request
- ngx_http_variable_request_body
- ngx_http_variable_request_body_file
- ngx_http_variable_request_completion
- ngx_http_variable_request_filename
- ngx_http_variable_request_get_size
- ngx_http_variable_request_length
- ngx_http_variable_request_line
- ngx_http_variable_request_method
- ngx_http_variable_request_set_size
- ngx_http_variable_request_time
- ngx_http_variable_scheme
- ngx_http_variable_sent_connection
- ngx_http_variable_sent_content_length
- ngx_http_variable_sent_content_type
- ngx_http_variable_sent_keep_alive
- ngx_http_variable_sent_last_modified
- ngx_http_variable_sent_location
- ngx_http_variable_sent_transfer_encoding
- ngx_http_variable_server_addr
- ngx_http_variable_server_name
- ngx_http_variable_server_port
- ngx_http_variable_set_args

- ngx_http_variable_status
- ngx_http_variable_tcpinfo
- ngx_http_variable_time_iso8601
- ngx_http_variable_time_local
- ngx_http_variable_unknown_header
- ngx_http_variable_unknown_header_in
- ngx_http_variable_unknown_header_out
- ngx_http_variables_add_core_vars
- ngx_http_variables_init_vars

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <nginx.h>
12
13
14  static ngx_int_t ngx_http_variable_request(ngx_http_request_t *r,
15      ngx_http_variable_value_t *v, uintptr_t data);
16  #if 0
17  static void ngx_http_variable_request_set(ngx_http_request_t *r,
18      ngx_http_variable_value_t *v, uintptr_t data);
19  #endif
20  static ngx_int_t ngx_http_variable_request_get_size(ngx_http_request_t *r,
21      ngx_http_variable_value_t *v, uintptr_t data);
22  static void ngx_http_variable_request_set_size(ngx_http_request_t *r,
23      ngx_http_variable_value_t *v, uintptr_t data);
24  static ngx_int_t ngx_http_variable_header(ngx_http_request_t *r,
25      ngx_http_variable_value_t *v, uintptr_t data);
26
27  static ngx_int_t ngx_http_variable_cookies(ngx_http_request_t *r,
28      ngx_http_variable_value_t *v, uintptr_t data);
29  static ngx_int_t ngx_http_variable_headers(ngx_http_request_t *r,
30      ngx_http_variable_value_t *v, uintptr_t data);
31  static ngx_int_t ngx_http_variable_headers_internal(ngx_http_request_t *r,
32      ngx_http_variable_value_t *v, uintptr_t data, u_char sep);
33
34  static ngx_int_t ngx_http_variable_unknown_header_in(ngx_http_request_t *r,
35      ngx_http_variable_value_t *v, uintptr_t data);
36  static ngx_int_t ngx_http_variable_unknown_header_out(ngx_http_request_t *r,
37      ngx_http_variable_value_t *v, uintptr_t data);
38  static ngx_int_t ngx_http_variable_request_line(ngx_http_request_t *r,
39      ngx_http_variable_value_t *v, uintptr_t data);
40  static ngx_int_t ngx_http_variable_cookie(ngx_http_request_t *r,
41      ngx_http_variable_value_t *v, uintptr_t data);
42  static ngx_int_t ngx_http_variable_argument(ngx_http_request_t *r,
43      ngx_http_variable_value_t *v, uintptr_t data);
44  #if (NGX_HAVE_TCP_INFO)
45  static ngx_int_t ngx_http_variable_tcpinfo(ngx_http_request_t *r,
46      ngx_http_variable_value_t *v, uintptr_t data);
47  #endif
48
49  static ngx_int_t ngx_http_variable_content_length(ngx_http_request_t *r,
50      ngx_http_variable_value_t *v, uintptr_t data);
51  static ngx_int_t ngx_http_variable_host(ngx_http_request_t *r,
52      ngx_http_variable_value_t *v, uintptr_t data);
```

```
static ngx_int_t ngx_http_variable_binary_remote_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_remote_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_remote_port(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_proxy_protocol_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_server_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_server_port(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_scheme(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_https(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static void ngx_http_variable_set_args(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_is_args(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_document_root(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_realpath_root(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_filename(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_server_name(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_method(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_remote_user(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_bytes_sent(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_body_bytes_sent(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_pipe(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_completion(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_body(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_body_file(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_length(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_request_time(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_status(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);

static ngx_int_t ngx_http_variable_sent_content_type(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_sent_content_length(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_sent_location(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_sent_last_modified(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_sent_connection(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_sent_keep_alive(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_sent_transfer_encoding(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);

static ngx_int_t ngx_http_variable_connection(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_connection_requests(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);

static ngx_int_t ngx_http_variable_nginx_version(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_hostname(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_pid(ngx_http_request_t *r,
```

```
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_msec(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_time_iso8601(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_variable_time_local(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);

/*
* TODO:
*     Apache CGI: AUTH_TYPE, PATH_INFO (null), PATH_TRANSLATED
*                 REMOTE_HOST (null), REMOTE_IDENT (null),
*                 SERVER_SOFTWARE
*
*     Apache SSI: DOCUMENT_NAME, LAST_MODIFIED, USER_NAME (file owner)
*/

/*
* the $http_host, $http_user_agent, $http_referer, and $http_via
* variables may be handled by generic
* ngx_http_variable_unknown_header_in(), but for performance reasons
* they are handled using dedicated entries
*/

static ngx_http_variable_t  ngx_http_core_variables[] = {

    { ngx_string("http_host"), NULL, ngx_http_variable_header,
      offsetof(ngx_http_request_t, headers_in.host), 0, 0 },

    { ngx_string("http_user_agent"), NULL, ngx_http_variable_header,
      offsetof(ngx_http_request_t, headers_in.user_agent), 0, 0 },

    { ngx_string("http_referer"), NULL, ngx_http_variable_header,
      offsetof(ngx_http_request_t, headers_in.referer), 0, 0 },

#if (NGX_HTTP_GZIP)
    { ngx_string("http_via"), NULL, ngx_http_variable_header,
      offsetof(ngx_http_request_t, headers_in.via), 0, 0 },
#endif

#if (NGX_HTTP_X_FORWARDED_FOR)
    { ngx_string("http_x_forwarded_for"), NULL, ngx_http_variable_headers,
      offsetof(ngx_http_request_t, headers_in.x_forwarded_for), 0, 0 },
#endif

    { ngx_string("http_cookie"), NULL, ngx_http_variable_cookies,
      offsetof(ngx_http_request_t, headers_in.cookies), 0, 0 },

    { ngx_string("content_length"), NULL, ngx_http_variable_content_length,
      0, 0, 0 },

    { ngx_string("content_type"), NULL, ngx_http_variable_header,
      offsetof(ngx_http_request_t, headers_in.content_type), 0, 0 },

    { ngx_string("host"), NULL, ngx_http_variable_host, 0, 0, 0 },

    { ngx_string("binary_remote_addr"), NULL,
      ngx_http_variable_binary_remote_addr, 0, 0, 0 },

    { ngx_string("remote_addr"), NULL, ngx_http_variable_remote_addr, 0, 0, 0 },

    { ngx_string("remote_port"), NULL, ngx_http_variable_remote_port, 0, 0, 0 },

    { ngx_string("proxy_protocol_addr"), NULL,
      ngx_http_variable_proxy_protocol_addr, 0, 0, 0 },

    { ngx_string("server_addr"), NULL, ngx_http_variable_server_addr, 0, 0, 0 },

    { ngx_string("server_port"), NULL, ngx_http_variable_server_port, 0, 0, 0 },

    { ngx_string("server_protocol"), NULL, ngx_http_variable_request,
      offsetof(ngx_http_request_t, http_protocol), 0, 0 },

    { ngx_string("scheme"), NULL, ngx_http_variable_scheme, 0, 0, 0 },

    { ngx_string("https"), NULL, ngx_http_variable_https, 0, 0, 0 },
```

```
205
206     { ngx_string("request_uri"), NULL, ngx_http_variable_request,
207       offsetof(ngx_http_request_t, unparsed_uri), 0, 0 },
208
209     { ngx_string("uri"), NULL, ngx_http_variable_request,
210       offsetof(ngx_http_request_t, uri),
211       NGX_HTTP_VAR_NOCACHEABLE, 0 },
212
213     { ngx_string("document_uri"), NULL, ngx_http_variable_request,
214       offsetof(ngx_http_request_t, uri),
215       NGX_HTTP_VAR_NOCACHEABLE, 0 },
216
217     { ngx_string("request"), NULL, ngx_http_variable_request_line, 0, 0, 0 },
218
219     { ngx_string("document_root"), NULL,
220       ngx_http_variable_document_root, 0, NGX_HTTP_VAR_NOCACHEABLE, 0 },
221
222     { ngx_string("realpath_root"), NULL,
223       ngx_http_variable_realpath_root, 0, NGX_HTTP_VAR_NOCACHEABLE, 0 },
224
225     { ngx_string("query_string"), NULL, ngx_http_variable_request,
226       offsetof(ngx_http_request_t, args),
227       NGX_HTTP_VAR_NOCACHEABLE, 0 },
228
229     { ngx_string("args"),
230       ngx_http_variable_set_args,
231       ngx_http_variable_request,
232       offsetof(ngx_http_request_t, args),
233       NGX_HTTP_VAR_CHANGEABLE|NGX_HTTP_VAR_NOCACHEABLE, 0 },
234
235     { ngx_string("is_args"), NULL, ngx_http_variable_is_args,
236       0, NGX_HTTP_VAR_NOCACHEABLE, 0 },
237
238     { ngx_string("request_filename"), NULL,
239       ngx_http_variable_request_filename, 0,
240       NGX_HTTP_VAR_NOCACHEABLE, 0 },
241
242     { ngx_string("server_name"), NULL, ngx_http_variable_server_name, 0, 0, 0 },
243
244     { ngx_string("request_method"), NULL,
245       ngx_http_variable_request_method, 0,
246       NGX_HTTP_VAR_NOCACHEABLE, 0 },
247
248     { ngx_string("remote_user"), NULL, ngx_http_variable_remote_user, 0, 0, 0 },
249
250     { ngx_string("bytes_sent"), NULL, ngx_http_variable_bytes_sent,
251       0, 0, 0 },
252
253     { ngx_string("body_bytes_sent"), NULL, ngx_http_variable_body_bytes_sent,
254       0, 0, 0 },
255
256     { ngx_string("pipe"), NULL, ngx_http_variable_pipe,
257       0, 0, 0 },
258
259     { ngx_string("request_completion"), NULL,
260       ngx_http_variable_request_completion,
261       0, 0, 0 },
262
263     { ngx_string("request_body"), NULL,
264       ngx_http_variable_request_body,
265       0, 0, 0 },
266
267     { ngx_string("request_body_file"), NULL,
268       ngx_http_variable_request_body_file,
269       0, 0, 0 },
270
271     { ngx_string("request_length"), NULL, ngx_http_variable_request_length,
272       0, NGX_HTTP_VAR_NOCACHEABLE, 0 },
273
274     { ngx_string("request_time"), NULL, ngx_http_variable_request_time,
275       0, NGX_HTTP_VAR_NOCACHEABLE, 0 },
276
277     { ngx_string("status"), NULL,
278       ngx_http_variable_status, 0,
279       NGX_HTTP_VAR_NOCACHEABLE, 0 },
280
```

```
    { ngx_string("sent_http_content_type"), NULL,
      ngx_http_variable_sent_content_type, 0, 0, 0 },

    { ngx_string("sent_http_content_length"), NULL,
      ngx_http_variable_sent_content_length, 0, 0, 0 },

    { ngx_string("sent_http_location"), NULL,
      ngx_http_variable_sent_location, 0, 0, 0 },

    { ngx_string("sent_http_last_modified"), NULL,
      ngx_http_variable_sent_last_modified, 0, 0, 0 },

    { ngx_string("sent_http_connection"), NULL,
      ngx_http_variable_sent_connection, 0, 0, 0 },

    { ngx_string("sent_http_keep_alive"), NULL,
      ngx_http_variable_sent_keep_alive, 0, 0, 0 },

    { ngx_string("sent_http_transfer_encoding"), NULL,
      ngx_http_variable_sent_transfer_encoding, 0, 0, 0 },

    { ngx_string("sent_http_cache_control"), NULL, ngx_http_variable_headers,
      offsetof(ngx_http_request_t, headers_out.cache_control), 0, 0 },

    { ngx_string("limit_rate"), ngx_http_variable_request_set_size,
      ngx_http_variable_request_get_size,
      offsetof(ngx_http_request_t, limit_rate),
      NGX_HTTP_VAR_CHANGEABLE|NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("connection"), NULL,
      ngx_http_variable_connection, 0, 0, 0 },

    { ngx_string("connection_requests"), NULL,
      ngx_http_variable_connection_requests, 0, 0, 0 },

    { ngx_string("nginx_version"), NULL, ngx_http_variable_nginx_version,
      0, 0, 0 },

    { ngx_string("hostname"), NULL, ngx_http_variable_hostname,
      0, 0, 0 },

    { ngx_string("pid"), NULL, ngx_http_variable_pid,
      0, 0, 0 },

    { ngx_string("msec"), NULL, ngx_http_variable_msec,
      0, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("time_iso8601"), NULL, ngx_http_variable_time_iso8601,
      0, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("time_local"), NULL, ngx_http_variable_time_local,
      0, NGX_HTTP_VAR_NOCACHEABLE, 0 },

#if (NGX_HAVE_TCP_INFO)
    { ngx_string("tcpinfo_rtt"), NULL, ngx_http_variable_tcpinfo,
      0, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("tcpinfo_rttvar"), NULL, ngx_http_variable_tcpinfo,
      1, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("tcpinfo_snd_cwnd"), NULL, ngx_http_variable_tcpinfo,
      2, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("tcpinfo_rcv_space"), NULL, ngx_http_variable_tcpinfo,
      3, NGX_HTTP_VAR_NOCACHEABLE, 0 },
#endif

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


ngx_http_variable_value_t  ngx_http_variable_null_value =
    ngx_http_variable("");
ngx_http_variable_value_t  ngx_http_variable_true_value =
    ngx_http_variable("1");
```

```
357
358 ngx_http_variable_t *
359 ngx_http_add_variable(ngx_conf_t *cf, ngx_str_t *name, ngx_uint_t flags)
360 {
361     ngx_int_t                   rc;
362     ngx_uint_t                  i;
363     ngx_hash_key_t             *key;
364     ngx_http_variable_t        *v;
365     ngx_http_core_main_conf_t  *cmcf;
366
367     if (name->len == 0) {
368         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
369                            "invalid variable name \"$\"");
370         return NULL;
371     }
372
373     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
374
375     key = cmcf->variables_keys->keys.elts;
376     for (i = 0; i < cmcf->variables_keys->keys.nelts; i++) {
377         if (name->len != key[i].key.len
378             || ngx_strncasecmp(name->data, key[i].key.data, name->len) != 0)
379         {
380             continue;
381         }
382
383         v = key[i].value;
384
385         if (!(v->flags & NGX_HTTP_VAR_CHANGEABLE)) {
386             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
387                                "the duplicate \"%V\" variable", name);
388             return NULL;
389         }
390
391         return v;
392     }
393
394     v = ngx_palloc(cf->pool, sizeof(ngx_http_variable_t));
395     if (v == NULL) {
396         return NULL;
397     }
398
399     v->name.len = name->len;
400     v->name.data = ngx_pnalloc(cf->pool, name->len);
401     if (v->name.data == NULL) {
402         return NULL;
403     }
404
405     ngx_strlow(v->name.data, name->data, name->len);
406
407     v->set_handler = NULL;
408     v->get_handler = NULL;
409     v->data = 0;
410     v->flags = flags;
411     v->index = 0;
412
413     rc = ngx_hash_add_key(cmcf->variables_keys, &v->name, v, 0);
414
415     if (rc == NGX_ERROR) {
416         return NULL;
417     }
418
419     if (rc == NGX_BUSY) {
420         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
421                            "conflicting variable name \"%V\"", name);
422         return NULL;
423     }
424
425     return v;
426 }
427
428
429 ngx_int_t
430 ngx_http_get_variable_index(ngx_conf_t *cf, ngx_str_t *name)
431 {
432     ngx_uint_t                  i;
```

```
    ngx_http_variable_t        *v;
    ngx_http_core_main_conf_t  *cmcf;

    if (name->len == 0) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid variable name \"$\"");
        return NGX_ERROR;
    }

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    v = cmcf->variables.elts;

    if (v == NULL) {
        if (ngx_array_init(&cmcf->variables, cf->pool, 4,
                           sizeof(ngx_http_variable_t))
            != NGX_OK)
        {
            return NGX_ERROR;
        }

    } else {
        for (i = 0; i < cmcf->variables.nelts; i++) {
            if (name->len != v[i].name.len
                || ngx_strncasecmp(name->data, v[i].name.data, name->len) != 0)
            {
                continue;
            }

            return i;
        }
    }

    v = ngx_array_push(&cmcf->variables);
    if (v == NULL) {
        return NGX_ERROR;
    }

    v->name.len = name->len;
    v->name.data = ngx_pnalloc(cf->pool, name->len);
    if (v->name.data == NULL) {
        return NGX_ERROR;
    }

    ngx_strlow(v->name.data, name->data, name->len);

    v->set_handler = NULL;
    v->get_handler = NULL;
    v->data = 0;
    v->flags = 0;
    v->index = cmcf->variables.nelts - 1;

    return v->index;
}


ngx_http_variable_value_t *
ngx_http_get_indexed_variable(ngx_http_request_t *r, ngx_uint_t index)
{
    ngx_http_variable_t        *v;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    if (cmcf->variables.nelts <= index) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      "unknown variable index: %ui", index);
        return NULL;
    }

    if (r->variables[index].not_found || r->variables[index].valid) {
        return &r->variables[index];
    }

    v = cmcf->variables.elts;

```

```
    if (v[index].get_handler(r, &r->variables[index], v[index].data)
        == NGX_OK)
    {
        if (v[index].flags & NGX_HTTP_VAR_NOCACHEABLE) {
            r->variables[index].no_cacheable = 1;
        }

        return &r->variables[index];
    }

    r->variables[index].valid = 0;
    r->variables[index].not_found = 1;

    return NULL;
}


ngx_http_variable_value_t *
ngx_http_get_flushed_variable(ngx_http_request_t *r, ngx_uint_t index)
{
    ngx_http_variable_value_t  *v;

    v = &r->variables[index];

    if (v->valid || v->not_found) {
        if (!v->no_cacheable) {
            return v;
        }

        v->valid = 0;
        v->not_found = 0;
    }

    return ngx_http_get_indexed_variable(r, index);
}


ngx_http_variable_value_t *
ngx_http_get_variable(ngx_http_request_t *r, ngx_str_t *name, ngx_uint_t key)
{
    ngx_http_variable_t        *v;
    ngx_http_variable_value_t  *vv;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    v = ngx_hash_find(&cmcf->variables_hash, key, name->data, name->len);

    if (v) {
        if (v->flags & NGX_HTTP_VAR_INDEXED) {
            return ngx_http_get_flushed_variable(r, v->index);

        } else {

            vv = ngx_palloc(r->pool, sizeof(ngx_http_variable_value_t));

            if (vv && v->get_handler(r, vv, v->data) == NGX_OK) {
                return vv;
            }

            return NULL;
        }
    }

    vv = ngx_palloc(r->pool, sizeof(ngx_http_variable_value_t));
    if (vv == NULL) {
        return NULL;
    }

    if (ngx_strncmp(name->data, "http_", 5) == 0) {

        if (ngx_http_variable_unknown_header_in(r, vv, (uintptr_t) name)
            == NGX_OK)
        {
            return vv;
        }
```

```
585
586         return NULL;
587     }
588
589     if (ngx_strncmp(name->data, "sent_http_", 10) == 0) {
590
591         if (ngx_http_variable_unknown_header_out(r, vv, (uintptr_t) name)
592             == NGX_OK)
593         {
594             return vv;
595         }
596
597         return NULL;
598     }
599
600     if (ngx_strncmp(name->data, "upstream_http_", 14) == 0) {
601
602         if (ngx_http_upstream_header_variable(r, vv, (uintptr_t) name)
603             == NGX_OK)
604         {
605             return vv;
606         }
607
608         return NULL;
609     }
610
611     if (ngx_strncmp(name->data, "cookie_", 7) == 0) {
612
613         if (ngx_http_variable_cookie(r, vv, (uintptr_t) name) == NGX_OK) {
614             return vv;
615         }
616
617         return NULL;
618     }
619
620     if (ngx_strncmp(name->data, "upstream_cookie_", 16) == 0) {
621
622         if (ngx_http_upstream_cookie_variable(r, vv, (uintptr_t) name)
623             == NGX_OK)
624         {
625             return vv;
626         }
627
628         return NULL;
629     }
630
631     if (ngx_strncmp(name->data, "arg_", 4) == 0) {
632
633         if (ngx_http_variable_argument(r, vv, (uintptr_t) name) == NGX_OK) {
634             return vv;
635         }
636
637         return NULL;
638     }
639
640     vv->not_found = 1;
641
642     return vv;
643 }
644
645
646 static ngx_int_t
647 ngx_http_variable_request(ngx_http_request_t *r, ngx_http_variable_value_t *v,
648     uintptr_t data)
649 {
650     ngx_str_t  *s;
651
652     s = (ngx_str_t *) ((char *) r + data);
653
654     if (s->data) {
655         v->len = s->len;
656         v->valid = 1;
657         v->no_cacheable = 0;
658         v->not_found = 0;
659         v->data = s->data;
660
```

```
    } else {
        v->not_found = 1;
    }

    return NGX_OK;
}


#if 0

static void
ngx_http_variable_request_set(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_str_t  *s;

    s = (ngx_str_t *) ((char *) r + data);

    s->len = v->len;
    s->data = v->data;
}

#endif


static ngx_int_t
ngx_http_variable_request_get_size(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    size_t  *sp;

    sp = (size_t *) ((char *) r + data);

    v->data = ngx_pnalloc(r->pool, NGX_SIZE_T_LEN);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    v->len = ngx_sprintf(v->data, "%uz", *sp) - v->data;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    return NGX_OK;
}


static void
ngx_http_variable_request_set_size(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ssize_t    s, *sp;
    ngx_str_t  val;

    val.len = v->len;
    val.data = v->data;

    s = ngx_parse_size(&val);

    if (s == NGX_ERROR) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "invalid size \"%V\"", &val);
        return;
    }

    sp = (ssize_t *) ((char *) r + data);

    *sp = s;

    return;
}


static ngx_int_t
ngx_http_variable_header(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
```

```
{
    ngx_table_elt_t  *h;

    h = *(ngx_table_elt_t **) ((char *) r + data);

    if (h) {
        v->len = h->value.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = h->value.data;

    } else {
        v->not_found = 1;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_cookies(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    return ngx_http_variable_headers_internal(r, v, data, ';');
}


static ngx_int_t
ngx_http_variable_headers(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    return ngx_http_variable_headers_internal(r, v, data, ',');
}


static ngx_int_t
ngx_http_variable_headers_internal(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data, u_char sep)
{
    size_t             len;
    u_char            *p, *end;
    ngx_uint_t         i, n;
    ngx_array_t       *a;
    ngx_table_elt_t  **h;

    a = (ngx_array_t *) ((char *) r + data);

    n = a->nelts;
    h = a->elts;

    len = 0;

    for (i = 0; i < n; i++) {

        if (h[i]->hash == 0) {
            continue;
        }

        len += h[i]->value.len + 2;
    }

    if (len == 0) {
        v->not_found = 1;
        return NGX_OK;
    }

    len -= 2;

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    if (n == 1) {
        v->len = (*h)->value.len;
        v->data = (*h)->value.data;
```

```
813
814         return NGX_OK;
815     }
816
817     p = ngx_pnalloc(r->pool, len);
818     if (p == NULL) {
819         return NGX_ERROR;
820     }
821
822     v->len = len;
823     v->data = p;
824
825     end = p + len;
826
827     for (i = 0; /* void */ ; i++) {
828
829         if (h[i]->hash == 0) {
830             continue;
831         }
832
833         p = ngx_copy(p, h[i]->value.data, h[i]->value.len);
834
835         if (p == end) {
836             break;
837         }
838
839         *p++ = sep; *p++ = ' ';
840     }
841
842     return NGX_OK;
843 }
844
845
846 static ngx_int_t
847 ngx_http_variable_unknown_header_in(ngx_http_request_t *r,
848     ngx_http_variable_value_t *v, uintptr_t data)
849 {
850     return ngx_http_variable_unknown_header(v, (ngx_str_t *) data,
851                                             &r->headers_in.headers.part,
852                                             sizeof("http_") - 1);
853 }
854
855
856 static ngx_int_t
857 ngx_http_variable_unknown_header_out(ngx_http_request_t *r,
858     ngx_http_variable_value_t *v, uintptr_t data)
859 {
860     return ngx_http_variable_unknown_header(v, (ngx_str_t *) data,
861                                             &r->headers_out.headers.part,
862                                             sizeof("sent_http_") - 1);
863 }
864
865
866 ngx_int_t
867 ngx_http_variable_unknown_header(ngx_http_variable_value_t *v, ngx_str_t *var,
868     ngx_list_part_t *part, size_t prefix)
869 {
870     u_char            ch;
871     ngx_uint_t        i, n;
872     ngx_table_elt_t  *header;
873
874     header = part->elts;
875
876     for (i = 0; /* void */ ; i++) {
877
878         if (i >= part->nelts) {
879             if (part->next == NULL) {
880                 break;
881             }
882
883             part = part->next;
884             header = part->elts;
885             i = 0;
886         }
887
888         if (header[i].hash == 0) {
```

```
            continue;
        }

        for (n = 0; n + prefix < var->len && n < header[i].key.len; n++) {
            ch = header[i].key.data[n];

            if (ch >= 'A' && ch <= 'Z') {
                ch |= 0x20;

            } else if (ch == '-') {
                ch = '_';
            }

            if (var->data[n + prefix] != ch) {
                break;
            }
        }

        if (n + prefix == var->len && n == header[i].key.len) {
            v->len = header[i].value.len;
            v->valid = 1;
            v->no_cacheable = 0;
            v->not_found = 0;
            v->data = header[i].value.data;

            return NGX_OK;
        }
    }

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_request_line(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char  *p, *s;

    s = r->request_line.data;

    if (s == NULL) {
        s = r->request_start;

        if (s == NULL) {
            v->not_found = 1;
            return NGX_OK;
        }

        for (p = s; p < r->header_in->last; p++) {
            if (*p == CR || *p == LF) {
                break;
            }
        }

        r->request_line.len = p - s;
        r->request_line.data = s;
    }

    v->len = r->request_line.len;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = s;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_cookie(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    ngx_str_t *name = (ngx_str_t *) data;
```

```
965
966     ngx_str_t  cookie, s;
967
968     s.len = name->len - (sizeof("cookie_") - 1);
969     s.data = name->data + sizeof("cookie_") - 1;
970
971     if (ngx_http_parse_multi_header_lines(&r->headers_in.cookies, &s, &cookie)
972         == NGX_DECLINED)
973     {
974         v->not_found = 1;
975         return NGX_OK;
976     }
977
978     v->len = cookie.len;
979     v->valid = 1;
980     v->no_cacheable = 0;
981     v->not_found = 0;
982     v->data = cookie.data;
983
984     return NGX_OK;
985 }
986
987
988 static ngx_int_t
989 ngx_http_variable_argument(ngx_http_request_t *r, ngx_http_variable_value_t *v,
990     uintptr_t data)
991 {
992     ngx_str_t *name = (ngx_str_t *) data;
993
994     u_char     *arg;
995     size_t      len;
996     ngx_str_t   value;
997
998     len = name->len - (sizeof("arg_") - 1);
999     arg = name->data + sizeof("arg_") - 1;
1000
1001     if (ngx_http_arg(r, arg, len, &value) != NGX_OK) {
1002         v->not_found = 1;
1003         return NGX_OK;
1004     }
1005
1006     v->data = value.data;
1007     v->len = value.len;
1008     v->valid = 1;
1009     v->no_cacheable = 0;
1010     v->not_found = 0;
1011
1012     return NGX_OK;
1013 }
1014
1015
1016 #if (NGX_HAVE_TCP_INFO)
1017
1018 static ngx_int_t
1019 ngx_http_variable_tcpinfo(ngx_http_request_t *r, ngx_http_variable_value_t *v,
1020     uintptr_t data)
1021 {
1022     struct tcp_info  ti;
1023     socklen_t        len;
1024     uint32_t         value;
1025
1026     len = sizeof(struct tcp_info);
1027     if (getsockopt(r->connection->fd, IPPROTO_TCP, TCP_INFO, &ti, &len) == -1) {
1028         v->not_found = 1;
1029         return NGX_OK;
1030     }
1031
1032     v->data = ngx_pnalloc(r->pool, NGX_INT32_LEN);
1033     if (v->data == NULL) {
1034         return NGX_ERROR;
1035     }
1036
1037     switch (data) {
1038     case 0:
1039         value = ti.tcpi_rtt;
1040         break;
```

```
1041
1042        case 1:
1043            value = ti.tcpi_rttvar;
1044            break;
1045
1046        case 2:
1047            value = ti.tcpi_snd_cwnd;
1048            break;
1049
1050        case 3:
1051            value = ti.tcpi_rcv_space;
1052            break;
1053
1054        /* suppress warning */
1055        default:
1056            value = 0;
1057            break;
1058        }
1059
1060        v->len = ngx_sprintf(v->data, "%uD", value) - v->data;
1061        v->valid = 1;
1062        v->no_cacheable = 0;
1063        v->not_found = 0;
1064
1065        return NGX_OK;
1066    }
1067
1068    #endif
1069
1070
1071    static ngx_int_t
1072    ngx_http_variable_content_length(ngx_http_request_t *r,
1073        ngx_http_variable_value_t *v, uintptr_t data)
1074    {
1075        u_char  *p;
1076
1077        if (r->headers_in.content_length) {
1078            v->len = r->headers_in.content_length->value.len;
1079            v->data = r->headers_in.content_length->value.data;
1080            v->valid = 1;
1081            v->no_cacheable = 0;
1082            v->not_found = 0;
1083
1084        } else if (r->headers_in.content_length_n >= 0) {
1085            p = ngx_pnalloc(r->pool, NGX_OFF_T_LEN);
1086            if (p == NULL) {
1087                return NGX_ERROR;
1088            }
1089
1090            v->len = ngx_sprintf(p, "%O", r->headers_in.content_length_n) - p;
1091            v->data = p;
1092            v->valid = 1;
1093            v->no_cacheable = 0;
1094            v->not_found = 0;
1095
1096        } else {
1097            v->not_found = 1;
1098        }
1099
1100        return NGX_OK;
1101    }
1102
1103
1104    static ngx_int_t
1105    ngx_http_variable_host(ngx_http_request_t *r, ngx_http_variable_value_t *v,
1106        uintptr_t data)
1107    {
1108        ngx_http_core_srv_conf_t  *cscf;
1109
1110        if (r->headers_in.server.len) {
1111            v->len = r->headers_in.server.len;
1112            v->data = r->headers_in.server.data;
1113
1114        } else {
1115            cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
1116
```

```
        v->len = cscf->server_name.len;
        v->data = cscf->server_name.data;
    }

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_binary_remote_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
#endif

    switch (r->connection->sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) r->connection->sockaddr;

        v->len = sizeof(struct in6_addr);
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = sin6->sin6_addr.s6_addr;

        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) r->connection->sockaddr;

        v->len = sizeof(in_addr_t);
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = (u_char *) &sin->sin_addr;

        break;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_remote_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    v->len = r->connection->addr_text.len;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = r->connection->addr_text.data;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_remote_port(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_uint_t            port;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
#endif

```

```
    v->len = 0;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    v->data = ngx_pnalloc(r->pool, sizeof("65535") - 1);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    switch (r->connection->sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) r->connection->sockaddr;
        port = ntohs(sin6->sin6_port);
        break;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
    case AF_UNIX:
        port = 0;
        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) r->connection->sockaddr;
        port = ntohs(sin->sin_port);
        break;
    }

    if (port > 0 && port < 65536) {
        v->len = ngx_sprintf(v->data, "%ui", port) - v->data;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_proxy_protocol_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    v->len = r->connection->proxy_protocol_addr.len;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = r->connection->proxy_protocol_addr.data;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_server_addr(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_str_t  s;
    u_char     addr[NGX_SOCKADDR_STRLEN];

    s.len = NGX_SOCKADDR_STRLEN;
    s.data = addr;

    if (ngx_connection_local_sockaddr(r->connection, &s, 0) != NGX_OK) {
        return NGX_ERROR;
    }

    s.data = ngx_pnalloc(r->pool, s.len);
    if (s.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s.data, addr, s.len);

    v->len = s.len;
    v->valid = 1;
```

```
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = s.data;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_server_port(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_uint_t            port;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
#endif

    v->len = 0;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    if (ngx_connection_local_sockaddr(r->connection, NULL, 0) != NGX_OK) {
        return NGX_ERROR;
    }

    v->data = ngx_pnalloc(r->pool, sizeof("65535") - 1);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    switch (r->connection->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) r->connection->local_sockaddr;
        port = ntohs(sin6->sin6_port);
        break;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
    case AF_UNIX:
        port = 0;
        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) r->connection->local_sockaddr;
        port = ntohs(sin->sin_port);
        break;
    }

    if (port > 0 && port < 65536) {
        v->len = ngx_sprintf(v->data, "%ui", port) - v->data;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_scheme(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
#if (NGX_HTTP_SSL)

    if (r->connection->ssl) {
        v->len = sizeof("https") - 1;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = (u_char *) "https";

        return NGX_OK;
    }
```

```
1345
1346 #endif
1347
1348     v->len = sizeof("http") - 1;
1349     v->valid = 1;
1350     v->no_cacheable = 0;
1351     v->not_found = 0;
1352     v->data = (u_char *) "http";
1353
1354     return NGX_OK;
1355 }
1356
1357
1358 static ngx_int_t
1359 ngx_http_variable_https(ngx_http_request_t *r,
1360     ngx_http_variable_value_t *v, uintptr_t data)
1361 {
1362 #if (NGX_HTTP_SSL)
1363
1364     if (r->connection->ssl) {
1365         v->len = sizeof("on") - 1;
1366         v->valid = 1;
1367         v->no_cacheable = 0;
1368         v->not_found = 0;
1369         v->data = (u_char *) "on";
1370
1371         return NGX_OK;
1372     }
1373
1374 #endif
1375
1376     *v = ngx_http_variable_null_value;
1377
1378     return NGX_OK;
1379 }
1380
1381
1382 static void
1383 ngx_http_variable_set_args(ngx_http_request_t *r,
1384     ngx_http_variable_value_t *v, uintptr_t data)
1385 {
1386     r->args.len = v->len;
1387     r->args.data = v->data;
1388     r->valid_unparsed_uri = 0;
1389 }
1390
1391
1392 static ngx_int_t
1393 ngx_http_variable_is_args(ngx_http_request_t *r,
1394     ngx_http_variable_value_t *v, uintptr_t data)
1395 {
1396     v->valid = 1;
1397     v->no_cacheable = 0;
1398     v->not_found = 0;
1399
1400     if (r->args.len == 0) {
1401         v->len = 0;
1402         v->data = NULL;
1403         return NGX_OK;
1404     }
1405
1406     v->len = 1;
1407     v->data = (u_char *) "?";
1408
1409     return NGX_OK;
1410 }
1411
1412
1413 static ngx_int_t
1414 ngx_http_variable_document_root(ngx_http_request_t *r,
1415     ngx_http_variable_value_t *v, uintptr_t data)
1416 {
1417     ngx_str_t                  path;
1418     ngx_http_core_loc_conf_t  *clcf;
1419
1420     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
```

```
1421
1422      if (clcf->root_lengths == NULL) {
1423          v->len = clcf->root.len;
1424          v->valid = 1;
1425          v->no_cacheable = 0;
1426          v->not_found = 0;
1427          v->data = clcf->root.data;
1428
1429      } else {
1430          if (ngx_http_script_run(r, &path, clcf->root_lengths->elts, 0,
1431                                  clcf->root_values->elts)
1432              == NULL)
1433          {
1434              return NGX_ERROR;
1435          }
1436
1437          if (ngx_get_full_name(r->pool, (ngx_str_t *) &ngx_cycle->prefix, &path)
1438              != NGX_OK)
1439          {
1440              return NGX_ERROR;
1441          }
1442
1443          v->len = path.len;
1444          v->valid = 1;
1445          v->no_cacheable = 0;
1446          v->not_found = 0;
1447          v->data = path.data;
1448      }
1449
1450      return NGX_OK;
1451  }
1452
1453
1454  static ngx_int_t
1455  ngx_http_variable_realpath_root(ngx_http_request_t *r,
1456      ngx_http_variable_value_t *v, uintptr_t data)
1457  {
1458      u_char                     *real;
1459      size_t                      len;
1460      ngx_str_t                   path;
1461      ngx_http_core_loc_conf_t   *clcf;
1462  #if (NGX_HAVE_MAX_PATH)
1463      u_char                      buffer[NGX_MAX_PATH];
1464  #endif
1465
1466      clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
1467
1468      if (clcf->root_lengths == NULL) {
1469          path = clcf->root;
1470
1471      } else {
1472          if (ngx_http_script_run(r, &path, clcf->root_lengths->elts, 1,
1473                                  clcf->root_values->elts)
1474              == NULL)
1475          {
1476              return NGX_ERROR;
1477          }
1478
1479          path.data[path.len - 1] = '\0';
1480
1481          if (ngx_get_full_name(r->pool, (ngx_str_t *) &ngx_cycle->prefix, &path)
1482              != NGX_OK)
1483          {
1484              return NGX_ERROR;
1485          }
1486      }
1487
1488  #if (NGX_HAVE_MAX_PATH)
1489      real = buffer;
1490  #else
1491      real = NULL;
1492  #endif
1493
1494      real = ngx_realpath(path.data, real);
1495
1496      if (real == NULL) {
```

```
1497         ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
1498                       ngx_realpath_n " \"%s\" failed", path.data);
1499         return NGX_ERROR;
1500     }
1501
1502     len = ngx_strlen(real);
1503
1504     v->data = ngx_pnalloc(r->pool, len);
1505     if (v->data == NULL) {
1506 #if !(NGX_HAVE_MAX_PATH)
1507         ngx_free(real);
1508 #endif
1509         return NGX_ERROR;
1510     }
1511
1512     v->len = len;
1513     v->valid = 1;
1514     v->no_cacheable = 0;
1515     v->not_found = 0;
1516
1517     ngx_memcpy(v->data, real, len);
1518
1519 #if !(NGX_HAVE_MAX_PATH)
1520     ngx_free(real);
1521 #endif
1522
1523     return NGX_OK;
1524 }
1525
1526
1527 static ngx_int_t
1528 ngx_http_variable_request_filename(ngx_http_request_t *r,
1529     ngx_http_variable_value_t *v, uintptr_t data)
1530 {
1531     size_t     root;
1532     ngx_str_t  path;
1533
1534     if (ngx_http_map_uri_to_path(r, &path, &root, 0) == NULL) {
1535         return NGX_ERROR;
1536     }
1537
1538     /* ngx_http_map_uri_to_path() allocates memory for terminating '\0' */
1539
1540     v->len = path.len - 1;
1541     v->valid = 1;
1542     v->no_cacheable = 0;
1543     v->not_found = 0;
1544     v->data = path.data;
1545
1546     return NGX_OK;
1547 }
1548
1549
1550 static ngx_int_t
1551 ngx_http_variable_server_name(ngx_http_request_t *r,
1552     ngx_http_variable_value_t *v, uintptr_t data)
1553 {
1554     ngx_http_core_srv_conf_t  *cscf;
1555
1556     cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
1557
1558     v->len = cscf->server_name.len;
1559     v->valid = 1;
1560     v->no_cacheable = 0;
1561     v->not_found = 0;
1562     v->data = cscf->server_name.data;
1563
1564     return NGX_OK;
1565 }
1566
1567
1568 static ngx_int_t
1569 ngx_http_variable_request_method(ngx_http_request_t *r,
1570     ngx_http_variable_value_t *v, uintptr_t data)
1571 {
1572     if (r->main->method_name.data) {
```

```
1573            v->len = r->main->method_name.len;
1574            v->valid = 1;
1575            v->no_cacheable = 0;
1576            v->not_found = 0;
1577            v->data = r->main->method_name.data;
1578
1579        } else {
1580            v->not_found = 1;
1581        }
1582
1583        return NGX_OK;
1584    }
1585
1586
1587    static ngx_int_t
1588    ngx_http_variable_remote_user(ngx_http_request_t *r,
1589        ngx_http_variable_value_t *v, uintptr_t data)
1590    {
1591        ngx_int_t  rc;
1592
1593        rc = ngx_http_auth_basic_user(r);
1594
1595        if (rc == NGX_DECLINED) {
1596            v->not_found = 1;
1597            return NGX_OK;
1598        }
1599
1600        if (rc == NGX_ERROR) {
1601            return NGX_ERROR;
1602        }
1603
1604        v->len = r->headers_in.user.len;
1605        v->valid = 1;
1606        v->no_cacheable = 0;
1607        v->not_found = 0;
1608        v->data = r->headers_in.user.data;
1609
1610        return NGX_OK;
1611    }
1612
1613
1614    static ngx_int_t
1615    ngx_http_variable_bytes_sent(ngx_http_request_t *r,
1616        ngx_http_variable_value_t *v, uintptr_t data)
1617    {
1618        u_char  *p;
1619
1620        p = ngx_pnalloc(r->pool, NGX_OFF_T_LEN);
1621        if (p == NULL) {
1622            return NGX_ERROR;
1623        }
1624
1625        v->len = ngx_sprintf(p, "%O", r->connection->sent) - p;
1626        v->valid = 1;
1627        v->no_cacheable = 0;
1628        v->not_found = 0;
1629        v->data = p;
1630
1631        return NGX_OK;
1632    }
1633
1634
1635    static ngx_int_t
1636    ngx_http_variable_body_bytes_sent(ngx_http_request_t *r,
1637        ngx_http_variable_value_t *v, uintptr_t data)
1638    {
1639        off_t    sent;
1640        u_char  *p;
1641
1642        sent = r->connection->sent - r->header_size;
1643
1644        if (sent < 0) {
1645            sent = 0;
1646        }
1647
1648        p = ngx_pnalloc(r->pool, NGX_OFF_T_LEN);
```

```
    if (p == NULL) {
        return NGX_ERROR;
    }

    v->len = ngx_sprintf(p, "%O", sent) - p;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = p;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_pipe(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    v->data = (u_char *) (r->pipeline ? "p" : ".");
    v->len = 1;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_status(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_uint_t  status;

    v->data = ngx_pnalloc(r->pool, NGX_INT_T_LEN);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    if (r->err_status) {
        status = r->err_status;

    } else if (r->headers_out.status) {
        status = r->headers_out.status;

    } else if (r->http_version == NGX_HTTP_VERSION_9) {
        status = 9;

    } else {
        status = 0;
    }

    v->len = ngx_sprintf(v->data, "%03ui", status) - v->data;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_sent_content_type(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    if (r->headers_out.content_type.len) {
        v->len = r->headers_out.content_type.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = r->headers_out.content_type.data;

    } else {
        v->not_found = 1;
    }

```

```
    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_sent_content_length(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char  *p;

    if (r->headers_out.content_length) {
        v->len = r->headers_out.content_length->value.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = r->headers_out.content_length->value.data;

        return NGX_OK;
    }

    if (r->headers_out.content_length_n >= 0) {
        p = ngx_pnalloc(r->pool, NGX_OFF_T_LEN);
        if (p == NULL) {
            return NGX_ERROR;
        }

        v->len = ngx_sprintf(p, "%O", r->headers_out.content_length_n) - p;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = p;

        return NGX_OK;
    }

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_sent_location(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_str_t  name;

    if (r->headers_out.location) {
        v->len = r->headers_out.location->value.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = r->headers_out.location->value.data;

        return NGX_OK;
    }

    ngx_str_set(&name, "sent_http_location");

    return ngx_http_variable_unknown_header(v, &name,
                                            &r->headers_out.headers.part,
                                            sizeof("sent_http_") - 1);
}


static ngx_int_t
ngx_http_variable_sent_last_modified(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char  *p;

    if (r->headers_out.last_modified) {
        v->len = r->headers_out.last_modified->value.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
```

```
1801         v->data = r->headers_out.last_modified->value.data;
1802
1803         return NGX_OK;
1804     }
1805
1806     if (r->headers_out.last_modified_time >= 0) {
1807         p = ngx_pnalloc(r->pool, sizeof("Mon, 28 Sep 1970 06:00:00 GMT") - 1);
1808         if (p == NULL) {
1809             return NGX_ERROR;
1810         }
1811
1812         v->len = ngx_http_time(p, r->headers_out.last_modified_time) - p;
1813         v->valid = 1;
1814         v->no_cacheable = 0;
1815         v->not_found = 0;
1816         v->data = p;
1817
1818         return NGX_OK;
1819     }
1820
1821     v->not_found = 1;
1822
1823     return NGX_OK;
1824 }
1825
1826
1827 static ngx_int_t
1828 ngx_http_variable_sent_connection(ngx_http_request_t *r,
1829     ngx_http_variable_value_t *v, uintptr_t data)
1830 {
1831     size_t   len;
1832     char    *p;
1833
1834     if (r->headers_out.status == NGX_HTTP_SWITCHING_PROTOCOLS) {
1835         len = sizeof("upgrade") - 1;
1836         p = "upgrade";
1837
1838     } else if (r->keepalive) {
1839         len = sizeof("keep-alive") - 1;
1840         p = "keep-alive";
1841
1842     } else {
1843         len = sizeof("close") - 1;
1844         p = "close";
1845     }
1846
1847     v->len = len;
1848     v->valid = 1;
1849     v->no_cacheable = 0;
1850     v->not_found = 0;
1851     v->data = (u_char *) p;
1852
1853     return NGX_OK;
1854 }
1855
1856
1857 static ngx_int_t
1858 ngx_http_variable_sent_keep_alive(ngx_http_request_t *r,
1859     ngx_http_variable_value_t *v, uintptr_t data)
1860 {
1861     u_char                    *p;
1862     ngx_http_core_loc_conf_t  *clcf;
1863
1864     if (r->keepalive) {
1865         clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
1866
1867         if (clcf->keepalive_header) {
1868
1869             p = ngx_pnalloc(r->pool, sizeof("timeout=") - 1 + NGX_TIME_T_LEN);
1870             if (p == NULL) {
1871                 return NGX_ERROR;
1872             }
1873
1874             v->len = ngx_sprintf(p, "timeout=%T", clcf->keepalive_header) - p;
1875             v->valid = 1;
1876             v->no_cacheable = 0;
```

```
            v->not_found = 0;
            v->data = p;

            return NGX_OK;
        }
    }

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_sent_transfer_encoding(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    if (r->chunked) {
        v->len = sizeof("chunked") - 1;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = (u_char *) "chunked";

    } else {
        v->not_found = 1;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_request_completion(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    if (r->request_complete) {
        v->len = 2;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = (u_char *) "OK";

        return NGX_OK;
    }

    v->len = 0;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = (u_char *) "";

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_request_body(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char       *p;
    size_t        len;
    ngx_buf_t    *buf;
    ngx_chain_t  *cl;

    if (r->request_body == NULL
        || r->request_body->bufs == NULL
        || r->request_body->temp_file)
    {
        v->not_found = 1;

        return NGX_OK;
    }

    cl = r->request_body->bufs;
    buf = cl->buf;
```

```
1953
1954        if (cl->next == NULL) {
1955            v->len = buf->last - buf->pos;
1956            v->valid = 1;
1957            v->no_cacheable = 0;
1958            v->not_found = 0;
1959            v->data = buf->pos;
1960
1961            return NGX_OK;
1962        }
1963
1964        len = buf->last - buf->pos;
1965        cl = cl->next;
1966
1967        for ( /* void */ ; cl; cl = cl->next) {
1968            buf = cl->buf;
1969            len += buf->last - buf->pos;
1970        }
1971
1972        p = ngx_pnalloc(r->pool, len);
1973        if (p == NULL) {
1974            return NGX_ERROR;
1975        }
1976
1977        v->data = p;
1978        cl = r->request_body->bufs;
1979
1980        for ( /* void */ ; cl; cl = cl->next) {
1981            buf = cl->buf;
1982            p = ngx_cpymem(p, buf->pos, buf->last - buf->pos);
1983        }
1984
1985        v->len = len;
1986        v->valid = 1;
1987        v->no_cacheable = 0;
1988        v->not_found = 0;
1989
1990        return NGX_OK;
1991    }
1992
1993
1994    static ngx_int_t
1995    ngx_http_variable_request_body_file(ngx_http_request_t *r,
1996        ngx_http_variable_value_t *v, uintptr_t data)
1997    {
1998        if (r->request_body == NULL || r->request_body->temp_file == NULL) {
1999            v->not_found = 1;
2000
2001            return NGX_OK;
2002        }
2003
2004        v->len = r->request_body->temp_file->file.name.len;
2005        v->valid = 1;
2006        v->no_cacheable = 0;
2007        v->not_found = 0;
2008        v->data = r->request_body->temp_file->file.name.data;
2009
2010        return NGX_OK;
2011    }
2012
2013
2014    static ngx_int_t
2015    ngx_http_variable_request_length(ngx_http_request_t *r,
2016        ngx_http_variable_value_t *v, uintptr_t data)
2017    {
2018        u_char  *p;
2019
2020        p = ngx_pnalloc(r->pool, NGX_OFF_T_LEN);
2021        if (p == NULL) {
2022            return NGX_ERROR;
2023        }
2024
2025        v->len = ngx_sprintf(p, "%O", r->request_length) - p;
2026        v->valid = 1;
2027        v->no_cacheable = 0;
2028        v->not_found = 0;
```

```
    v->data = p;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_request_time(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char          *p;
    ngx_time_t      *tp;
    ngx_msec_int_t   ms;

    p = ngx_pnalloc(r->pool, NGX_TIME_T_LEN + 4);
    if (p == NULL) {
        return NGX_ERROR;
    }

    tp = ngx_timeofday();

    ms = (ngx_msec_int_t)
             ((tp->sec - r->start_sec) * 1000 + (tp->msec - r->start_msec));
    ms = ngx_max(ms, 0);

    v->len = ngx_sprintf(p, "%T.%03M", (time_t) ms / 1000, ms % 1000) - p;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = p;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_connection(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char  *p;

    p = ngx_pnalloc(r->pool, NGX_ATOMIC_T_LEN);
    if (p == NULL) {
        return NGX_ERROR;
    }

    v->len = ngx_sprintf(p, "%uA", r->connection->number) - p;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = p;

    return NGX_OK;
}


static ngx_int_t
ngx_http_variable_connection_requests(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char  *p;

    p = ngx_pnalloc(r->pool, NGX_INT_T_LEN);
    if (p == NULL) {
        return NGX_ERROR;
    }

    v->len = ngx_sprintf(p, "%ui", r->connection->requests) - p;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = p;

    return NGX_OK;
}

```

```
2105
2106 static ngx_int_t
2107 ngx_http_variable_nginx_version(ngx_http_request_t *r,
2108     ngx_http_variable_value_t *v, uintptr_t data)
2109 {
2110     v->len = sizeof(NGINX_VERSION) - 1;
2111     v->valid = 1;
2112     v->no_cacheable = 0;
2113     v->not_found = 0;
2114     v->data = (u_char *) NGINX_VERSION;
2115
2116     return NGX_OK;
2117 }
2118
2119
2120 static ngx_int_t
2121 ngx_http_variable_hostname(ngx_http_request_t *r,
2122     ngx_http_variable_value_t *v, uintptr_t data)
2123 {
2124     v->len = ngx_cycle->hostname.len;
2125     v->valid = 1;
2126     v->no_cacheable = 0;
2127     v->not_found = 0;
2128     v->data = ngx_cycle->hostname.data;
2129
2130     return NGX_OK;
2131 }
2132
2133
2134 static ngx_int_t
2135 ngx_http_variable_pid(ngx_http_request_t *r,
2136     ngx_http_variable_value_t *v, uintptr_t data)
2137 {
2138     u_char  *p;
2139
2140     p = ngx_pnalloc(r->pool, NGX_INT64_LEN);
2141     if (p == NULL) {
2142         return NGX_ERROR;
2143     }
2144
2145     v->len = ngx_sprintf(p, "%P", ngx_pid) - p;
2146     v->valid = 1;
2147     v->no_cacheable = 0;
2148     v->not_found = 0;
2149     v->data = p;
2150
2151     return NGX_OK;
2152 }
2153
2154
2155 static ngx_int_t
2156 ngx_http_variable_msec(ngx_http_request_t *r,
2157     ngx_http_variable_value_t *v, uintptr_t data)
2158 {
2159     u_char      *p;
2160     ngx_time_t  *tp;
2161
2162     p = ngx_pnalloc(r->pool, NGX_TIME_T_LEN + 4);
2163     if (p == NULL) {
2164         return NGX_ERROR;
2165     }
2166
2167     tp = ngx_timeofday();
2168
2169     v->len = ngx_sprintf(p, "%T.%03M", tp->sec, tp->msec) - p;
2170     v->valid = 1;
2171     v->no_cacheable = 0;
2172     v->not_found = 0;
2173     v->data = p;
2174
2175     return NGX_OK;
2176 }
2177
2178
2179 static ngx_int_t
2180 ngx_http_variable_time_iso8601(ngx_http_request_t *r,
```

```
2181      ngx_http_variable_value_t *v, uintptr_t data)
2182  {
2183      u_char  *p;
2184
2185      p = ngx_pnalloc(r->pool, ngx_cached_http_log_iso8601.len);
2186      if (p == NULL) {
2187          return NGX_ERROR;
2188      }
2189
2190      ngx_memcpy(p, ngx_cached_http_log_iso8601.data,
2191                 ngx_cached_http_log_iso8601.len);
2192
2193      v->len = ngx_cached_http_log_iso8601.len;
2194      v->valid = 1;
2195      v->no_cacheable = 0;
2196      v->not_found = 0;
2197      v->data = p;
2198
2199      return NGX_OK;
2200  }
2201
2202
2203  static ngx_int_t
2204  ngx_http_variable_time_local(ngx_http_request_t *r,
2205      ngx_http_variable_value_t *v, uintptr_t data)
2206  {
2207      u_char  *p;
2208
2209      p = ngx_pnalloc(r->pool, ngx_cached_http_log_time.len);
2210      if (p == NULL) {
2211          return NGX_ERROR;
2212      }
2213
2214      ngx_memcpy(p, ngx_cached_http_log_time.data, ngx_cached_http_log_time.len);
2215
2216      v->len = ngx_cached_http_log_time.len;
2217      v->valid = 1;
2218      v->no_cacheable = 0;
2219      v->not_found = 0;
2220      v->data = p;
2221
2222      return NGX_OK;
2223  }
2224
2225
2226  void *
2227  ngx_http_map_find(ngx_http_request_t *r, ngx_http_map_t *map, ngx_str_t *match)
2228  {
2229      void        *value;
2230      u_char      *low;
2231      size_t       len;
2232      ngx_uint_t   key;
2233
2234      len = match->len;
2235
2236      if (len) {
2237          low = ngx_pnalloc(r->pool, len);
2238          if (low == NULL) {
2239              return NULL;
2240          }
2241
2242      } else {
2243          low = NULL;
2244      }
2245
2246      key = ngx_hash_strlow(low, match->data, len);
2247
2248      value = ngx_hash_find_combined(&map->hash, key, low, len);
2249      if (value) {
2250          return value;
2251      }
2252
2253  #if (NGX_PCRE)
2254
2255      if (len && map->nregex) {
2256          ngx_int_t              n;
```

```
        ngx_uint_t             i;
        ngx_http_map_regex_t  *reg;

        reg = map->regex;

        for (i = 0; i < map->nregex; i++) {

            n = ngx_http_regex_exec(r, reg[i].regex, match);

            if (n == NGX_OK) {
                return reg[i].value;
            }

            if (n == NGX_DECLINED) {
                continue;
            }

            /* NGX_ERROR */

            return NULL;
        }
    }

#endif

    return NULL;
}


#if (NGX_PCRE)

static ngx_int_t
ngx_http_variable_not_found(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    v->not_found = 1;
    return NGX_OK;
}


ngx_http_regex_t *
ngx_http_regex_compile(ngx_conf_t *cf, ngx_regex_compile_t *rc)
{
    u_char                     *p;
    size_t                      size;
    ngx_str_t                   name;
    ngx_uint_t                  i, n;
    ngx_http_variable_t        *v;
    ngx_http_regex_t           *re;
    ngx_http_regex_variable_t  *rv;
    ngx_http_core_main_conf_t  *cmcf;

    rc->pool = cf->pool;

    if (ngx_regex_compile(rc) != NGX_OK) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "%V", &rc->err);
        return NULL;
    }

    re = ngx_pcalloc(cf->pool, sizeof(ngx_http_regex_t));
    if (re == NULL) {
        return NULL;
    }

    re->regex = rc->regex;
    re->ncaptures = rc->captures;
    re->name = rc->pattern;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
    cmcf->ncaptures = ngx_max(cmcf->ncaptures, re->ncaptures);

    n = (ngx_uint_t) rc->named_captures;

    if (n == 0) {
        return re;
    }
```

```
2333
2334        rv = ngx_palloc(rc->pool, n * sizeof(ngx_http_regex_variable_t));
2335        if (rv == NULL) {
2336            return NULL;
2337        }
2338
2339        re->variables = rv;
2340        re->nvariables = n;
2341
2342        size = rc->name_size;
2343        p = rc->names;
2344
2345        for (i = 0; i < n; i++) {
2346            rv[i].capture = 2 * ((p[0] << 8) + p[1]);
2347
2348            name.data = &p[2];
2349            name.len = ngx_strlen(name.data);
2350
2351            v = ngx_http_add_variable(cf, &name, NGX_HTTP_VAR_CHANGEABLE);
2352            if (v == NULL) {
2353                return NULL;
2354            }
2355
2356            rv[i].index = ngx_http_get_variable_index(cf, &name);
2357            if (rv[i].index == NGX_ERROR) {
2358                return NULL;
2359            }
2360
2361            v->get_handler = ngx_http_variable_not_found;
2362
2363            p += size;
2364        }
2365
2366        return re;
2367    }
2368
2369
2370    ngx_int_t
2371    ngx_http_regex_exec(ngx_http_request_t *r, ngx_http_regex_t *re, ngx_str_t *s)
2372    {
2373        ngx_int_t                   rc, index;
2374        ngx_uint_t                  i, n, len;
2375        ngx_http_variable_value_t  *vv;
2376        ngx_http_core_main_conf_t  *cmcf;
2377
2378        cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
2379
2380        if (re->ncaptures) {
2381            len = cmcf->ncaptures;
2382
2383            if (r->captures == NULL) {
2384                r->captures = ngx_palloc(r->pool, len * sizeof(int));
2385                if (r->captures == NULL) {
2386                    return NGX_ERROR;
2387                }
2388            }
2389
2390        } else {
2391            len = 0;
2392        }
2393
2394        rc = ngx_regex_exec(re->regex, s, r->captures, len);
2395
2396        if (rc == NGX_REGEX_NO_MATCHED) {
2397            return NGX_DECLINED;
2398        }
2399
2400        if (rc < 0) {
2401            ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
2402                          ngx_regex_exec_n " failed: %i on \"%V\" using \"%V\"",
2403                          rc, s, &re->name);
2404            return NGX_ERROR;
2405        }
2406
2407        for (i = 0; i < re->nvariables; i++) {
2408
```

```
        n = re->variables[i].capture;
        index = re->variables[i].index;
        vv = &r->variables[index];

        vv->len = r->captures[n + 1] - r->captures[n];
        vv->valid = 1;
        vv->no_cacheable = 0;
        vv->not_found = 0;
        vv->data = &s->data[r->captures[n]];

#if (NGX_DEBUG)
        {
        ngx_http_variable_t  *v;

        v = cmcf->variables.elts;

        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http regex set $%V to \"%*s\"",
                       &v[index].name, vv->len, vv->data);
        }
#endif
    }

    r->ncaptures = rc * 2;
    r->captures_data = s->data;

    return NGX_OK;
}

#endif


ngx_int_t
ngx_http_variables_add_core_vars(ngx_conf_t *cf)
{
    ngx_int_t                   rc;
    ngx_http_variable_t        *cv, *v;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    cmcf->variables_keys = ngx_pcalloc(cf->temp_pool,
                                       sizeof(ngx_hash_keys_arrays_t));
    if (cmcf->variables_keys == NULL) {
        return NGX_ERROR;
    }

    cmcf->variables_keys->pool = cf->pool;
    cmcf->variables_keys->temp_pool = cf->pool;

    if (ngx_hash_keys_array_init(cmcf->variables_keys, NGX_HASH_SMALL)
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    for (cv = ngx_http_core_variables; cv->name.len; cv++) {
        v = ngx_palloc(cf->pool, sizeof(ngx_http_variable_t));
        if (v == NULL) {
            return NGX_ERROR;
        }

        *v = *cv;

        rc = ngx_hash_add_key(cmcf->variables_keys, &v->name, v,
                              NGX_HASH_READONLY_KEY);

        if (rc == NGX_OK) {
            continue;
        }

        if (rc == NGX_BUSY) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "conflicting variable name \"%V\"", &v->name);
        }

```

```
2485            return NGX_ERROR;
2486        }
2487
2488        return NGX_OK;
2489    }
2490
2491
2492    ngx_int_t
2493    ngx_http_variables_init_vars(ngx_conf_t *cf)
2494    {
2495        ngx_uint_t                  i, n;
2496        ngx_hash_key_t             *key;
2497        ngx_hash_init_t             hash;
2498        ngx_http_variable_t        *v, *av;
2499        ngx_http_core_main_conf_t  *cmcf;
2500
2501        /* set the handlers for the indexed http variables */
2502
2503        cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
2504
2505        v = cmcf->variables.elts;
2506        key = cmcf->variables_keys->keys.elts;
2507
2508        for (i = 0; i < cmcf->variables.nelts; i++) {
2509
2510            for (n = 0; n < cmcf->variables_keys->keys.nelts; n++) {
2511
2512                av = key[n].value;
2513
2514                if (v[i].name.len == key[n].key.len
2515                    && ngx_strncmp(v[i].name.data, key[n].key.data, v[i].name.len)
2516                       == 0)
2517                {
2518                    v[i].get_handler = av->get_handler;
2519                    v[i].data = av->data;
2520
2521                    av->flags |= NGX_HTTP_VAR_INDEXED;
2522                    v[i].flags = av->flags;
2523
2524                    av->index = i;
2525
2526                    if (av->get_handler == NULL) {
2527                        break;
2528                    }
2529
2530                    goto next;
2531                }
2532            }
2533
2534            if (ngx_strncmp(v[i].name.data, "http_", 5) == 0) {
2535                v[i].get_handler = ngx_http_variable_unknown_header_in;
2536                v[i].data = (uintptr_t) &v[i].name;
2537
2538                continue;
2539            }
2540
2541            if (ngx_strncmp(v[i].name.data, "sent_http_", 10) == 0) {
2542                v[i].get_handler = ngx_http_variable_unknown_header_out;
2543                v[i].data = (uintptr_t) &v[i].name;
2544
2545                continue;
2546            }
2547
2548            if (ngx_strncmp(v[i].name.data, "upstream_http_", 14) == 0) {
2549                v[i].get_handler = ngx_http_upstream_header_variable;
2550                v[i].data = (uintptr_t) &v[i].name;
2551                v[i].flags = NGX_HTTP_VAR_NOCACHEABLE;
2552
2553                continue;
2554            }
2555
2556            if (ngx_strncmp(v[i].name.data, "cookie_", 7) == 0) {
2557                v[i].get_handler = ngx_http_variable_cookie;
2558                v[i].data = (uintptr_t) &v[i].name;
2559
2560                continue;
```

```
2561            }
2562
2563            if (ngx_strncmp(v[i].name.data, "upstream_cookie_", 16) == 0) {
2564                v[i].get_handler = ngx_http_upstream_cookie_variable;
2565                v[i].data = (uintptr_t) &v[i].name;
2566                v[i].flags = NGX_HTTP_VAR_NOCACHEABLE;
2567
2568                continue;
2569            }
2570
2571            if (ngx_strncmp(v[i].name.data, "arg_", 4) == 0) {
2572                v[i].get_handler = ngx_http_variable_argument;
2573                v[i].data = (uintptr_t) &v[i].name;
2574                v[i].flags = NGX_HTTP_VAR_NOCACHEABLE;
2575
2576                continue;
2577            }
2578
2579            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
2580                          "unknown \"%V\" variable", &v[i].name);
2581
2582            return NGX_ERROR;
2583
2584        next:
2585            continue;
2586        }
2587
2588
2589        for (n = 0; n < cmcf->variables_keys->keys.nelts; n++) {
2590            av = key[n].value;
2591
2592            if (av->flags & NGX_HTTP_VAR_NOHASH) {
2593                key[n].key.data = NULL;
2594            }
2595        }
2596
2597
2598        hash.hash = &cmcf->variables_hash;
2599        hash.key = ngx_hash_key;
2600        hash.max_size = cmcf->variables_hash_max_size;
2601        hash.bucket_size = cmcf->variables_hash_bucket_size;
2602        hash.name = "variables_hash";
2603        hash.pool = cf->pool;
2604        hash.temp_pool = NULL;
2605
2606        if (ngx_hash_init(&hash, cmcf->variables_keys->keys.elts,
2607                          cmcf->variables_keys->keys.nelts)
2608            != NGX_OK)
2609        {
2610            return NGX_ERROR;
2611        }
2612
2613        cmcf->variables_keys = NULL;
2614
2615        return NGX_OK;
2616    }
```

One Level Up Top Level

src/core/nginx.h - nginx-1.7.10

Macros defined

- NGINX_VAR
- NGINX_VER
- NGINX_VERSION
- NGINX_VER_BUILD
- NGINX_VER_BUILD
- NGX_OLDPID_EXT
- _NGINX_H_INCLUDED_
- nginx_version

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGINX_H_INCLUDED_
9  #define _NGINX_H_INCLUDED_
10
11
12  #define nginx_version      1007010
13  #define NGINX_VERSION      "1.7.10"
14  #define NGINX_VER          "nginx/" NGINX_VERSION
15
16  #ifdef NGX_BUILD
17  #define NGINX_VER_BUILD    NGINX_VER " (" NGX_BUILD ")"
18  #else
19  #define NGINX_VER_BUILD    NGINX_VER
20  #endif
21
22  #define NGINX_VAR          "NGINX"
23  #define NGX_OLDPID_EXT     ".oldbin"
24
25
26  #endif /* _NGINX_H_INCLUDED_ */
```

src/core/ngx_conf_file.c - nginx-1.7.10

Global variables defined

- argument_number
- ngx_conf_commands
- ngx_conf_module

Functions defined

- ngx_conf_check_num_bounds
- ngx_conf_deprecated
- ngx_conf_flush_files
- ngx_conf_full_name
- ngx_conf_handler
- ngx_conf_include
- ngx_conf_log_error
- ngx_conf_open_file
- ngx_conf_param
- ngx_conf_parse
- ngx_conf_read_token
- ngx_conf_set_bitmask_slot
- ngx_conf_set_bufs_slot
- ngx_conf_set_enum_slot
- ngx_conf_set_flag_slot
- ngx_conf_set_keyval_slot
- ngx_conf_set_msec_slot
- ngx_conf_set_num_slot
- ngx_conf_set_off_slot
- ngx_conf_set_sec_slot
- ngx_conf_set_size_slot
- ngx_conf_set_str_array_slot
- ngx_conf_set_str_slot

Macros defined

- NGX_CONF_BUFFER

Source code

```

/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>

 #define NGX_CONF_BUFFER  4096

 static ngx_int_t ngx_conf_handler(ngx_conf_t *cf, ngx_int_t last);
 static ngx_int_t ngx_conf_read_token(ngx_conf_t *cf);
 static void ngx_conf_flush_files(ngx_cycle_t *cycle);


 static ngx_command_t  ngx_conf_commands[] = {

     { ngx_string("include"),
       NGX_ANY_CONF|NGX_CONF_TAKE1,
       ngx_conf_include,
       0,
       0,
       NULL },

       ngx_null_command
 };


 ngx_module_t  ngx_conf_module = {
     NGX_MODULE_V1,
     NULL,                                  /* module context */
     ngx_conf_commands,                     /* module directives */
     NGX_CONF_MODULE,                       /* module type */
     NULL,                                  /* init master */
     NULL,                                  /* init module */
     NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     ngx_conf_flush_files,                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
 };


 /* The eight fixed arguments */

 static ngx_uint_t argument_number[] = {
     NGX_CONF_NOARGS,
     NGX_CONF_TAKE1,
     NGX_CONF_TAKE2,
     NGX_CONF_TAKE3,
     NGX_CONF_TAKE4,
     NGX_CONF_TAKE5,
     NGX_CONF_TAKE6,
     NGX_CONF_TAKE7
 };


 char *
 ngx_conf_param(ngx_conf_t *cf)
 {
     char             *rv;
     ngx_str_t        *param;
     ngx_buf_t         b;
     ngx_conf_file_t   conf_file;

     param = &cf->cycle->conf_param;

     if (param->len == 0) {
         return NGX_CONF_OK;
     }
```

```
74
75      ngx_memzero(&conf_file, sizeof(ngx_conf_file_t));
76
77      ngx_memzero(&b, sizeof(ngx_buf_t));
78
79      b.start = param->data;
80      b.pos = param->data;
81      b.last = param->data + param->len;
82      b.end = b.last;
83      b.temporary = 1;
84
85      conf_file.file.fd = NGX_INVALID_FILE;
86      conf_file.file.name.data = NULL;
87      conf_file.line = 0;
88
89      cf->conf_file = &conf_file;
90      cf->conf_file->buffer = &b;
91
92      rv = ngx_conf_parse(cf, NULL);
93
94      cf->conf_file = NULL;
95
96      return rv;
97  }
98
99
100 char *
101 ngx_conf_parse(ngx_conf_t *cf, ngx_str_t *filename)
102 {
103     char             *rv;
104     ngx_fd_t          fd;
105     ngx_int_t         rc;
106     ngx_buf_t         buf;
107     ngx_conf_file_t  *prev, conf_file;
108     enum {
109         parse_file = 0,
110         parse_block,
111         parse_param
112     } type;
113
114 #if (NGX_SUPPRESS_WARN)
115     fd = NGX_INVALID_FILE;
116     prev = NULL;
117 #endif
118
119     if (filename) {
120
121         /* open configuration file */
122
123         fd = ngx_open_file(filename->data, NGX_FILE_RDONLY, NGX_FILE_OPEN, 0);
124         if (fd == NGX_INVALID_FILE) {
125             ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
126                                ngx_open_file_n " \"%s\" failed",
127                                filename->data);
128             return NGX_CONF_ERROR;
129         }
130
131         prev = cf->conf_file;
132
133         cf->conf_file = &conf_file;
134
135         if (ngx_fd_info(fd, &cf->conf_file->file.info) == NGX_FILE_ERROR) {
136             ngx_log_error(NGX_LOG_EMERG, cf->log, ngx_errno,
137                           ngx_fd_info_n " \"%s\" failed", filename->data);
138         }
139
140         cf->conf_file->buffer = &buf;
141
142         buf.start = ngx_alloc(NGX_CONF_BUFFER, cf->log);
143         if (buf.start == NULL) {
144             goto failed;
145         }
146
147         buf.pos = buf.start;
148         buf.last = buf.start;
149         buf.end = buf.last + NGX_CONF_BUFFER;
```

```
150         buf.temporary = 1;
151
152         cf->conf_file->file.fd = fd;
153         cf->conf_file->file.name.len = filename->len;
154         cf->conf_file->file.name.data = filename->data;
155         cf->conf_file->file.offset = 0;
156         cf->conf_file->file.log = cf->log;
157         cf->conf_file->line = 1;
158
159         type = parse_file;
160
161     } else if (cf->conf_file->file.fd != NGX_INVALID_FILE) {
162
163         type = parse_block;
164
165     } else {
166         type = parse_param;
167     }
168
169
170     for ( ;; ) {
171         rc = ngx_conf_read_token(cf);
172
173         /*
174          * ngx_conf_read_token() may return
175          *
176          *    NGX_ERROR             there is error
177          *    NGX_OK                the token terminated by ";" was found
178          *    NGX_CONF_BLOCK_START  the token terminated by "{" was found
179          *    NGX_CONF_BLOCK_DONE   the "}" was found
180          *    NGX_CONF_FILE_DONE    the configuration file is done
181          */
182
183         if (rc == NGX_ERROR) {
184             goto done;
185         }
186
187         if (rc == NGX_CONF_BLOCK_DONE) {
188
189             if (type != parse_block) {
190                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "unexpected \"}\"");
191                 goto failed;
192             }
193
194             goto done;
195         }
196
197         if (rc == NGX_CONF_FILE_DONE) {
198
199             if (type == parse_block) {
200                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
201                                    "unexpected end of file, expecting \"}\"");
202                 goto failed;
203             }
204
205             goto done;
206         }
207
208         if (rc == NGX_CONF_BLOCK_START) {
209
210             if (type == parse_param) {
211                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
212                                    "block directives are not supported "
213                                    "in -g option");
214                 goto failed;
215             }
216         }
217
218         /* rc == NGX_OK || rc == NGX_CONF_BLOCK_START */
219
220         if (cf->handler) {
221
222             /*
223              * the custom handler, i.e., that is used in the http's
224              * "types { ... }" directive
225              */
```

```
226
227             if (rc == NGX_CONF_BLOCK_START) {
228                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "unexpected \"{\"");
229                 goto failed;
230             }
231
232             rv = (*cf->handler)(cf, NULL, cf->handler_conf);
233             if (rv == NGX_CONF_OK) {
234                 continue;
235             }
236
237             if (rv == NGX_CONF_ERROR) {
238                 goto failed;
239             }
240
241             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, rv);
242
243             goto failed;
244         }
245
246
247         rc = ngx_conf_handler(cf, rc);
248
249         if (rc == NGX_ERROR) {
250             goto failed;
251         }
252     }
253
254 failed:
255
256     rc = NGX_ERROR;
257
258 done:
259
260     if (filename) {
261         if (cf->conf_file->buffer->start) {
262             ngx_free(cf->conf_file->buffer->start);
263         }
264
265         if (ngx_close_file(fd) == NGX_FILE_ERROR) {
266             ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
267                           ngx_close_file_n " %s failed",
268                           filename->data);
269             rc = NGX_ERROR;
270         }
271
272         cf->conf_file = prev;
273     }
274
275     if (rc == NGX_ERROR) {
276         return NGX_CONF_ERROR;
277     }
278
279     return NGX_CONF_OK;
280 }
281
282
283 static ngx_int_t
284 ngx_conf_handler(ngx_conf_t *cf, ngx_int_t last)
285 {
286     char           *rv;
287     void           *conf, **confp;
288     ngx_uint_t      i, found;
289     ngx_str_t      *name;
290     ngx_command_t  *cmd;
291
292     name = cf->args->elts;
293
294     found = 0;
295
296     for (i = 0; ngx_modules[i]; i++) {
297
298         cmd = ngx_modules[i]->commands;
299         if (cmd == NULL) {
300             continue;
301         }
```

```
302
303         for ( /* void */ ; cmd->name.len; cmd++) {
304
305             if (name->len != cmd->name.len) {
306                 continue;
307             }
308
309             if (ngx_strcmp(name->data, cmd->name.data) != 0) {
310                 continue;
311             }
312
313             found = 1;
314
315             if (ngx_modules[i]->type != NGX_CONF_MODULE
316                 && ngx_modules[i]->type != cf->module_type)
317             {
318                 continue;
319             }
320
321             /* is the directive's location right ? */
322
323             if (!(cmd->type & cf->cmd_type)) {
324                 continue;
325             }
326
327             if (!(cmd->type & NGX_CONF_BLOCK) && last != NGX_OK) {
328                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
329                                    "directive \"%s\" is not terminated by \";\"",
330                                    name->data);
331                 return NGX_ERROR;
332             }
333
334             if ((cmd->type & NGX_CONF_BLOCK) && last != NGX_CONF_BLOCK_START) {
335                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
336                                    "directive \"%s\" has no opening \"{\"",
337                                    name->data);
338                 return NGX_ERROR;
339             }
340
341             /* is the directive's argument count right ? */
342
343             if (!(cmd->type & NGX_CONF_ANY)) {
344
345                 if (cmd->type & NGX_CONF_FLAG) {
346
347                     if (cf->args->nelts != 2) {
348                         goto invalid;
349                     }
350
351                 } else if (cmd->type & NGX_CONF_1MORE) {
352
353                     if (cf->args->nelts < 2) {
354                         goto invalid;
355                     }
356
357                 } else if (cmd->type & NGX_CONF_2MORE) {
358
359                     if (cf->args->nelts < 3) {
360                         goto invalid;
361                     }
362
363                 } else if (cf->args->nelts > NGX_CONF_MAX_ARGS) {
364
365                     goto invalid;
366
367                 } else if (!(cmd->type & argument_number[cf->args->nelts - 1]))
368                 {
369                     goto invalid;
370                 }
371             }
372
373             /* set up the directive's configuration context */
374
375             conf = NULL;
376
377             if (cmd->type & NGX_DIRECT_CONF) {
```

```
                conf = ((void **) cf->ctx)[ngx_modules[i]->index];

            } else if (cmd->type & NGX_MAIN_CONF) {
                conf = &(((void **) cf->ctx)[ngx_modules[i]->index]);

            } else if (cf->ctx) {
                confp = *(void **) ((char *) cf->ctx + cmd->conf);

                if (confp) {
                    conf = confp[ngx_modules[i]->ctx_index];
                }
            }

            rv = cmd->set(cf, cmd, conf);

            if (rv == NGX_CONF_OK) {
                return NGX_OK;
            }

            if (rv == NGX_CONF_ERROR) {
                return NGX_ERROR;
            }

            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "\"%s\" directive %s", name->data, rv);

            return NGX_ERROR;
        }
    }

    if (found) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"%s\" directive is not allowed here", name->data);

        return NGX_ERROR;
    }

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "unknown directive \"%s\"", name->data);

    return NGX_ERROR;

invalid:

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "invalid number of arguments in \"%s\" directive",
                       name->data);

    return NGX_ERROR;
}


static ngx_int_t
ngx_conf_read_token(ngx_conf_t *cf)
{
    u_char      *start, ch, *src, *dst;
    off_t        file_size;
    size_t       len;
    ssize_t      n, size;
    ngx_uint_t   found, need_space, last_space, sharp_comment, variable;
    ngx_uint_t   quoted, s_quoted, d_quoted, start_line;
    ngx_str_t   *word;
    ngx_buf_t   *b;

    found = 0;
    need_space = 0;
    last_space = 1;
    sharp_comment = 0;
    variable = 0;
    quoted = 0;
    s_quoted = 0;
    d_quoted = 0;

    cf->args->nelts = 0;
    b = cf->conf_file->buffer;
    start = b->pos;
```

```
    start_line = cf->conf_file->line;

    file_size = ngx_file_size(&cf->conf_file->file.info);

    for ( ;; ) {

        if (b->pos >= b->last) {

            if (cf->conf_file->file.offset >= file_size) {

                if (cf->args->nelts > 0 || !last_space) {

                    if (cf->conf_file->file.fd == NGX_INVALID_FILE) {
                        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                           "unexpected end of parameter, "
                                           "expecting \";\"");
                        return NGX_ERROR;
                    }

                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "unexpected end of file, "
                                       "expecting \";\" or \"}\"");
                    return NGX_ERROR;
                }

                return NGX_CONF_FILE_DONE;
            }

            len = b->pos - start;

            if (len == NGX_CONF_BUFFER) {
                cf->conf_file->line = start_line;

                if (d_quoted) {
                    ch = '"';

                } else if (s_quoted) {
                    ch = '\'';

                } else {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "too long parameter \"%*s...\" started",
                                       10, start);
                    return NGX_ERROR;
                }

                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "too long parameter, probably "
                                   "missing terminating \"%c\" character", ch);
                return NGX_ERROR;
            }

            if (len) {
                ngx_memmove(b->start, start, len);
            }

            size = (ssize_t) (file_size - cf->conf_file->file.offset);

            if (size > b->end - (b->start + len)) {
                size = b->end - (b->start + len);
            }

            n = ngx_read_file(&cf->conf_file->file, b->start + len, size,
                              cf->conf_file->file.offset);

            if (n == NGX_ERROR) {
                return NGX_ERROR;
            }

            if (n != size) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   ngx_read_file_n " returned "
                                   "only %z bytes instead of %z",
                                   n, size);
                return NGX_ERROR;
            }
```

```
            b->pos = b->start + len;
            b->last = b->pos + n;
            start = b->start;
        }

        ch = *b->pos++;

        if (ch == LF) {
            cf->conf_file->line++;

            if (sharp_comment) {
                sharp_comment = 0;
            }
        }

        if (sharp_comment) {
            continue;
        }

        if (quoted) {
            quoted = 0;
            continue;
        }

        if (need_space) {
            if (ch == ' ' || ch == '\t' || ch == CR || ch == LF) {
                last_space = 1;
                need_space = 0;
                continue;
            }

            if (ch == ';') {
                return NGX_OK;
            }

            if (ch == '{') {
                return NGX_CONF_BLOCK_START;
            }

            if (ch == ')') {
                last_space = 1;
                need_space = 0;

            } else {
                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                    "unexpected \"%c\"", ch);
                 return NGX_ERROR;
            }
        }

        if (last_space) {
            if (ch == ' ' || ch == '\t' || ch == CR || ch == LF) {
                continue;
            }

            start = b->pos - 1;
            start_line = cf->conf_file->line;

            switch (ch) {

            case ';':
            case '{':
                if (cf->args->nelts == 0) {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "unexpected \"%c\"", ch);
                    return NGX_ERROR;
                }

                if (ch == '{') {
                    return NGX_CONF_BLOCK_START;
                }

                return NGX_OK;

            case '}':
```

```
                if (cf->args->nelts != 0) {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "unexpected \"}\"");
                    return NGX_ERROR;
                }

                return NGX_CONF_BLOCK_DONE;

            case '#':
                sharp_comment = 1;
                continue;

            case '\\':
                quoted = 1;
                last_space = 0;
                continue;

            case '"':
                start++;
                d_quoted = 1;
                last_space = 0;
                continue;

            case '\'':
                start++;
                s_quoted = 1;
                last_space = 0;
                continue;

            default:
                last_space = 0;
            }

        } else {
            if (ch == '{' && variable) {
                continue;
            }

            variable = 0;

            if (ch == '\\') {
                quoted = 1;
                continue;
            }

            if (ch == '$') {
                variable = 1;
                continue;
            }

            if (d_quoted) {
                if (ch == '"') {
                    d_quoted = 0;
                    need_space = 1;
                    found = 1;
                }

            } else if (s_quoted) {
                if (ch == '\'') {
                    s_quoted = 0;
                    need_space = 1;
                    found = 1;
                }

            } else if (ch == ' ' || ch == '\t' || ch == CR || ch == LF
                       || ch == ';' || ch == '{')
            {
                last_space = 1;
                found = 1;
            }

            if (found) {
                word = ngx_array_push(cf->args);
                if (word == NULL) {
                    return NGX_ERROR;
                }
```

```
                word->data = ngx_pnalloc(cf->pool, b->pos - start + 1);
                if (word->data == NULL) {
                    return NGX_ERROR;
                }

                for (dst = word->data, src = start, len = 0;
                     src < b->pos - 1;
                     len++)
                {
                    if (*src == '\\') {
                        switch (src[1]) {
                        case '"':
                        case '\'':
                        case '\\':
                            src++;
                            break;

                        case 't':
                            *dst++ = '\t';
                            src += 2;
                            continue;

                        case 'r':
                            *dst++ = '\r';
                            src += 2;
                            continue;

                        case 'n':
                            *dst++ = '\n';
                            src += 2;
                            continue;
                        }

                    }
                    *dst++ = *src++;
                }
                *dst = '\0';
                word->len = len;

                if (ch == ';') {
                    return NGX_OK;
                }

                if (ch == '{') {
                    return NGX_CONF_BLOCK_START;
                }

                found = 0;
            }
        }
    }
}


char *
ngx_conf_include(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char        *rv;
    ngx_int_t    n;
    ngx_str_t   *value, file, name;
    ngx_glob_t   gl;

    value = cf->args->elts;
    file = value[1];

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, cf->log, 0, "include %s", file.data);

    if (ngx_conf_full_name(cf->cycle, &file, 1) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (strpbrk((char *) file.data, "*?[") == NULL) {

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, cf->log, 0, "include %s", file.data);

```

```
        return ngx_conf_parse(cf, &file);
    }

    ngx_memzero(&gl, sizeof(ngx_glob_t));

    gl.pattern = file.data;
    gl.log = cf->log;
    gl.test = 1;

    if (ngx_open_glob(&gl) != NGX_OK) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
                           ngx_open_glob_n " \"%s\" failed", file.data);
        return NGX_CONF_ERROR;
    }

    rv = NGX_CONF_OK;

    for ( ;; ) {
        n = ngx_read_glob(&gl, &name);

        if (n != NGX_OK) {
            break;
        }

        file.len = name.len++;
        file.data = ngx_pstrdup(cf->pool, &name);
        if (file.data == NULL) {
            return NGX_CONF_ERROR;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, cf->log, 0, "include %s", file.data);

        rv = ngx_conf_parse(cf, &file);

        if (rv != NGX_CONF_OK) {
            break;
        }
    }

    ngx_close_glob(&gl);

    return rv;
}


ngx_int_t
ngx_conf_full_name(ngx_cycle_t *cycle, ngx_str_t *name, ngx_uint_t conf_prefix)
{
    ngx_str_t  *prefix;

    prefix = conf_prefix ? &cycle->conf_prefix : &cycle->prefix;

    return ngx_get_full_name(cycle->pool, prefix, name);
}


ngx_open_file_t *
ngx_conf_open_file(ngx_cycle_t *cycle, ngx_str_t *name)
{
    ngx_str_t         full;
    ngx_uint_t        i;
    ngx_list_part_t  *part;
    ngx_open_file_t  *file;

#if (NGX_SUPPRESS_WARN)
    ngx_str_null(&full);
#endif

    if (name->len) {
        full = *name;

        if (ngx_conf_full_name(cycle, &full, 0) != NGX_OK) {
            return NULL;
        }

        part = &cycle->open_files.part;
```

```
        file = part->elts;

        for (i = 0; /* void */ ; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }
                part = part->next;
                file = part->elts;
                i = 0;
            }

            if (full.len != file[i].name.len) {
                continue;
            }

            if (ngx_strcmp(full.data, file[i].name.data) == 0) {
                return &file[i];
            }
        }
    }

    file = ngx_list_push(&cycle->open_files);
    if (file == NULL) {
        return NULL;
    }

    if (name->len) {
        file->fd = NGX_INVALID_FILE;
        file->name = full;

    } else {
        file->fd = ngx_stderr;
        file->name = *name;
    }

    file->flush = NULL;
    file->data = NULL;

    return file;
}


static void
ngx_conf_flush_files(ngx_cycle_t *cycle)
{
    ngx_uint_t        i;
    ngx_list_part_t  *part;
    ngx_open_file_t  *file;

    ngx_log_debug0(NGX_LOG_DEBUG_CORE, cycle->log, 0, "flush files");

    part = &cycle->open_files.part;
    file = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }
            part = part->next;
            file = part->elts;
            i = 0;
        }

        if (file[i].flush) {
            file[i].flush(&file[i], cycle->log);
        }
    }
}


void ngx_cdecl
ngx_conf_log_error(ngx_uint_t level, ngx_conf_t *cf, ngx_err_t err,
```

```
    const char *fmt, ...)
{
    u_char   errstr[NGX_MAX_CONF_ERRSTR], *p, *last;
    va_list  args;

    last = errstr + NGX_MAX_CONF_ERRSTR;

    va_start(args, fmt);
    p = ngx_vslprintf(errstr, last, fmt, args);
    va_end(args);

    if (err) {
        p = ngx_log_errno(p, last, err);
    }

    if (cf->conf_file == NULL) {
        ngx_log_error(level, cf->log, 0, "%*s", p - errstr, errstr);
        return;
    }

    if (cf->conf_file->file.fd == NGX_INVALID_FILE) {
        ngx_log_error(level, cf->log, 0, "%*s in command line",
                      p - errstr, errstr);
        return;
    }

    ngx_log_error(level, cf->log, 0, "%*s in %s:%ui",
                  p - errstr, errstr,
                  cf->conf_file->file.name.data, cf->conf_file->line);
}


char *
ngx_conf_set_flag_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    ngx_str_t        *value;
    ngx_flag_t       *fp;
    ngx_conf_post_t  *post;

    fp = (ngx_flag_t *) (p + cmd->offset);

    if (*fp != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    if (ngx_strcasecmp(value[1].data, (u_char *) "on") == 0) {
        *fp = 1;

    } else if (ngx_strcasecmp(value[1].data, (u_char *) "off") == 0) {
        *fp = 0;

    } else {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid value \"%s\" in \"%s\" directive, "
                           "it must be \"on\" or \"off\"",
                           value[1].data, cmd->name.data);
        return NGX_CONF_ERROR;
    }

    if (cmd->post) {
        post = cmd->post;
        return post->post_handler(cf, post, fp);
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_str_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;
```

```
986
987     ngx_str_t        *field, *value;
988     ngx_conf_post_t  *post;
989
990     field = (ngx_str_t *) (p + cmd->offset);
991
992     if (field->data) {
993         return "is duplicate";
994     }
995
996     value = cf->args->elts;
997
998     *field = value[1];
999
1000     if (cmd->post) {
1001         post = cmd->post;
1002         return post->post_handler(cf, post, field);
1003     }
1004
1005     return NGX_CONF_OK;
1006 }
1007
1008
1009 char *
1010 ngx_conf_set_str_array_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1011 {
1012     char  *p = conf;
1013
1014     ngx_str_t         *value, *s;
1015     ngx_array_t      **a;
1016     ngx_conf_post_t   *post;
1017
1018     a = (ngx_array_t **) (p + cmd->offset);
1019
1020     if (*a == NGX_CONF_UNSET_PTR) {
1021         *a = ngx_array_create(cf->pool, 4, sizeof(ngx_str_t));
1022         if (*a == NULL) {
1023             return NGX_CONF_ERROR;
1024         }
1025     }
1026
1027     s = ngx_array_push(*a);
1028     if (s == NULL) {
1029         return NGX_CONF_ERROR;
1030     }
1031
1032     value = cf->args->elts;
1033
1034     *s = value[1];
1035
1036     if (cmd->post) {
1037         post = cmd->post;
1038         return post->post_handler(cf, post, s);
1039     }
1040
1041     return NGX_CONF_OK;
1042 }
1043
1044
1045 char *
1046 ngx_conf_set_keyval_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1047 {
1048     char  *p = conf;
1049
1050     ngx_str_t         *value;
1051     ngx_array_t      **a;
1052     ngx_keyval_t      *kv;
1053     ngx_conf_post_t   *post;
1054
1055     a = (ngx_array_t **) (p + cmd->offset);
1056
1057     if (*a == NULL) {
1058         *a = ngx_array_create(cf->pool, 4, sizeof(ngx_keyval_t));
1059         if (*a == NULL) {
1060             return NGX_CONF_ERROR;
1061         }
```

```
1062      }
1063
1064      kv = ngx_array_push(*a);
1065      if (kv == NULL) {
1066          return NGX_CONF_ERROR;
1067      }
1068
1069      value = cf->args->elts;
1070
1071      kv->key = value[1];
1072      kv->value = value[2];
1073
1074      if (cmd->post) {
1075          post = cmd->post;
1076          return post->post_handler(cf, post, kv);
1077      }
1078
1079      return NGX_CONF_OK;
1080  }
1081
1082
1083  char *
1084  ngx_conf_set_num_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1085  {
1086      char  *p = conf;
1087
1088      ngx_int_t        *np;
1089      ngx_str_t        *value;
1090      ngx_conf_post_t  *post;
1091
1092
1093      np = (ngx_int_t *) (p + cmd->offset);
1094
1095      if (*np != NGX_CONF_UNSET) {
1096          return "is duplicate";
1097      }
1098
1099      value = cf->args->elts;
1100      *np = ngx_atoi(value[1].data, value[1].len);
1101      if (*np == NGX_ERROR) {
1102          return "invalid number";
1103      }
1104
1105      if (cmd->post) {
1106          post = cmd->post;
1107          return post->post_handler(cf, post, np);
1108      }
1109
1110      return NGX_CONF_OK;
1111  }
1112
1113
1114  char *
1115  ngx_conf_set_size_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1116  {
1117      char  *p = conf;
1118
1119      size_t           *sp;
1120      ngx_str_t        *value;
1121      ngx_conf_post_t  *post;
1122
1123
1124      sp = (size_t *) (p + cmd->offset);
1125      if (*sp != NGX_CONF_UNSET_SIZE) {
1126          return "is duplicate";
1127      }
1128
1129      value = cf->args->elts;
1130
1131      *sp = ngx_parse_size(&value[1]);
1132      if (*sp == (size_t) NGX_ERROR) {
1133          return "invalid value";
1134      }
1135
1136      if (cmd->post) {
1137          post = cmd->post;
```

```
        return post->post_handler(cf, post, sp);
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_off_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    off_t            *op;
    ngx_str_t        *value;
    ngx_conf_post_t  *post;


    op = (off_t *) (p + cmd->offset);
    if (*op != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    *op = ngx_parse_offset(&value[1]);
    if (*op == (off_t) NGX_ERROR) {
        return "invalid value";
    }

    if (cmd->post) {
        post = cmd->post;
        return post->post_handler(cf, post, op);
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_msec_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    ngx_msec_t       *msp;
    ngx_str_t        *value;
    ngx_conf_post_t  *post;


    msp = (ngx_msec_t *) (p + cmd->offset);
    if (*msp != NGX_CONF_UNSET_MSEC) {
        return "is duplicate";
    }

    value = cf->args->elts;

    *msp = ngx_parse_time(&value[1], 0);
    if (*msp == (ngx_msec_t) NGX_ERROR) {
        return "invalid value";
    }

    if (cmd->post) {
        post = cmd->post;
        return post->post_handler(cf, post, msp);
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_sec_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    time_t           *sp;
    ngx_str_t        *value;
```

```
    ngx_conf_post_t  *post;


    sp = (time_t *) (p + cmd->offset);
    if (*sp != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    *sp = ngx_parse_time(&value[1], 1);
    if (*sp == (time_t) NGX_ERROR) {
        return "invalid value";
    }

    if (cmd->post) {
        post = cmd->post;
        return post->post_handler(cf, post, sp);
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_bufs_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char *p = conf;

    ngx_str_t   *value;
    ngx_bufs_t  *bufs;


    bufs = (ngx_bufs_t *) (p + cmd->offset);
    if (bufs->num) {
        return "is duplicate";
    }

    value = cf->args->elts;

    bufs->num = ngx_atoi(value[1].data, value[1].len);
    if (bufs->num == NGX_ERROR || bufs->num == 0) {
        return "invalid value";
    }

    bufs->size = ngx_parse_size(&value[2]);
    if (bufs->size == (size_t) NGX_ERROR || bufs->size == 0) {
        return "invalid value";
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_enum_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    ngx_uint_t       *np, i;
    ngx_str_t        *value;
    ngx_conf_enum_t  *e;

    np = (ngx_uint_t *) (p + cmd->offset);

    if (*np != NGX_CONF_UNSET_UINT) {
        return "is duplicate";
    }

    value = cf->args->elts;
    e = cmd->post;

    for (i = 0; e[i].name.len != 0; i++) {
        if (e[i].name.len != value[1].len
            || ngx_strcasecmp(e[i].name.data, value[1].data) != 0)
        {
```

```
            continue;
        }

        *np = e[i].value;

        return NGX_CONF_OK;
    }

    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                       "invalid value \"%s\"", value[1].data);

    return NGX_CONF_ERROR;
}


char *
ngx_conf_set_bitmask_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    ngx_uint_t          *np, i, m;
    ngx_str_t           *value;
    ngx_conf_bitmask_t  *mask;


    np = (ngx_uint_t *) (p + cmd->offset);
    value = cf->args->elts;
    mask = cmd->post;

    for (i = 1; i < cf->args->nelts; i++) {
        for (m = 0; mask[m].name.len != 0; m++) {

            if (mask[m].name.len != value[i].len
                || ngx_strcasecmp(mask[m].name.data, value[i].data) != 0)
            {
                continue;
            }

            if (*np & mask[m].mask) {
                ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                                   "duplicate value \"%s\"", value[i].data);

            } else {
                *np |= mask[m].mask;
            }

            break;
        }

        if (mask[m].name.len == 0) {
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                               "invalid value \"%s\"", value[i].data);

            return NGX_CONF_ERROR;
        }
    }

    return NGX_CONF_OK;
}


#if 0

char *
ngx_conf_unsupported(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    return "unsupported on this platform";
}

#endif


char *
ngx_conf_deprecated(ngx_conf_t *cf, void *post, void *data)
{
    ngx_conf_deprecated_t  *d = post;
```

```
1366
1367        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1368                           "the \"%s\" directive is deprecated, "
1369                           "use the \"%s\" directive instead",
1370                           d->old_name, d->new_name);
1371
1372        return NGX_CONF_OK;
1373    }
1374
1375
1376    char *
1377    ngx_conf_check_num_bounds(ngx_conf_t *cf, void *post, void *data)
1378    {
1379        ngx_conf_num_bounds_t  *bounds = post;
1380        ngx_int_t  *np = data;
1381
1382        if (bounds->high == -1) {
1383            if (*np >= bounds->low) {
1384                return NGX_CONF_OK;
1385            }
1386
1387            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1388                               "value must be equal to or greater than %i",
1389                               bounds->low);
1390
1391            return NGX_CONF_ERROR;
1392        }
1393
1394        if (*np >= bounds->low && *np <= bounds->high) {
1395            return NGX_CONF_OK;
1396        }
1397
1398        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1399                           "value must be between %i and %i",
1400                           bounds->low, bounds->high);
1401
1402        return NGX_CONF_ERROR;
1403    }
```

One Level Up Top Level

src/core/ngx_buf.h - nginx-1.7.10

Data types defined

- ngx_buf_s
- ngx_buf_t
- ngx_buf_tag_t
- ngx_bufs_t
- ngx_chain_s
- ngx_chain_writer_ctx_t
- ngx_output_chain_aio_pt
- ngx_output_chain_ctx_s
- ngx_output_chain_ctx_t
- ngx_output_chain_filter_pt

Macros defined

- NGX_CHAIN_ERROR
- _NGX_BUF_H_INCLUDED_
- ngx_alloc_buf
- ngx_buf_in_memory
- ngx_buf_in_memory_only
- ngx_buf_size
- ngx_buf_special
- ngx_buf_sync_only
- ngx_calloc_buf
- ngx_free_chain

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_BUF_H_INCLUDED_
9  #define _NGX_BUF_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef void *            ngx_buf_tag_t;
```

```
17
18  typedef struct ngx_buf_s  ngx_buf_t;
19
20  struct ngx_buf_s {
21      u_char          *pos;
22      u_char          *last;
23      off_t            file_pos;
24      off_t            file_last;
25
26      u_char          *start;         /* start of buffer */
27      u_char          *end;           /* end of buffer */
28      ngx_buf_tag_t    tag;
29      ngx_file_t      *file;
30      ngx_buf_t       *shadow;
31
32
33      /* the buf's content could be changed */
34      unsigned         temporary:1;
35
36      /*
37       * the buf's content is in a memory cache or in a read only memory
38       * and must not be changed
39       */
40      unsigned         memory:1;
41
42      /* the buf's content is mmap()ed and must not be changed */
43      unsigned         mmap:1;
44
45      unsigned         recycled:1;
46      unsigned         in_file:1;
47      unsigned         flush:1;
48      unsigned         sync:1;
49      unsigned         last_buf:1;
50      unsigned         last_in_chain:1;
51
52      unsigned         last_shadow:1;
53      unsigned         temp_file:1;
54
55      /* STUB */ int   num;
56  };
57
58
59  struct ngx_chain_s {
60      ngx_buf_t    *buf;
61      ngx_chain_t  *next;
62  };
63
64
65  typedef struct {
66      ngx_int_t    num;
67      size_t       size;
68  } ngx_bufs_t;
69
70
71  typedef struct ngx_output_chain_ctx_s  ngx_output_chain_ctx_t;
72
73  typedef ngx_int_t (*ngx_output_chain_filter_pt)(void *ctx, ngx_chain_t *in);
74
75  #if (NGX_HAVE_FILE_AIO)
76  typedef void (*ngx_output_chain_aio_pt)(ngx_output_chain_ctx_t *ctx,
77      ngx_file_t *file);
78  #endif
79
80  struct ngx_output_chain_ctx_s {
81      ngx_buf_t                   *buf;
82      ngx_chain_t                 *in;
83      ngx_chain_t                 *free;
84      ngx_chain_t                 *busy;
85
86      unsigned                     sendfile:1;
87      unsigned                     directio:1;
88  #if (NGX_HAVE_ALIGNED_DIRECTIO)
89      unsigned                     unaligned:1;
90  #endif
91      unsigned                     need_in_memory:1;
92      unsigned                     need_in_temp:1;
```

```
#if (NGX_HAVE_FILE_AIO)
    unsigned                     aio:1;

    ngx_output_chain_aio_pt      aio_handler;
#endif

    off_t                        alignment;

     ngx_pool_t                  *pool;
     ngx_int_t                    allocated;
     ngx_bufs_t                   bufs;
     ngx_buf_tag_t                tag;

     ngx_output_chain_filter_pt   output_filter;
     void                        *filter_ctx;
};


typedef struct {
    ngx_chain_t                 *out;
    ngx_chain_t                **last;
    ngx_connection_t            *connection;
    ngx_pool_t                  *pool;
    off_t                        limit;
} ngx_chain_writer_ctx_t;


#define NGX_CHAIN_ERROR     (ngx_chain_t *) NGX_ERROR


#define ngx_buf_in_memory(b)        (b->temporary || b->memory || b->mmap)
#define ngx_buf_in_memory_only(b)   (ngx_buf_in_memory(b) && !b->in_file)

#define ngx_buf_special(b)                                                   \
    ((b->flush || b->last_buf || b->sync)                                    \
     && !ngx_buf_in_memory(b) && !b->in_file)

#define ngx_buf_sync_only(b)                                                 \
    (b->sync                                                                 \
     && !ngx_buf_in_memory(b) && !b->in_file && !b->flush && !b->last_buf)

#define ngx_buf_size(b)                                                      \
    (ngx_buf_in_memory(b) ? (off_t) (b->last - b->pos):                      \
                            (b->file_last - b->file_pos))

ngx_buf_t *ngx_create_temp_buf(ngx_pool_t *pool, size_t size);
ngx_chain_t *ngx_create_chain_of_bufs(ngx_pool_t *pool, ngx_bufs_t *bufs);


#define ngx_alloc_buf(pool)  ngx_palloc(pool, sizeof(ngx_buf_t))
#define ngx_calloc_buf(pool) ngx_pcalloc(pool, sizeof(ngx_buf_t))

ngx_chain_t *ngx_alloc_chain_link(ngx_pool_t *pool);
#define ngx_free_chain(pool, cl)                                             \
    cl->next = pool->chain;                                                  \
    pool->chain = cl



ngx_int_t ngx_output_chain(ngx_output_chain_ctx_t *ctx, ngx_chain_t *in);
ngx_int_t ngx_chain_writer(void *ctx, ngx_chain_t *in);

ngx_int_t ngx_chain_add_copy(ngx_pool_t *pool, ngx_chain_t **chain,
    ngx_chain_t *in);
ngx_chain_t *ngx_chain_get_free_buf(ngx_pool_t *p, ngx_chain_t **free);
void ngx_chain_update_chains(ngx_pool_t *p, ngx_chain_t **free,
    ngx_chain_t **busy, ngx_chain_t **out, ngx_buf_tag_t tag);

off_t ngx_chain_coalesce_file(ngx_chain_t **in, off_t limit);

ngx_chain_t *ngx_chain_update_sent(ngx_chain_t *in, off_t sent);

#endif /* _NGX_BUF_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_buf.c - nginx-1.7.10

Functions defined

- ngx_alloc_chain_link
- ngx_chain_add_copy
- ngx_chain_coalesce_file
- ngx_chain_get_free_buf
- ngx_chain_update_chains
- ngx_chain_update_sent
- ngx_create_chain_of_bufs
- ngx_create_temp_buf

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 ngx_buf_t *
13 ngx_create_temp_buf(ngx_pool_t *pool, size_t size)
14 {
15     ngx_buf_t *b;
16
17     b = ngx_calloc_buf(pool);
18     if (b == NULL) {
19         return NULL;
20     }
21
22     b->start = ngx_palloc(pool, size);
23     if (b->start == NULL) {
24         return NULL;
25     }
26
27     /*
28      * set by ngx_calloc_buf():
29      *
30      *     b->file_pos = 0;
31      *     b->file_last = 0;
32      *     b->file = NULL;
33      *     b->shadow = NULL;
34      *     b->tag = 0;
35      *     and flags
36      */
37
38     b->pos = b->start;
39     b->last = b->start;
40     b->end = b->last + size;
41     b->temporary = 1;
42
43     return b;
44 }
45
46
47 ngx_chain_t *
```

```
ngx_alloc_chain_link(ngx_pool_t *pool)
{
    ngx_chain_t  *cl;

    cl = pool->chain;

    if (cl) {
        pool->chain = cl->next;
        return cl;
    }

    cl = ngx_palloc(pool, sizeof(ngx_chain_t));
    if (cl == NULL) {
        return NULL;
    }

    return cl;
}


ngx_chain_t *
ngx_create_chain_of_bufs(ngx_pool_t *pool, ngx_bufs_t *bufs)
{
    u_char       *p;
    ngx_int_t     i;
    ngx_buf_t    *b;
    ngx_chain_t  *chain, *cl, **ll;

    p = ngx_palloc(pool, bufs->num * bufs->size);
    if (p == NULL) {
        return NULL;
    }

    ll = &chain;

    for (i = 0; i < bufs->num; i++) {

        b = ngx_calloc_buf(pool);
        if (b == NULL) {
            return NULL;
        }

        /*
         * set by ngx_calloc_buf():
         *
         *     b->file_pos = 0;
         *     b->file_last = 0;
         *     b->file = NULL;
         *     b->shadow = NULL;
         *     b->tag = 0;
         *     and flags
         *
         */

        b->pos = p;
        b->last = p;
        b->temporary = 1;

        b->start = p;
        p += bufs->size;
        b->end = p;

        cl = ngx_alloc_chain_link(pool);
        if (cl == NULL) {
            return NULL;
        }

        cl->buf = b;
        *ll = cl;
        ll = &cl->next;
    }

    *ll = NULL;

    return chain;
}
```

```c


ngx_int_t
ngx_chain_add_copy(ngx_pool_t *pool, ngx_chain_t **chain, ngx_chain_t *in)
{
    ngx_chain_t  *cl, **ll;

    ll = chain;

    for (cl = *chain; cl; cl = cl->next) {
        ll = &cl->next;
    }

    while (in) {
        cl = ngx_alloc_chain_link(pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        cl->buf = in->buf;
        *ll = cl;
        ll = &cl->next;
        in = in->next;
    }

    *ll = NULL;

    return NGX_OK;
}


ngx_chain_t *
ngx_chain_get_free_buf(ngx_pool_t *p, ngx_chain_t **free)
{
    ngx_chain_t  *cl;

    if (*free) {
        cl = *free;
        *free = cl->next;
        cl->next = NULL;
        return cl;
    }

    cl = ngx_alloc_chain_link(p);
    if (cl == NULL) {
        return NULL;
    }

    cl->buf = ngx_calloc_buf(p);
    if (cl->buf == NULL) {
        return NULL;
    }

    cl->next = NULL;

    return cl;
}


void
ngx_chain_update_chains(ngx_pool_t *p, ngx_chain_t **free, ngx_chain_t **busy,
    ngx_chain_t **out, ngx_buf_tag_t tag)
{
    ngx_chain_t  *cl;

    if (*busy == NULL) {
        *busy = *out;

    } else {
        for (cl = *busy; cl->next; cl = cl->next) { /* void */ }

        cl->next = *out;
    }

    *out = NULL;

```

```
    while (*busy) {
        cl = *busy;

        if (ngx_buf_size(cl->buf) != 0) {
            break;
        }

        if (cl->buf->tag != tag) {
            *busy = cl->next;
            ngx_free_chain(p, cl);
            continue;
        }

        cl->buf->pos = cl->buf->start;
        cl->buf->last = cl->buf->start;

        *busy = cl->next;
        cl->next = *free;
        *free = cl;
    }
}


off_t
ngx_chain_coalesce_file(ngx_chain_t **in, off_t limit)
{
    off_t         total, size, aligned, fprev;
    ngx_fd_t      fd;
    ngx_chain_t  *cl;

    total = 0;

    cl = *in;
    fd = cl->buf->file->fd;

    do {
        size = cl->buf->file_last - cl->buf->file_pos;

        if (size > limit - total) {
            size = limit - total;

            aligned = (cl->buf->file_pos + size + ngx_pagesize - 1)
                      & ~((off_t) ngx_pagesize - 1);

            if (aligned <= cl->buf->file_last) {
                size = aligned - cl->buf->file_pos;
            }
        }

        total += size;
        fprev = cl->buf->file_pos + size;
        cl = cl->next;

    } while (cl
             && cl->buf->in_file
             && total < limit
             && fd == cl->buf->file->fd
             && fprev == cl->buf->file_pos);

    *in = cl;

    return total;
}


ngx_chain_t *
ngx_chain_update_sent(ngx_chain_t *in, off_t sent)
{
    off_t  size;

    for ( /* void */ ; in; in = in->next) {

        if (ngx_buf_special(in->buf)) {
            continue;
        }

```

```
276         if (sent == 0) {
277             break;
278         }
279 
280         size = ngx_buf_size(in->buf);
281 
282         if (sent >= size) {
283             sent -= size;
284 
285             if (ngx_buf_in_memory(in->buf)) {
286                 in->buf->pos = in->buf->last;
287             }
288 
289             if (in->buf->in_file) {
290                 in->buf->file_pos = in->buf->file_last;
291             }
292 
293             continue;
294         }
295 
296         if (ngx_buf_in_memory(in->buf)) {
297             in->buf->pos += (size_t) sent;
298         }
299 
300         if (in->buf->in_file) {
301             in->buf->file_pos += sent;
302         }
303 
304         break;
305     }
306 
307     return in;
308 }
```

One Level Up Top Level

src/core/ngx_output_chain.c - nginx-1.7.10

Functions defined

- ngx_chain_writer
- ngx_output_chain
- ngx_output_chain_add_copy
- ngx_output_chain_align_file_buf
- ngx_output_chain_as_is
- ngx_output_chain_copy_buf
- ngx_output_chain_get_buf

Macros defined

- NGX_NONE

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #if 0
14  #define NGX_SENDFILE_LIMIT  4096
15  #endif
16
17  /*
18  * When DIRECTIO is enabled FreeBSD, Solaris, and MacOSX read directly
19  * to an application memory from a device if parameters are aligned
20  * to device sector boundary (512 bytes).  They fallback to usual read
21  * operation if the parameters are not aligned.
22  * Linux allows DIRECTIO only if the parameters are aligned to a filesystem
23  * sector boundary, otherwise it returns EINVAL.  The sector size is
24  * usually 512 bytes, however, on XFS it may be 4096 bytes.
25  */
26
27  #define NGX_NONE            1
28
29
30  static ngx_inline ngx_int_t
31      ngx_output_chain_as_is(ngx_output_chain_ctx_t *ctx, ngx_buf_t *buf);
32  static ngx_int_t ngx_output_chain_add_copy(ngx_pool_t *pool,
33      ngx_chain_t **chain, ngx_chain_t *in);
34  static ngx_int_t ngx_output_chain_align_file_buf(ngx_output_chain_ctx_t *ctx,
35      off_t bsize);
36  static ngx_int_t ngx_output_chain_get_buf(ngx_output_chain_ctx_t *ctx,
37      off_t bsize);
38  static ngx_int_t ngx_output_chain_copy_buf(ngx_output_chain_ctx_t *ctx);
39
40
41  ngx_int_t
42  ngx_output_chain(ngx_output_chain_ctx_t *ctx, ngx_chain_t *in)
43  {
```

```
    off_t         bsize;
    ngx_int_t     rc, last;
    ngx_chain_t  *cl, *out, **last_out;

    if (ctx->in == NULL && ctx->busy == NULL
#if (NGX_HAVE_FILE_AIO)
        && !ctx->aio
#endif
       )
    {
        /*
         * the short path for the case when the ctx->in and ctx->busy chains
         * are empty, the incoming chain is empty too or has the single buf
         * that does not require the copy
         */

        if (in == NULL) {
            return ctx->output_filter(ctx->filter_ctx, in);
        }

        if (in->next == NULL
#if (NGX_SENDFILE_LIMIT)
            && !(in->buf->in_file && in->buf->file_last > NGX_SENDFILE_LIMIT)
#endif
            && ngx_output_chain_as_is(ctx, in->buf))
        {
            return ctx->output_filter(ctx->filter_ctx, in);
        }
    }

    /* add the incoming buf to the chain ctx->in */

    if (in) {
        if (ngx_output_chain_add_copy(ctx->pool, &ctx->in, in) == NGX_ERROR) {
            return NGX_ERROR;
        }
    }

    out = NULL;
    last_out = &out;
    last = NGX_NONE;

    for ( ;; ) {

#if (NGX_HAVE_FILE_AIO)
        if (ctx->aio) {
            return NGX_AGAIN;
        }
#endif

        while (ctx->in) {

            /*
             * cycle while there are the ctx->in bufs
             * and there are the free output bufs to copy in
             */

             bsize = ngx_buf_size(ctx->in->buf);

             if (bsize == 0 && !ngx_buf_special(ctx->in->buf)) {

                 ngx_log_error(NGX_LOG_ALERT, ctx->pool->log, 0,
                               "zero size buf in output "
                               "t:%d r:%d f:%d %p %p-%p %p %O-%O",
                               ctx->in->buf->temporary,
                               ctx->in->buf->recycled,
                               ctx->in->buf->in_file,
                               ctx->in->buf->start,
                               ctx->in->buf->pos,
                               ctx->in->buf->last,
                               ctx->in->buf->file,
                               ctx->in->buf->file_pos,
                               ctx->in->buf->file_last);

                 ngx_debug_point();

```

```
            ctx->in = ctx->in->next;

            continue;
        }

        if (ngx_output_chain_as_is(ctx, ctx->in->buf)) {

            /* move the chain link to the output chain */

            cl = ctx->in;
            ctx->in = cl->next;

            *last_out = cl;
            last_out = &cl->next;
            cl->next = NULL;

            continue;
        }

        if (ctx->buf == NULL) {

            rc = ngx_output_chain_align_file_buf(ctx, bsize);

            if (rc == NGX_ERROR) {
                return NGX_ERROR;
            }

            if (rc != NGX_OK) {

                if (ctx->free) {

                    /* get the free buf */

                    cl = ctx->free;
                    ctx->buf = cl->buf;
                    ctx->free = cl->next;

                    ngx_free_chain(ctx->pool, cl);

                } else if (out || ctx->allocated == ctx->bufs.num) {

                    break;

                } else if (ngx_output_chain_get_buf(ctx, bsize) != NGX_OK) {
                    return NGX_ERROR;
                }
            }
        }

        rc = ngx_output_chain_copy_buf(ctx);

        if (rc == NGX_ERROR) {
            return rc;
        }

        if (rc == NGX_AGAIN) {
            if (out) {
                break;
            }

            return rc;
        }

        /* delete the completed buf from the ctx->in chain */

        if (ngx_buf_size(ctx->in->buf) == 0) {
            ctx->in = ctx->in->next;
        }

        cl = ngx_alloc_chain_link(ctx->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        cl->buf = ctx->buf;
        cl->next = NULL;
```

```
            *last_out = cl;
            last_out = &cl->next;
            ctx->buf = NULL;
        }

        if (out == NULL && last != NGX_NONE) {

            if (ctx->in) {
                return NGX_AGAIN;
            }

            return last;
        }

        last = ctx->output_filter(ctx->filter_ctx, out);

        if (last == NGX_ERROR || last == NGX_DONE) {
            return last;
        }

        ngx_chain_update_chains(ctx->pool, &ctx->free, &ctx->busy, &out,
                                ctx->tag);
        last_out = &out;
    }
}


static ngx_inline ngx_int_t
ngx_output_chain_as_is(ngx_output_chain_ctx_t *ctx, ngx_buf_t *buf)
{
    ngx_uint_t  sendfile;

    if (ngx_buf_special(buf)) {
        return 1;
    }

    if (buf->in_file && buf->file->directio) {
        return 0;
    }

    sendfile = ctx->sendfile;

#if (NGX_SENDFILE_LIMIT)

    if (buf->in_file && buf->file_pos >= NGX_SENDFILE_LIMIT) {
        sendfile = 0;
    }

#endif

    if (!sendfile) {

        if (!ngx_buf_in_memory(buf)) {
            return 0;
        }

        buf->in_file = 0;
    }

    if (ctx->need_in_memory && !ngx_buf_in_memory(buf)) {
        return 0;
    }

    if (ctx->need_in_temp && (buf->memory || buf->mmap)) {
        return 0;
    }

    return 1;
}


static ngx_int_t
ngx_output_chain_add_copy(ngx_pool_t *pool, ngx_chain_t **chain,
    ngx_chain_t *in)
{
    ngx_chain_t  *cl, **ll;
```

```
#if (NGX_SENDFILE_LIMIT)
    ngx_buf_t    *b, *buf;
#endif

    ll = chain;

    for (cl = *chain; cl; cl = cl->next) {
        ll = &cl->next;
    }

    while (in) {

        cl = ngx_alloc_chain_link(pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

#if (NGX_SENDFILE_LIMIT)

        buf = in->buf;

        if (buf->in_file
            && buf->file_pos < NGX_SENDFILE_LIMIT
            && buf->file_last > NGX_SENDFILE_LIMIT)
        {
            /* split a file buf on two bufs by the sendfile limit */

            b = ngx_calloc_buf(pool);
            if (b == NULL) {
                return NGX_ERROR;
            }

            ngx_memcpy(b, buf, sizeof(ngx_buf_t));

            if (ngx_buf_in_memory(buf)) {
                buf->pos += (ssize_t) (NGX_SENDFILE_LIMIT - buf->file_pos);
                b->last = buf->pos;
            }

            buf->file_pos = NGX_SENDFILE_LIMIT;
            b->file_last = NGX_SENDFILE_LIMIT;

            cl->buf = b;

        } else {
            cl->buf = buf;
            in = in->next;
        }

#else
        cl->buf = in->buf;
        in = in->next;

#endif

        cl->next = NULL;
        *ll = cl;
        ll = &cl->next;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_output_chain_align_file_buf(ngx_output_chain_ctx_t *ctx, off_t bsize)
{
    size_t      size;
    ngx_buf_t  *in;

    in = ctx->in->buf;

    if (in->file == NULL || !in->file->directio) {
        return NGX_DECLINED;
    }

```

```
    ctx->directio = 1;

    size = (size_t) (in->file_pos - (in->file_pos & ~(ctx->alignment - 1)));

    if (size == 0) {

        if (bsize >= (off_t) ctx->bufs.size) {
            return NGX_DECLINED;
        }

        size = (size_t) bsize;

    } else {
        size = (size_t) ctx->alignment - size;

        if ((off_t) size > bsize) {
            size = (size_t) bsize;
        }
    }

    ctx->buf = ngx_create_temp_buf(ctx->pool, size);
    if (ctx->buf == NULL) {
        return NGX_ERROR;
    }

    /*
     * we do not set ctx->buf->tag, because we do not want
     * to reuse the buf via ctx->free list
     */

#if (NGX_HAVE_ALIGNED_DIRECTIO)
    ctx->unaligned = 1;
#endif

    return NGX_OK;
}


static ngx_int_t
ngx_output_chain_get_buf(ngx_output_chain_ctx_t *ctx, off_t bsize)
{
    size_t       size;
    ngx_buf_t   *b, *in;
    ngx_uint_t   recycled;

    in = ctx->in->buf;
    size = ctx->bufs.size;
    recycled = 1;

    if (in->last_in_chain) {

        if (bsize < (off_t) size) {

            /*
             * allocate a small temp buf for a small last buf
             * or its small last part
             */

            size = (size_t) bsize;
            recycled = 0;

        } else if (!ctx->directio
                   && ctx->bufs.num == 1
                   && (bsize < (off_t) (size + size / 4)))
        {
            /*
             * allocate a temp buf that equals to a last buf,
             * if there is no directio, the last buf size is lesser
             * than 1.25 of bufs.size and the temp buf is single
             */

            size = (size_t) bsize;
            recycled = 0;
        }
    }

```

```
424     b = ngx_calloc_buf(ctx->pool);
425     if (b == NULL) {
426         return NGX_ERROR;
427     }
428
429     if (ctx->directio) {
430
431         /*
432          * allocate block aligned to a disk sector size to enable
433          * userland buffer direct usage conjunctly with directio
434          */
435
436         b->start = ngx_pmemalign(ctx->pool, size, (size_t) ctx->alignment);
437         if (b->start == NULL) {
438             return NGX_ERROR;
439         }
440
441     } else {
442         b->start = ngx_palloc(ctx->pool, size);
443         if (b->start == NULL) {
444             return NGX_ERROR;
445         }
446     }
447
448     b->pos = b->start;
449     b->last = b->start;
450     b->end = b->last + size;
451     b->temporary = 1;
452     b->tag = ctx->tag;
453     b->recycled = recycled;
454
455     ctx->buf = b;
456     ctx->allocated++;
457
458     return NGX_OK;
459 }
460
461
462 static ngx_int_t
463 ngx_output_chain_copy_buf(ngx_output_chain_ctx_t *ctx)
464 {
465     off_t        size;
466     ssize_t      n;
467     ngx_buf_t   *src, *dst;
468     ngx_uint_t   sendfile;
469
470     src = ctx->in->buf;
471     dst = ctx->buf;
472
473     size = ngx_buf_size(src);
474     size = ngx_min(size, dst->end - dst->pos);
475
476     sendfile = ctx->sendfile & !ctx->directio;
477
478 #if (NGX_SENDFILE_LIMIT)
479
480     if (src->in_file && src->file_pos >= NGX_SENDFILE_LIMIT) {
481         sendfile = 0;
482     }
483
484 #endif
485
486     if (ngx_buf_in_memory(src)) {
487         ngx_memcpy(dst->pos, src->pos, (size_t) size);
488         src->pos += (size_t) size;
489         dst->last += (size_t) size;
490
491         if (src->in_file) {
492
493             if (sendfile) {
494                 dst->in_file = 1;
495                 dst->file = src->file;
496                 dst->file_pos = src->file_pos;
497                 dst->file_last = src->file_pos + size;
498
499             } else {
```

```
                dst->in_file = 0;
            }

            src->file_pos += size;

        } else {
            dst->in_file = 0;
        }

        if (src->pos == src->last) {
            dst->flush = src->flush;
            dst->last_buf = src->last_buf;
            dst->last_in_chain = src->last_in_chain;
        }

    } else {

#if (NGX_HAVE_ALIGNED_DIRECTIO)

        if (ctx->unaligned) {
            if (ngx_directio_off(src->file->fd) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_ALERT, ctx->pool->log, ngx_errno,
                              ngx_directio_off_n " \"%s\" failed",
                              src->file->name.data);
            }
        }

#endif

#if (NGX_HAVE_FILE_AIO)

        if (ctx->aio_handler) {
            n = ngx_file_aio_read(src->file, dst->pos, (size_t) size,
                                  src->file_pos, ctx->pool);
            if (n == NGX_AGAIN) {
                ctx->aio_handler(ctx, src->file);
                return NGX_AGAIN;
            }

        } else {
            n = ngx_read_file(src->file, dst->pos, (size_t) size,
                              src->file_pos);
        }
#else

        n = ngx_read_file(src->file, dst->pos, (size_t) size, src->file_pos);

#endif

#if (NGX_HAVE_ALIGNED_DIRECTIO)

        if (ctx->unaligned) {
            ngx_err_t  err;

            err = ngx_errno;

            if (ngx_directio_on(src->file->fd) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_ALERT, ctx->pool->log, ngx_errno,
                              ngx_directio_on_n " \"%s\" failed",
                              src->file->name.data);
            }

            ngx_set_errno(err);

            ctx->unaligned = 0;
        }

#endif

        if (n == NGX_ERROR) {
            return (ngx_int_t) n;
        }

        if (n != size) {
            ngx_log_error(NGX_LOG_ALERT, ctx->pool->log, 0,
                          ngx_read_file_n " read only %z of %O from \"%s\"",
```

```
                          n, size, src->file->name.data);
            return NGX_ERROR;
        }

        dst->last += n;

        if (sendfile) {
            dst->in_file = 1;
            dst->file = src->file;
            dst->file_pos = src->file_pos;
            dst->file_last = src->file_pos + n;

        } else {
            dst->in_file = 0;
        }

        src->file_pos += n;

        if (src->file_pos == src->file_last) {
            dst->flush = src->flush;
            dst->last_buf = src->last_buf;
            dst->last_in_chain = src->last_in_chain;
        }
    }

    return NGX_OK;
}


ngx_int_t
ngx_chain_writer(void *data, ngx_chain_t *in)
{
    ngx_chain_writer_ctx_t *ctx = data;

    off_t              size;
    ngx_chain_t       *cl;
    ngx_connection_t  *c;

    c = ctx->connection;

    for (size = 0; in; in = in->next) {

#if 1
        if (ngx_buf_size(in->buf) == 0 && !ngx_buf_special(in->buf)) {
            ngx_debug_point();
        }
#endif

        size += ngx_buf_size(in->buf);

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, c->log, 0,
                       "chain writer buf fl:%d s:%uO",
                       in->buf->flush, ngx_buf_size(in->buf));

        cl = ngx_alloc_chain_link(ctx->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        cl->buf = in->buf;
        cl->next = NULL;
        *ctx->last = cl;
        ctx->last = &cl->next;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, c->log, 0,
                   "chain writer in: %p", ctx->out);

    for (cl = ctx->out; cl; cl = cl->next) {

#if 1
        if (ngx_buf_size(cl->buf) == 0 && !ngx_buf_special(cl->buf)) {
            ngx_debug_point();
        }

#endif
```

```
652
653         size += ngx_buf_size(cl->buf);
654     }
655
656     if (size == 0 && !c->buffered) {
657         return NGX_OK;
658     }
659
660     ctx->out = c->send_chain(c, ctx->out, ctx->limit);
661
662     ngx_log_debug1(NGX_LOG_DEBUG_CORE, c->log, 0,
663                    "chain writer out: %p", ctx->out);
664
665     if (ctx->out == NGX_CHAIN_ERROR) {
666         return NGX_ERROR;
667     }
668
669     if (ctx->out == NULL) {
670         ctx->last = &ctx->out;
671
672         if (!c->buffered) {
673             return NGX_OK;
674         }
675     }
676
677     return NGX_AGAIN;
678 }
```

One Level Up Top Level

src/http/ngx_http_parse.c - nginx-1.7.10

Global variables defined

- usual

Functions defined

- ngx_http_arg
- ngx_http_parse_chunked
- ngx_http_parse_complex_uri
- ngx_http_parse_header_line
- ngx_http_parse_multi_header_lines
- ngx_http_parse_request_line
- ngx_http_parse_set_cookie_lines
- ngx_http_parse_status_line
- ngx_http_parse_unsafe_uri
- ngx_http_parse_uri
- ngx_http_split_args

Macros defined

- ngx_str3Ocmp
- ngx_str3Ocmp
- ngx_str3_cmp
- ngx_str3_cmp
- ngx_str4cmp
- ngx_str4cmp
- ngx_str5cmp
- ngx_str5cmp
- ngx_str6cmp
- ngx_str6cmp
- ngx_str7_cmp
- ngx_str7_cmp
- ngx_str8cmp
- ngx_str8cmp
- ngx_str9cmp

- ngx_str9cmp

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static uint32_t  usual[] = {
14      0xffffdbfe, /* 1111 1111 1111 1111  1101 1011 1111 1110 */
15
16                  /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
17      0x7fff37d6, /* 0111 1111 1111 1111  0011 0111 1101 0110 */
18
19                  /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
20  #if (NGX_WIN32)
21      0xefffffff, /* 1110 1111 1111 1111  1111 1111 1111 1111 */
22  #else
23      0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
24  #endif
25
26                  /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
27      0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
28
29      0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
30      0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
31      0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
32      0xffffffff  /* 1111 1111 1111 1111  1111 1111 1111 1111 */
33  };
34
35
36  #if (NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED)
37
38  #define ngx_str3_cmp(m, c0, c1, c2, c3)                                     \
39      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)
40
41  #define ngx_str3Ocmp(m, c0, c1, c2, c3)                                     \
42      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)
43
44  #define ngx_str4cmp(m, c0, c1, c2, c3)                                      \
45      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)
46
47  #define ngx_str5cmp(m, c0, c1, c2, c3, c4)                                  \
48      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)           \
49          && m[4] == c4
50
51  #define ngx_str6cmp(m, c0, c1, c2, c3, c4, c5)                              \
52      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)           \
53          && (((uint32_t *) m)[1] & 0xffff) == ((c5 << 8) | c4)
54
55  #define ngx_str7_cmp(m, c0, c1, c2, c3, c4, c5, c6, c7)                     \
56      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)           \
57          && ((uint32_t *) m)[1] == ((c7 << 24) | (c6 << 16) | (c5 << 8) | c4)
58
59  #define ngx_str8cmp(m, c0, c1, c2, c3, c4, c5, c6, c7)                      \
60      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)           \
61          && ((uint32_t *) m)[1] == ((c7 << 24) | (c6 << 16) | (c5 << 8) | c4)
62
63  #define ngx_str9cmp(m, c0, c1, c2, c3, c4, c5, c6, c7, c8)                  \
64      *(uint32_t *) m == ((c3 << 24) | (c2 << 16) | (c1 << 8) | c0)           \
65          && ((uint32_t *) m)[1] == ((c7 << 24) | (c6 << 16) | (c5 << 8) | c4)  \
66          && m[8] == c8
67
68  #else /* !(NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED) */
69
70  #define ngx_str3_cmp(m, c0, c1, c2, c3)                                     \
```

```
    m[0] == c0 && m[1] == c1 && m[2] == c2

#define ngx_str3Ocmp(m, c0, c1, c2, c3)                                      \
    m[0] == c0 && m[2] == c2 && m[3] == c3

#define ngx_str4cmp(m, c0, c1, c2, c3)                                       \
    m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3

#define ngx_str5cmp(m, c0, c1, c2, c3, c4)                                   \
    m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3 && m[4] == c4

#define ngx_str6cmp(m, c0, c1, c2, c3, c4, c5)                               \
    m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3                     \
        && m[4] == c4 && m[5] == c5

#define ngx_str7_cmp(m, c0, c1, c2, c3, c4, c5, c6, c7)                      \
    m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3                     \
        && m[4] == c4 && m[5] == c5 && m[6] == c6

#define ngx_str8cmp(m, c0, c1, c2, c3, c4, c5, c6, c7)                       \
    m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3                     \
        && m[4] == c4 && m[5] == c5 && m[6] == c6 && m[7] == c7

#define ngx_str9cmp(m, c0, c1, c2, c3, c4, c5, c6, c7, c8)                   \
    m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3                     \
        && m[4] == c4 && m[5] == c5 && m[6] == c6 && m[7] == c7 && m[8] == c8

#endif


/* gcc, icc, msvc and others compile these switches as an jump table */

ngx_int_t
ngx_http_parse_request_line(ngx_http_request_t *r, ngx_buf_t *b)
{
    u_char  c, ch, *p, *m;
    enum {
        sw_start = 0,
        sw_method,
        sw_spaces_before_uri,
        sw_schema,
        sw_schema_slash,
        sw_schema_slash_slash,
        sw_host_start,
        sw_host,
        sw_host_end,
        sw_host_ip_literal,
        sw_port,
        sw_host_http_09,
        sw_after_slash_in_uri,
        sw_check_uri,
        sw_check_uri_http_09,
        sw_uri,
        sw_http_09,
        sw_http_H,
        sw_http_HT,
        sw_http_HTT,
        sw_http_HTTP,
        sw_first_major_digit,
        sw_major_digit,
        sw_first_minor_digit,
        sw_minor_digit,
        sw_spaces_after_digit,
        sw_almost_done
    } state;

    state = r->state;

    for (p = b->pos; p < b->last; p++) {
        ch = *p;

        switch (state) {

        /* HTTP methods: GET, HEAD, POST */
        case sw_start:
            r->request_start = p;
```

```
147
148             if (ch == CR || ch == LF) {
149                 break;
150             }
151
152             if ((ch < 'A' || ch > 'Z') && ch != '_') {
153                 return NGX_HTTP_PARSE_INVALID_METHOD;
154             }
155
156             state = sw_method;
157             break;
158
159         case sw_method:
160             if (ch == ' ') {
161                 r->method_end = p - 1;
162                 m = r->request_start;
163
164                 switch (p - m) {
165
166                 case 3:
167                     if (ngx_str3_cmp(m, 'G', 'E', 'T', ' ')) {
168                         r->method = NGX_HTTP_GET;
169                         break;
170                     }
171
172                     if (ngx_str3_cmp(m, 'P', 'U', 'T', ' ')) {
173                         r->method = NGX_HTTP_PUT;
174                         break;
175                     }
176
177                     break;
178
179                 case 4:
180                     if (m[1] == 'O') {
181
182                         if (ngx_str3Ocmp(m, 'P', 'O', 'S', 'T')) {
183                             r->method = NGX_HTTP_POST;
184                             break;
185                         }
186
187                         if (ngx_str3Ocmp(m, 'C', 'O', 'P', 'Y')) {
188                             r->method = NGX_HTTP_COPY;
189                             break;
190                         }
191
192                         if (ngx_str3Ocmp(m, 'M', 'O', 'V', 'E')) {
193                             r->method = NGX_HTTP_MOVE;
194                             break;
195                         }
196
197                         if (ngx_str3Ocmp(m, 'L', 'O', 'C', 'K')) {
198                             r->method = NGX_HTTP_LOCK;
199                             break;
200                         }
201
202                     } else {
203
204                         if (ngx_str4cmp(m, 'H', 'E', 'A', 'D')) {
205                             r->method = NGX_HTTP_HEAD;
206                             break;
207                         }
208                     }
209
210                     break;
211
212                 case 5:
213                     if (ngx_str5cmp(m, 'M', 'K', 'C', 'O', 'L')) {
214                         r->method = NGX_HTTP_MKCOL;
215                         break;
216                     }
217
218                     if (ngx_str5cmp(m, 'P', 'A', 'T', 'C', 'H')) {
219                         r->method = NGX_HTTP_PATCH;
220                         break;
221                     }
222
```

```
                if (ngx_str5cmp(m, 'T', 'R', 'A', 'C', 'E')) {
                    r->method = NGX_HTTP_TRACE;
                    break;
                }

                break;

            case 6:
                if (ngx_str6cmp(m, 'D', 'E', 'L', 'E', 'T', 'E')) {
                    r->method = NGX_HTTP_DELETE;
                    break;
                }

                if (ngx_str6cmp(m, 'U', 'N', 'L', 'O', 'C', 'K')) {
                    r->method = NGX_HTTP_UNLOCK;
                    break;
                }

                break;

            case 7:
                if (ngx_str7_cmp(m, 'O', 'P', 'T', 'I', 'O', 'N', 'S', ' '))
                {
                    r->method = NGX_HTTP_OPTIONS;
                }

                break;

            case 8:
                if (ngx_str8cmp(m, 'P', 'R', 'O', 'P', 'F', 'I', 'N', 'D'))
                {
                    r->method = NGX_HTTP_PROPFIND;
                }

                break;

            case 9:
                if (ngx_str9cmp(m,
                        'P', 'R', 'O', 'P', 'P', 'A', 'T', 'C', 'H'))
                {
                    r->method = NGX_HTTP_PROPPATCH;
                }

                break;
            }

            state = sw_spaces_before_uri;
            break;
        }

        if ((ch < 'A' || ch > 'Z') && ch != '_') {
            return NGX_HTTP_PARSE_INVALID_METHOD;
        }

        break;

    /* space* before URI */
    case sw_spaces_before_uri:

        if (ch == '/') {
            r->uri_start = p;
            state = sw_after_slash_in_uri;
            break;
        }

        c = (u_char) (ch | 0x20);
        if (c >= 'a' && c <= 'z') {
            r->schema_start = p;
            state = sw_schema;
            break;
        }

        switch (ch) {
        case ' ':
            break;
        default:
```

```
            return NGX_HTTP_PARSE_INVALID_REQUEST;
        }
        break;

    case sw_schema:

        c = (u_char) (ch | 0x20);
        if (c >= 'a' && c <= 'z') {
            break;
        }

        switch (ch) {
        case ':':
            r->schema_end = p;
            state = sw_schema_slash;
            break;
        default:
            return NGX_HTTP_PARSE_INVALID_REQUEST;
        }
        break;

    case sw_schema_slash:
        switch (ch) {
        case '/':
            state = sw_schema_slash_slash;
            break;
        default:
            return NGX_HTTP_PARSE_INVALID_REQUEST;
        }
        break;

    case sw_schema_slash_slash:
        switch (ch) {
        case '/':
            state = sw_host_start;
            break;
        default:
            return NGX_HTTP_PARSE_INVALID_REQUEST;
        }
        break;

    case sw_host_start:

        r->host_start = p;

        if (ch == '[') {
            state = sw_host_ip_literal;
            break;
        }

        state = sw_host;

        /* fall through */

    case sw_host:

        c = (u_char) (ch | 0x20);
        if (c >= 'a' && c <= 'z') {
            break;
        }

        if ((ch >= '0' && ch <= '9') || ch == '.' || ch == '-') {
            break;
        }

        /* fall through */

    case sw_host_end:

        r->host_end = p;

        switch (ch) {
        case ':':
            state = sw_port;
            break;
        case '/':
```

```
                r->uri_start = p;
                state = sw_after_slash_in_uri;
                break;
            case ' ':
                /*
                 * use single "/" from request line to preserve pointers,
                 * if request line will be copied to large client buffer
                 */
                r->uri_start = r->schema_end + 1;
                r->uri_end = r->schema_end + 2;
                state = sw_host_http_09;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        case sw_host_ip_literal:

            if (ch >= '0' && ch <= '9') {
                break;
            }

            c = (u_char) (ch | 0x20);
            if (c >= 'a' && c <= 'z') {
                break;
            }

            switch (ch) {
            case ':':
                break;
            case ']':
                state = sw_host_end;
                break;
            case '-':
            case '.':
            case '_':
            case '~':
                /* unreserved */
                break;
            case '!':
            case '$':
            case '&':
            case '\'':
            case '(':
            case ')':
            case '*':
            case '+':
            case ',':
            case ';':
            case '=':
                /* sub-delims */
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        case sw_port:
            if (ch >= '0' && ch <= '9') {
                break;
            }

            switch (ch) {
            case '/':
                r->port_end = p;
                r->uri_start = p;
                state = sw_after_slash_in_uri;
                break;
            case ' ':
                r->port_end = p;
                /*
                 * use single "/" from request line to preserve pointers,
                 * if request line will be copied to large client buffer
                 */
                r->uri_start = r->schema_end + 1;
```

```
                r->uri_end = r->schema_end + 2;
                state = sw_host_http_09;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        /* space+ after "http://host[:port] " */
        case sw_host_http_09:
            switch (ch) {
            case ' ':
                break;
            case CR:
                r->http_minor = 9;
                state = sw_almost_done;
                break;
            case LF:
                r->http_minor = 9;
                goto done;
            case 'H':
                r->http_protocol.data = p;
                state = sw_http_H;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;


        /* check "/.", "//", "%", and "\" (Win32) in URI */
        case sw_after_slash_in_uri:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                state = sw_check_uri;
                break;
            }

            switch (ch) {
            case ' ':
                r->uri_end = p;
                state = sw_check_uri_http_09;
                break;
            case CR:
                r->uri_end = p;
                r->http_minor = 9;
                state = sw_almost_done;
                break;
            case LF:
                r->uri_end = p;
                r->http_minor = 9;
                goto done;
            case '.':
                r->complex_uri = 1;
                state = sw_uri;
                break;
            case '%':
                r->quoted_uri = 1;
                state = sw_uri;
                break;
            case '/':
                r->complex_uri = 1;
                state = sw_uri;
                break;
#if (NGX_WIN32)
            case '\\':
                r->complex_uri = 1;
                state = sw_uri;
                break;
#endif
            case '?':
                r->args_start = p + 1;
                state = sw_uri;
                break;
            case '#':
                r->complex_uri = 1;
```

```
                state = sw_uri;
                break;
            case '+':
                r->plus_in_uri = 1;
                break;
            case '\0':
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            default:
                state = sw_check_uri;
                break;
            }
            break;

        /* check "/", "%" and "\" (Win32) in URI */
        case sw_check_uri:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                break;
            }

            switch (ch) {
            case '/':
#if (NGX_WIN32)
                if (r->uri_ext == p) {
                    r->complex_uri = 1;
                    state = sw_uri;
                    break;
                }
#endif
                r->uri_ext = NULL;
                state = sw_after_slash_in_uri;
                break;
            case '.':
                r->uri_ext = p + 1;
                break;
            case ' ':
                r->uri_end = p;
                state = sw_check_uri_http_09;
                break;
            case CR:
                r->uri_end = p;
                r->http_minor = 9;
                state = sw_almost_done;
                break;
            case LF:
                r->uri_end = p;
                r->http_minor = 9;
                goto done;
#if (NGX_WIN32)
            case '\\':
                r->complex_uri = 1;
                state = sw_after_slash_in_uri;
                break;
#endif
            case '%':
                r->quoted_uri = 1;
                state = sw_uri;
                break;
            case '?':
                r->args_start = p + 1;
                state = sw_uri;
                break;
            case '#':
                r->complex_uri = 1;
                state = sw_uri;
                break;
            case '+':
                r->plus_in_uri = 1;
                break;
            case '\0':
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        /* space+ after URI */
        case sw_check_uri_http_09:
```

```
603             switch (ch) {
604             case ' ':
605                 break;
606             case CR:
607                 r->http_minor = 9;
608                 state = sw_almost_done;
609                 break;
610             case LF:
611                 r->http_minor = 9;
612                 goto done;
613             case 'H':
614                 r->http_protocol.data = p;
615                 state = sw_http_H;
616                 break;
617             default:
618                 r->space_in_uri = 1;
619                 state = sw_check_uri;
620                 p--;
621                 break;
622             }
623             break;
624
625
626         /* URI */
627         case sw_uri:
628
629             if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
630                 break;
631             }
632
633             switch (ch) {
634             case ' ':
635                 r->uri_end = p;
636                 state = sw_http_09;
637                 break;
638             case CR:
639                 r->uri_end = p;
640                 r->http_minor = 9;
641                 state = sw_almost_done;
642                 break;
643             case LF:
644                 r->uri_end = p;
645                 r->http_minor = 9;
646                 goto done;
647             case '#':
648                 r->complex_uri = 1;
649                 break;
650             case '\0':
651                 return NGX_HTTP_PARSE_INVALID_REQUEST;
652             }
653             break;
654
655         /* space+ after URI */
656         case sw_http_09:
657             switch (ch) {
658             case ' ':
659                 break;
660             case CR:
661                 r->http_minor = 9;
662                 state = sw_almost_done;
663                 break;
664             case LF:
665                 r->http_minor = 9;
666                 goto done;
667             case 'H':
668                 r->http_protocol.data = p;
669                 state = sw_http_H;
670                 break;
671             default:
672                 r->space_in_uri = 1;
673                 state = sw_uri;
674                 p--;
675                 break;
676             }
677             break;
678
```

```
        case sw_http_H:
            switch (ch) {
            case 'T':
                state = sw_http_HT;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        case sw_http_HT:
            switch (ch) {
            case 'T':
                state = sw_http_HTT;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        case sw_http_HTT:
            switch (ch) {
            case 'P':
                state = sw_http_HTTP;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        case sw_http_HTTP:
            switch (ch) {
            case '/':
                state = sw_first_major_digit;
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        /* first digit of major HTTP version */
        case sw_first_major_digit:
            if (ch < '1' || ch > '9') {
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }

            r->http_major = ch - '0';
            state = sw_major_digit;
            break;

        /* major HTTP version or dot */
        case sw_major_digit:
            if (ch == '.') {
                state = sw_first_minor_digit;
                break;
            }

            if (ch < '0' || ch > '9') {
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }

            r->http_major = r->http_major * 10 + ch - '0';
            break;

        /* first digit of minor HTTP version */
        case sw_first_minor_digit:
            if (ch < '0' || ch > '9') {
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }

            r->http_minor = ch - '0';
            state = sw_minor_digit;
            break;

        /* minor HTTP version or end of request line */
        case sw_minor_digit:
```

```
            if (ch == CR) {
                state = sw_almost_done;
                break;
            }

            if (ch == LF) {
                goto done;
            }

            if (ch == ' ') {
                state = sw_spaces_after_digit;
                break;
            }

            if (ch < '0' || ch > '9') {
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }

            r->http_minor = r->http_minor * 10 + ch - '0';
            break;

        case sw_spaces_after_digit:
            switch (ch) {
            case ' ':
                break;
            case CR:
                state = sw_almost_done;
                break;
            case LF:
                goto done;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
            break;

        /* end of request line */
        case sw_almost_done:
            r->request_end = p - 1;
            switch (ch) {
            case LF:
                goto done;
            default:
                return NGX_HTTP_PARSE_INVALID_REQUEST;
            }
        }
    }

    b->pos = p;
    r->state = state;

    return NGX_AGAIN;

done:

    b->pos = p + 1;

    if (r->request_end == NULL) {
        r->request_end = p;
    }

    r->http_version = r->http_major * 1000 + r->http_minor;
    r->state = sw_start;

    if (r->http_version == 9 && r->method != NGX_HTTP_GET) {
        return NGX_HTTP_PARSE_INVALID_09_METHOD;
    }

    return NGX_OK;
}


ngx_int_t
ngx_http_parse_header_line(ngx_http_request_t *r, ngx_buf_t *b,
    ngx_uint_t allow_underscores)
{
    u_char      c, ch, *p;
```

```
    ngx_uint_t  hash, i;
    enum {
        sw_start = 0,
        sw_name,
        sw_space_before_value,
        sw_value,
        sw_space_after_value,
        sw_ignore_line,
        sw_almost_done,
        sw_header_almost_done
    } state;

    /* the last '\0' is not needed because string is zero terminated */

    static u_char  lowcase[] =
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0-\0\0" "0123456789\0\0\0\0\0\0"
        "\0abcdefghijklmnopqrstuvwxyz\0\0\0\0\0"
        "\0abcdefghijklmnopqrstuvwxyz\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
        "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0";

    state = r->state;
    hash = r->header_hash;
    i = r->lowcase_index;

    for (p = b->pos; p < b->last; p++) {
        ch = *p;

        switch (state) {

        /* first char */
        case sw_start:
            r->header_name_start = p;
            r->invalid_header = 0;

            switch (ch) {
            case CR:
                r->header_end = p;
                state = sw_header_almost_done;
                break;
            case LF:
                r->header_end = p;
                goto header_done;
            default:
                state = sw_name;

                c = lowcase[ch];

                if (c) {
                    hash = ngx_hash(0, c);
                    r->lowcase_header[0] = c;
                    i = 1;
                    break;
                }

                if (ch == '_') {
                    if (allow_underscores) {
                        hash = ngx_hash(0, ch);
                        r->lowcase_header[0] = ch;
                        i = 1;

                    } else {
                        r->invalid_header = 1;
                    }

                    break;
                }

                if (ch == '\0') {
                    return NGX_HTTP_PARSE_INVALID_HEADER;
                }

                r->invalid_header = 1;
```

```
                break;

            }
            break;

        /* header name */
        case sw_name:
            c = lowcase[ch];

            if (c) {
                hash = ngx_hash(hash, c);
                r->lowcase_header[i++] = c;
                i &= (NGX_HTTP_LC_HEADER_LEN - 1);
                break;
            }

            if (ch == '_') {
                if (allow_underscores) {
                    hash = ngx_hash(hash, ch);
                    r->lowcase_header[i++] = ch;
                    i &= (NGX_HTTP_LC_HEADER_LEN - 1);

                } else {
                    r->invalid_header = 1;
                }

                break;
            }

            if (ch == ':') {
                r->header_name_end = p;
                state = sw_space_before_value;
                break;
            }

            if (ch == CR) {
                r->header_name_end = p;
                r->header_start = p;
                r->header_end = p;
                state = sw_almost_done;
                break;
            }

            if (ch == LF) {
                r->header_name_end = p;
                r->header_start = p;
                r->header_end = p;
                goto done;
            }

            /* IIS may send the duplicate "HTTP/1.1 ..." lines */
            if (ch == '/'
                && r->upstream
                && p - r->header_name_start == 4
                && ngx_strncmp(r->header_name_start, "HTTP", 4) == 0)
            {
                state = sw_ignore_line;
                break;
            }

            if (ch == '\0') {
                return NGX_HTTP_PARSE_INVALID_HEADER;
            }

            r->invalid_header = 1;

            break;

        /* space* before header value */
        case sw_space_before_value:
            switch (ch) {
            case ' ':
                break;
            case CR:
                r->header_start = p;
```

```
                r->header_end = p;
                state = sw_almost_done;
                break;
            case LF:
                r->header_start = p;
                r->header_end = p;
                goto done;
            case '\0':
                return NGX_HTTP_PARSE_INVALID_HEADER;
            default:
                r->header_start = p;
                state = sw_value;
                break;
            }
            break;

        /* header value */
        case sw_value:
            switch (ch) {
            case ' ':
                r->header_end = p;
                state = sw_space_after_value;
                break;
            case CR:
                r->header_end = p;
                state = sw_almost_done;
                break;
            case LF:
                r->header_end = p;
                goto done;
            case '\0':
                return NGX_HTTP_PARSE_INVALID_HEADER;
            }
            break;

        /* space* before end of header line */
        case sw_space_after_value:
            switch (ch) {
            case ' ':
                break;
            case CR:
                state = sw_almost_done;
                break;
            case LF:
                goto done;
            case '\0':
                return NGX_HTTP_PARSE_INVALID_HEADER;
            default:
                state = sw_value;
                break;
            }
            break;

        /* ignore header line */
        case sw_ignore_line:
            switch (ch) {
            case LF:
                state = sw_start;
                break;
            default:
                break;
            }
            break;

        /* end of header line */
        case sw_almost_done:
            switch (ch) {
            case LF:
                goto done;
            case CR:
                break;
            default:
                return NGX_HTTP_PARSE_INVALID_HEADER;
            }
            break;

```

```
        /* end of header */
        case sw_header_almost_done:
            switch (ch) {
            case LF:
                goto header_done;
            default:
                return NGX_HTTP_PARSE_INVALID_HEADER;
            }
        }
    }

    b->pos = p;
    r->state = state;
    r->header_hash = hash;
    r->lowcase_index = i;

    return NGX_AGAIN;

done:

    b->pos = p + 1;
    r->state = sw_start;
    r->header_hash = hash;
    r->lowcase_index = i;

    return NGX_OK;

header_done:

    b->pos = p + 1;
    r->state = sw_start;

    return NGX_HTTP_PARSE_HEADER_DONE;
}


ngx_int_t
ngx_http_parse_uri(ngx_http_request_t *r)
{
    u_char  *p, ch;
    enum {
        sw_start = 0,
        sw_after_slash_in_uri,
        sw_check_uri,
        sw_uri
    } state;

    state = sw_start;

    for (p = r->uri_start; p != r->uri_end; p++) {

        ch = *p;

        switch (state) {

        case sw_start:

            if (ch != '/') {
                return NGX_ERROR;
            }

            state = sw_after_slash_in_uri;
            break;

        /* check "/.", "//", "%", and "\" (Win32) in URI */
        case sw_after_slash_in_uri:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                state = sw_check_uri;
                break;
            }

            switch (ch) {
            case ' ':
                r->space_in_uri = 1;
                state = sw_check_uri;
```

```
                break;
            case '.':
                r->complex_uri = 1;
                state = sw_uri;
                break;
            case '%':
                r->quoted_uri = 1;
                state = sw_uri;
                break;
            case '/':
                r->complex_uri = 1;
                state = sw_uri;
                break;
#if (NGX_WIN32)
            case '\\':
                r->complex_uri = 1;
                state = sw_uri;
                break;
#endif
            case '?':
                r->args_start = p + 1;
                state = sw_uri;
                break;
            case '#':
                r->complex_uri = 1;
                state = sw_uri;
                break;
            case '+':
                r->plus_in_uri = 1;
                break;
            default:
                state = sw_check_uri;
                break;
            }
            break;

        /* check "/", "%" and "\" (Win32) in URI */
        case sw_check_uri:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                break;
            }

            switch (ch) {
            case '/':
#if (NGX_WIN32)
                if (r->uri_ext == p) {
                    r->complex_uri = 1;
                    state = sw_uri;
                    break;
                }
#endif
                r->uri_ext = NULL;
                state = sw_after_slash_in_uri;
                break;
            case '.':
                r->uri_ext = p + 1;
                break;
            case ' ':
                r->space_in_uri = 1;
                break;
#if (NGX_WIN32)
            case '\\':
                r->complex_uri = 1;
                state = sw_after_slash_in_uri;
                break;
#endif
            case '%':
                r->quoted_uri = 1;
                state = sw_uri;
                break;
            case '?':
                r->args_start = p + 1;
                state = sw_uri;
                break;
            case '#':
```

```
                r->complex_uri = 1;
                state = sw_uri;
                break;
            case '+':
                r->plus_in_uri = 1;
                break;
            }
            break;

        /* URI */
        case sw_uri:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                break;
            }

            switch (ch) {
            case ' ':
                r->space_in_uri = 1;
                break;
            case '#':
                r->complex_uri = 1;
                break;
            }
            break;
        }
    }

    return NGX_OK;
}


ngx_int_t
ngx_http_parse_complex_uri(ngx_http_request_t *r, ngx_uint_t merge_slashes)
{
    u_char  c, ch, decoded, *p, *u;
    enum {
        sw_usual = 0,
        sw_slash,
        sw_dot,
        sw_dot_dot,
        sw_quoted,
        sw_quoted_second
    } state, quoted_state;

#if (NGX_SUPPRESS_WARN)
    decoded = '\0';
    quoted_state = sw_usual;
#endif

    state = sw_usual;
    p = r->uri_start;
    u = r->uri.data;
    r->uri_ext = NULL;
    r->args_start = NULL;

    ch = *p++;

    while (p <= r->uri_end) {

        /*
         * we use "ch = *p++" inside the cycle, but this operation is safe,
         * because after the URI there is always at least one character:
         * the line feed
         */

        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "s:%d in:'%Xd:%c'", state, ch, ch);

        switch (state) {

        case sw_usual:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                *u++ = ch;
                ch = *p++;
```

```
1287                 break;
1288             }
1289
1290             switch (ch) {
1291 #if (NGX_WIN32)
1292             case '\\':
1293                 if (u - 2 >= r->uri.data
1294                     && *(u - 1) == '.' && *(u - 2) != '.')
1295                 {
1296                     u--;
1297                 }
1298
1299                 r->uri_ext = NULL;
1300
1301                 if (p == r->uri_start + r->uri.len) {
1302
1303                     /*
1304                      * we omit the last "\" to cause redirect because
1305                      * the browsers do not treat "\" as "/" in relative URL path
1306                      */
1307
1308                     break;
1309                 }
1310
1311                 state = sw_slash;
1312                 *u++ = '/';
1313                 break;
1314 #endif
1315             case '/':
1316 #if (NGX_WIN32)
1317                 if (u - 2 >= r->uri.data
1318                     && *(u - 1) == '.' && *(u - 2) != '.')
1319                 {
1320                     u--;
1321                 }
1322 #endif
1323                 r->uri_ext = NULL;
1324                 state = sw_slash;
1325                 *u++ = ch;
1326                 break;
1327             case '%':
1328                 quoted_state = state;
1329                 state = sw_quoted;
1330                 break;
1331             case '?':
1332                 r->args_start = p;
1333                 goto args;
1334             case '#':
1335                 goto done;
1336             case '.':
1337                 r->uri_ext = u + 1;
1338                 *u++ = ch;
1339                 break;
1340             case '+':
1341                 r->plus_in_uri = 1;
1342                 /* fall through */
1343             default:
1344                 *u++ = ch;
1345                 break;
1346             }
1347
1348             ch = *p++;
1349             break;
1350
1351         case sw_slash:
1352
1353             if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
1354                 state = sw_usual;
1355                 *u++ = ch;
1356                 ch = *p++;
1357                 break;
1358             }
1359
1360             switch (ch) {
1361 #if (NGX_WIN32)
1362             case '\\':
```

```
                break;
#endif
            case '/':
                if (!merge_slashes) {
                    *u++ = ch;
                }
                break;
            case '.':
                state = sw_dot;
                *u++ = ch;
                break;
            case '%':
                quoted_state = state;
                state = sw_quoted;
                break;
            case '?':
                r->args_start = p;
                goto args;
            case '#':
                goto done;
            case '+':
                r->plus_in_uri = 1;
            default:
                state = sw_usual;
                *u++ = ch;
                break;
            }

            ch = *p++;
            break;

        case sw_dot:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                state = sw_usual;
                *u++ = ch;
                ch = *p++;
                break;
            }

            switch (ch) {
#if (NGX_WIN32)
            case '\\':
#endif
            case '/':
                state = sw_slash;
                u--;
                break;
            case '.':
                state = sw_dot_dot;
                *u++ = ch;
                break;
            case '%':
                quoted_state = state;
                state = sw_quoted;
                break;
            case '?':
                r->args_start = p;
                goto args;
            case '#':
                goto done;
            case '+':
                r->plus_in_uri = 1;
            default:
                state = sw_usual;
                *u++ = ch;
                break;
            }

            ch = *p++;
            break;

        case sw_dot_dot:

            if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
                state = sw_usual;
```

```
1439                *u++ = ch;
1440                ch = *p++;
1441                break;
1442            }
1443
1444            switch (ch) {
1445    #if (NGX_WIN32)
1446            case '\\':
1447    #endif
1448            case '/':
1449                state = sw_slash;
1450                u -= 5;
1451                for ( ;; ) {
1452                    if (u < r->uri.data) {
1453                        return NGX_HTTP_PARSE_INVALID_REQUEST;
1454                    }
1455                    if (*u == '/') {
1456                        u++;
1457                        break;
1458                    }
1459                    u--;
1460                }
1461                break;
1462            case '%':
1463                quoted_state = state;
1464                state = sw_quoted;
1465                break;
1466            case '?':
1467                r->args_start = p;
1468                goto args;
1469            case '#':
1470                goto done;
1471            case '+':
1472                r->plus_in_uri = 1;
1473            default:
1474                state = sw_usual;
1475                *u++ = ch;
1476                break;
1477            }
1478
1479            ch = *p++;
1480            break;
1481
1482        case sw_quoted:
1483            r->quoted_uri = 1;
1484
1485            if (ch >= '0' && ch <= '9') {
1486                decoded = (u_char) (ch - '0');
1487                state = sw_quoted_second;
1488                ch = *p++;
1489                break;
1490            }
1491
1492            c = (u_char) (ch | 0x20);
1493            if (c >= 'a' && c <= 'f') {
1494                decoded = (u_char) (c - 'a' + 10);
1495                state = sw_quoted_second;
1496                ch = *p++;
1497                break;
1498            }
1499
1500            return NGX_HTTP_PARSE_INVALID_REQUEST;
1501
1502        case sw_quoted_second:
1503            if (ch >= '0' && ch <= '9') {
1504                ch = (u_char) ((decoded << 4) + ch - '0');
1505
1506                if (ch == '%' || ch == '#') {
1507                    state = sw_usual;
1508                    *u++ = ch;
1509                    ch = *p++;
1510                    break;
1511
1512                } else if (ch == '\0') {
1513                    return NGX_HTTP_PARSE_INVALID_REQUEST;
1514                }
```

```
                state = quoted_state;
                break;
            }

            c = (u_char) (ch | 0x20);
            if (c >= 'a' && c <= 'f') {
                ch = (u_char) ((decoded << 4) + c - 'a' + 10);

                if (ch == '?') {
                    state = sw_usual;
                    *u++ = ch;
                    ch = *p++;
                    break;

                } else if (ch == '+') {
                    r->plus_in_uri = 1;
                }

                state = quoted_state;
                break;
            }

            return NGX_HTTP_PARSE_INVALID_REQUEST;
        }
    }

done:

    r->uri.len = u - r->uri.data;

    if (r->uri_ext) {
        r->exten.len = u - r->uri_ext;
        r->exten.data = r->uri_ext;
    }

    r->uri_ext = NULL;

    return NGX_OK;

args:

    while (p < r->uri_end) {
        if (*p++ != '#') {
            continue;
        }

        r->args.len = p - 1 - r->args_start;
        r->args.data = r->args_start;
        r->args_start = NULL;

        break;
    }

    r->uri.len = u - r->uri.data;

    if (r->uri_ext) {
        r->exten.len = u - r->uri_ext;
        r->exten.data = r->uri_ext;
    }

    r->uri_ext = NULL;

    return NGX_OK;
}


ngx_int_t
ngx_http_parse_status_line(ngx_http_request_t *r, ngx_buf_t *b,
    ngx_http_status_t *status)
{
    u_char   ch;
    u_char  *p;
    enum {
        sw_start = 0,
        sw_H,
```

```
        sw_HT,
        sw_HTT,
        sw_HTTP,
        sw_first_major_digit,
        sw_major_digit,
        sw_first_minor_digit,
        sw_minor_digit,
        sw_status,
        sw_space_after_status,
        sw_status_text,
        sw_almost_done
    } state;

    state = r->state;

    for (p = b->pos; p < b->last; p++) {
        ch = *p;

        switch (state) {

        /* "HTTP/" */
        case sw_start:
            switch (ch) {
            case 'H':
                state = sw_H;
                break;
            default:
                return NGX_ERROR;
            }
            break;

        case sw_H:
            switch (ch) {
            case 'T':
                state = sw_HT;
                break;
            default:
                return NGX_ERROR;
            }
            break;

        case sw_HT:
            switch (ch) {
            case 'T':
                state = sw_HTT;
                break;
            default:
                return NGX_ERROR;
            }
            break;

        case sw_HTT:
            switch (ch) {
            case 'P':
                state = sw_HTTP;
                break;
            default:
                return NGX_ERROR;
            }
            break;

        case sw_HTTP:
            switch (ch) {
            case '/':
                state = sw_first_major_digit;
                break;
            default:
                return NGX_ERROR;
            }
            break;

        /* the first digit of major HTTP version */
        case sw_first_major_digit:
            if (ch < '1' || ch > '9') {
                return NGX_ERROR;
            }
```

```
1667
1668            r->http_major = ch - '0';
1669            state = sw_major_digit;
1670            break;
1671
1672        /* the major HTTP version or dot */
1673        case sw_major_digit:
1674            if (ch == '.') {
1675                state = sw_first_minor_digit;
1676                break;
1677            }
1678
1679            if (ch < '0' || ch > '9') {
1680                return NGX_ERROR;
1681            }
1682
1683            r->http_major = r->http_major * 10 + ch - '0';
1684            break;
1685
1686        /* the first digit of minor HTTP version */
1687        case sw_first_minor_digit:
1688            if (ch < '0' || ch > '9') {
1689                return NGX_ERROR;
1690            }
1691
1692            r->http_minor = ch - '0';
1693            state = sw_minor_digit;
1694            break;
1695
1696        /* the minor HTTP version or the end of the request line */
1697        case sw_minor_digit:
1698            if (ch == ' ') {
1699                state = sw_status;
1700                break;
1701            }
1702
1703            if (ch < '0' || ch > '9') {
1704                return NGX_ERROR;
1705            }
1706
1707            r->http_minor = r->http_minor * 10 + ch - '0';
1708            break;
1709
1710        /* HTTP status code */
1711        case sw_status:
1712            if (ch == ' ') {
1713                break;
1714            }
1715
1716            if (ch < '0' || ch > '9') {
1717                return NGX_ERROR;
1718            }
1719
1720            status->code = status->code * 10 + ch - '0';
1721
1722            if (++status->count == 3) {
1723                state = sw_space_after_status;
1724                status->start = p - 2;
1725            }
1726
1727            break;
1728
1729        /* space or end of line */
1730        case sw_space_after_status:
1731            switch (ch) {
1732            case ' ':
1733                state = sw_status_text;
1734                break;
1735            case '.':                    /* IIS may send 403.1, 403.2, etc */
1736                state = sw_status_text;
1737                break;
1738            case CR:
1739                state = sw_almost_done;
1740                break;
1741            case LF:
1742                goto done;
```

```
            default:
                return NGX_ERROR;
            }
            break;

        /* any text until end of line */
        case sw_status_text:
            switch (ch) {
            case CR:
                state = sw_almost_done;

                break;
            case LF:
                goto done;
            }
            break;

        /* end of status line */
        case sw_almost_done:
            status->end = p - 1;
            switch (ch) {
            case LF:
                goto done;
            default:
                return NGX_ERROR;
            }
        }
    }

    b->pos = p;
    r->state = state;

    return NGX_AGAIN;

done:

    b->pos = p + 1;

    if (status->end == NULL) {
        status->end = p;
    }

    status->http_version = r->http_major * 1000 + r->http_minor;
    r->state = sw_start;

    return NGX_OK;
}


ngx_int_t
ngx_http_parse_unsafe_uri(ngx_http_request_t *r, ngx_str_t *uri,
    ngx_str_t *args, ngx_uint_t *flags)
{
    u_char      ch, *p, *src, *dst;
    size_t      len;
    ngx_uint_t  quoted;

    len = uri->len;
    p = uri->data;
    quoted = 0;

    if (len == 0 || p[0] == '?') {
        goto unsafe;
    }

    if (p[0] == '.' && len > 1 && p[1] == '.'
        && (len == 2 || ngx_path_separator(p[2])))
    {
        goto unsafe;
    }

    for ( /* void */ ; len; len--) {

        ch = *p++;

        if (ch == '%') {
```

```
            quoted = 1;
            continue;
        }

        if (usual[ch >> 5] & (1 << (ch & 0x1f))) {
            continue;
        }

        if (ch == '?') {
            args->len = len - 1;
            args->data = p;
            uri->len -= len;

            break;
        }

        if (ch == '\0') {
            goto unsafe;
        }

        if (ngx_path_separator(ch) && len > 2) {

            /* detect "/../" and "/.." */

            if (p[0] == '.' && p[1] == '.'
                && (len == 3 || ngx_path_separator(p[2])))
            {
                goto unsafe;
            }
        }
    }

    if (quoted) {
        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "escaped URI: \"%V\"", uri);

        src = uri->data;

        dst = ngx_pnalloc(r->pool, uri->len);
        if (dst == NULL) {
            return NGX_ERROR;
        }

        uri->data = dst;

        ngx_unescape_uri(&dst, &src, uri->len, 0);

        uri->len = dst - uri->data;

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "unescaped URI: \"%V\"", uri);

        len = uri->len;
        p = uri->data;

        if (p[0] == '.' && len > 1 && p[1] == '.'
            && (len == 2 || ngx_path_separator(p[2])))
        {
            goto unsafe;
        }

        for ( /* void */ ; len; len--) {

            ch = *p++;

            if (ch == '\0') {
                goto unsafe;
            }

            if (ngx_path_separator(ch) && len > 2) {

                /* detect "/../" and "/.." */

                if (p[0] == '.' && p[1] == '.'
                    && (len == 3 || ngx_path_separator(p[2])))
                {
```

```
                    goto unsafe;
                }
            }
        }
    }

    return NGX_OK;

unsafe:

    if (*flags & NGX_HTTP_LOG_UNSAFE) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "unsafe URI \"%V\" was detected", uri);
    }

    return NGX_ERROR;
}


ngx_int_t
ngx_http_parse_multi_header_lines(ngx_array_t *headers, ngx_str_t *name,
    ngx_str_t *value)
{
    ngx_uint_t         i;
    u_char            *start, *last, *end, ch;
    ngx_table_elt_t  **h;

    h = headers->elts;

    for (i = 0; i < headers->nelts; i++) {

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, headers->pool->log, 0,
                       "parse header: \"%V: %V\"", &h[i]->key, &h[i]->value);

        if (name->len > h[i]->value.len) {
            continue;
        }

        start = h[i]->value.data;
        end = h[i]->value.data + h[i]->value.len;

        while (start < end) {

            if (ngx_strncasecmp(start, name->data, name->len) != 0) {
                goto skip;
            }

            for (start += name->len; start < end && *start == ' '; start++) {
                /* void */
            }

            if (value == NULL) {
                if (start == end || *start == ',') {
                    return i;
                }

                goto skip;
            }

            if (start == end || *start++ != '=') {
                /* the invalid header value */
                goto skip;
            }

            while (start < end && *start == ' ') { start++; }

            for (last = start; last < end && *last != ';'; last++) {
                /* void */
            }

            value->len = last - start;
            value->data = start;

            return i;

        skip:
```

```
            while (start < end) {
                ch = *start++;
                if (ch == ';' || ch == ',') {
                    break;
                }
            }

            while (start < end && *start == ' ') { start++; }
        }
    }

    return NGX_DECLINED;
}


ngx_int_t
ngx_http_parse_set_cookie_lines(ngx_array_t *headers, ngx_str_t *name,
    ngx_str_t *value)
{
    ngx_uint_t         i;
    u_char            *start, *last, *end;
    ngx_table_elt_t  **h;

    h = headers->elts;

    for (i = 0; i < headers->nelts; i++) {

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, headers->pool->log, 0,
                       "parse header: \"%V: %V\"", &h[i]->key, &h[i]->value);

        if (name->len >= h[i]->value.len) {
            continue;
        }

        start = h[i]->value.data;
        end = h[i]->value.data + h[i]->value.len;

        if (ngx_strncasecmp(start, name->data, name->len) != 0) {
            continue;
        }

        for (start += name->len; start < end && *start == ' '; start++) {
            /* void */
        }

        if (start == end || *start++ != '=') {
            /* the invalid header value */
            continue;
        }

        while (start < end && *start == ' ') { start++; }

        for (last = start; last < end && *last != ';'; last++) {
            /* void */
        }

        value->len = last - start;
        value->data = start;

        return i;
    }

    return NGX_DECLINED;
}


ngx_int_t
ngx_http_arg(ngx_http_request_t *r, u_char *name, size_t len, ngx_str_t *value)
{
    u_char  *p, *last;

    if (r->args.len == 0) {
        return NGX_DECLINED;
    }

```

```
    p = r->args.data;
    last = p + r->args.len;

    for ( /* void */ ; p < last; p++) {

        /* we need '=' after name, so drop one char from last */

        p = ngx_strlcasestrn(p, last - 1, name, len - 1);

        if (p == NULL) {
            return NGX_DECLINED;
        }

        if ((p == r->args.data || *(p - 1) == '&') && *(p + len) == '=') {

            value->data = p + len + 1;

            p = ngx_strlchr(p, last, '&');

            if (p == NULL) {
                p = r->args.data + r->args.len;
            }

            value->len = p - value->data;

            return NGX_OK;
        }
    }

    return NGX_DECLINED;
}


void
ngx_http_split_args(ngx_http_request_t *r, ngx_str_t *uri, ngx_str_t *args)
{
    u_char  *p, *last;

    last = uri->data + uri->len;

    p = ngx_strlchr(uri->data, last, '?');

    if (p) {
        uri->len = p - uri->data;
        p++;
        args->len = last - p;
        args->data = p;

    } else {
        args->len = 0;
    }
}


ngx_int_t
ngx_http_parse_chunked(ngx_http_request_t *r, ngx_buf_t *b,
    ngx_http_chunked_t *ctx)
{
    u_char     *pos, ch, c;
    ngx_int_t   rc;
    enum {
        sw_chunk_start = 0,
        sw_chunk_size,
        sw_chunk_extension,
        sw_chunk_extension_almost_done,
        sw_chunk_data,
        sw_after_data,
        sw_after_data_almost_done,
        sw_last_chunk_extension,
        sw_last_chunk_extension_almost_done,
        sw_trailer,
        sw_trailer_almost_done,
        sw_trailer_header,
        sw_trailer_header_almost_done
    } state;

```

```
2123      state = ctx->state;
2124
2125      if (state == sw_chunk_data && ctx->size == 0) {
2126          state = sw_after_data;
2127      }
2128
2129      rc = NGX_AGAIN;
2130
2131      for (pos = b->pos; pos < b->last; pos++) {
2132
2133          ch = *pos;
2134
2135          ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2136                         "http chunked byte: %02Xd s:%d", ch, state);
2137
2138          switch (state) {
2139
2140          case sw_chunk_start:
2141              if (ch >= '0' && ch <= '9') {
2142                  state = sw_chunk_size;
2143                  ctx->size = ch - '0';
2144                  break;
2145              }
2146
2147              c = (u_char) (ch | 0x20);
2148
2149              if (c >= 'a' && c <= 'f') {
2150                  state = sw_chunk_size;
2151                  ctx->size = c - 'a' + 10;
2152                  break;
2153              }
2154
2155              goto invalid;
2156
2157          case sw_chunk_size:
2158              if (ch >= '0' && ch <= '9') {
2159                  ctx->size = ctx->size * 16 + (ch - '0');
2160                  break;
2161              }
2162
2163              c = (u_char) (ch | 0x20);
2164
2165              if (c >= 'a' && c <= 'f') {
2166                  ctx->size = ctx->size * 16 + (c - 'a' + 10);
2167                  break;
2168              }
2169
2170              if (ctx->size == 0) {
2171
2172                  switch (ch) {
2173                  case CR:
2174                      state = sw_last_chunk_extension_almost_done;
2175                      break;
2176                  case LF:
2177                      state = sw_trailer;
2178                      break;
2179                  case ';':
2180                  case ' ':
2181                  case '\t':
2182                      state = sw_last_chunk_extension;
2183                      break;
2184                  default:
2185                      goto invalid;
2186                  }
2187
2188                  break;
2189              }
2190
2191              switch (ch) {
2192              case CR:
2193                  state = sw_chunk_extension_almost_done;
2194                  break;
2195              case LF:
2196                  state = sw_chunk_data;
2197                  break;
2198              case ';':
```

```
            case ' ':
            case '\t':
                state = sw_chunk_extension;
                break;
            default:
                goto invalid;
            }

            break;

        case sw_chunk_extension:
            switch (ch) {
            case CR:
                state = sw_chunk_extension_almost_done;
                break;
            case LF:
                state = sw_chunk_data;
            }
            break;

        case sw_chunk_extension_almost_done:
            if (ch == LF) {
                state = sw_chunk_data;
                break;
            }
            goto invalid;

        case sw_chunk_data:
            rc = NGX_OK;
            goto data;

        case sw_after_data:
            switch (ch) {
            case CR:
                state = sw_after_data_almost_done;
                break;
            case LF:
                state = sw_chunk_start;
            }
            break;

        case sw_after_data_almost_done:
            if (ch == LF) {
                state = sw_chunk_start;
                break;
            }
            goto invalid;

        case sw_last_chunk_extension:
            switch (ch) {
            case CR:
                state = sw_last_chunk_extension_almost_done;
                break;
            case LF:
                state = sw_trailer;
            }
            break;

        case sw_last_chunk_extension_almost_done:
            if (ch == LF) {
                state = sw_trailer;
                break;
            }
            goto invalid;

        case sw_trailer:
            switch (ch) {
            case CR:
                state = sw_trailer_almost_done;
                break;
            case LF:
                goto done;
            default:
                state = sw_trailer_header;
            }
            break;
```

```
2275
2276        case sw_trailer_almost_done:
2277            if (ch == LF) {
2278                goto done;
2279            }
2280            goto invalid;
2281
2282        case sw_trailer_header:
2283            switch (ch) {
2284            case CR:
2285                state = sw_trailer_header_almost_done;
2286                break;
2287            case LF:
2288                state = sw_trailer;
2289            }
2290            break;
2291
2292        case sw_trailer_header_almost_done:
2293            if (ch == LF) {
2294                state = sw_trailer;
2295                break;
2296            }
2297            goto invalid;
2298
2299        }
2300    }
2301
2302data:
2303
2304    ctx->state = state;
2305    b->pos = pos;
2306
2307    switch (state) {
2308
2309    case sw_chunk_start:
2310        ctx->length = 3 /* "0" LF LF */;
2311        break;
2312    case sw_chunk_size:
2313        ctx->length = 1 /* LF */
2314                      + (ctx->size ? ctx->size + 4 /* LF "0" LF LF */
2315                                   : 1 /* LF */);
2316        break;
2317    case sw_chunk_extension:
2318    case sw_chunk_extension_almost_done:
2319        ctx->length = 1 /* LF */ + ctx->size + 4 /* LF "0" LF LF */;
2320        break;
2321    case sw_chunk_data:
2322        ctx->length = ctx->size + 4 /* LF "0" LF LF */;
2323        break;
2324    case sw_after_data:
2325    case sw_after_data_almost_done:
2326        ctx->length = 4 /* LF "0" LF LF */;
2327        break;
2328    case sw_last_chunk_extension:
2329    case sw_last_chunk_extension_almost_done:
2330        ctx->length = 2 /* LF LF */;
2331        break;
2332    case sw_trailer:
2333    case sw_trailer_almost_done:
2334        ctx->length = 1 /* LF */;
2335        break;
2336    case sw_trailer_header:
2337    case sw_trailer_header_almost_done:
2338        ctx->length = 2 /* LF LF */;
2339        break;
2340
2341    }
2342
2343    if (ctx->size < 0 || ctx->length < 0) {
2344        goto invalid;
2345    }
2346
2347    return rc;
2348
2349done:
2350
```

```
2351     ctx->state = 0;
2352     b->pos = pos + 1;
2353 
2354     return NGX_DONE;
2355 
2356 invalid:
2357 
2358     return NGX_ERROR;
2359 }
```

One Level Up Top Level

src/core/ngx_md5.c - nginx-1.7.10

Functions defined

- ngx_md5_body
- ngx_md5_final
- ngx_md5_init
- ngx_md5_update

Macros defined

- F
- G
- GET
- GET
- H
- I
- SET
- SET
- STEP

Source code

```
1
2  /*
3   * An internal implementation, based on Alexander Peslyak's
4   * public domain implementation:
5   * http://openwall.info/wiki/people/solar/software/public-domain-source-code/md5
6   * It is not expected to be optimal and is used only
7   * if no MD5 implementation was found in system.
8   */
9
10
11  #include <ngx_config.h>
12  #include <ngx_core.h>
13  #include <ngx_md5.h>
14
15
16  #if !(NGX_HAVE_MD5)
17
18  static const u_char *ngx_md5_body(ngx_md5_t *ctx, const u_char *data,
19      size_t size);
20
21
22  void
23  ngx_md5_init(ngx_md5_t *ctx)
24  {
25      ctx->a = 0x67452301;
26      ctx->b = 0xefcdab89;
27      ctx->c = 0x98badcfe;
28      ctx->d = 0x10325476;
29
30      ctx->bytes = 0;
31  }
32
```

```
33
34  void
35  ngx_md5_update(ngx_md5_t *ctx, const void *data, size_t size)
36  {
37      size_t  used, free;
38
39      used = (size_t) (ctx->bytes & 0x3f);
40      ctx->bytes += size;
41
42      if (used) {
43          free = 64 - used;
44
45          if (size < free) {
46              ngx_memcpy(&ctx->buffer[used], data, size);
47              return;
48          }
49
50          ngx_memcpy(&ctx->buffer[used], data, free);
51          data = (u_char *) data + free;
52          size -= free;
53          (void) ngx_md5_body(ctx, ctx->buffer, 64);
54      }
55
56      if (size >= 64) {
57          data = ngx_md5_body(ctx, data, size & ~(size_t) 0x3f);
58          size &= 0x3f;
59      }
60
61      ngx_memcpy(ctx->buffer, data, size);
62  }
63
64
65  void
66  ngx_md5_final(u_char result[16], ngx_md5_t *ctx)
67  {
68      size_t  used, free;
69
70      used = (size_t) (ctx->bytes & 0x3f);
71
72      ctx->buffer[used++] = 0x80;
73
74      free = 64 - used;
75
76      if (free < 8) {
77          ngx_memzero(&ctx->buffer[used], free);
78          (void) ngx_md5_body(ctx, ctx->buffer, 64);
79          used = 0;
80          free = 64;
81      }
82
83      ngx_memzero(&ctx->buffer[used], free - 8);
84
85      ctx->bytes <<= 3;
86      ctx->buffer[56] = (u_char) ctx->bytes;
87      ctx->buffer[57] = (u_char) (ctx->bytes >> 8);
88      ctx->buffer[58] = (u_char) (ctx->bytes >> 16);
89      ctx->buffer[59] = (u_char) (ctx->bytes >> 24);
90      ctx->buffer[60] = (u_char) (ctx->bytes >> 32);
91      ctx->buffer[61] = (u_char) (ctx->bytes >> 40);
92      ctx->buffer[62] = (u_char) (ctx->bytes >> 48);
93      ctx->buffer[63] = (u_char) (ctx->bytes >> 56);
94
95      (void) ngx_md5_body(ctx, ctx->buffer, 64);
96
97      result[0] = (u_char) ctx->a;
98      result[1] = (u_char) (ctx->a >> 8);
99      result[2] = (u_char) (ctx->a >> 16);
100      result[3] = (u_char) (ctx->a >> 24);
101      result[4] = (u_char) ctx->b;
102      result[5] = (u_char) (ctx->b >> 8);
103      result[6] = (u_char) (ctx->b >> 16);
104      result[7] = (u_char) (ctx->b >> 24);
105      result[8] = (u_char) ctx->c;
106      result[9] = (u_char) (ctx->c >> 8);
107      result[10] = (u_char) (ctx->c >> 16);
108      result[11] = (u_char) (ctx->c >> 24);
```

```
    result[12] = (u_char) ctx->d;
    result[13] = (u_char) (ctx->d >> 8);
    result[14] = (u_char) (ctx->d >> 16);
    result[15] = (u_char) (ctx->d >> 24);

    ngx_memzero(ctx, sizeof(*ctx));
}


/*
* The basic MD5 functions.
*
* F and G are optimized compared to their RFC 1321 definitions for
* architectures that lack an AND-NOT instruction, just like in
* Colin Plumb's implementation.
*/

#define F(x, y, z)  ((z) ^ ((x) & ((y) ^ (z))))
#define G(x, y, z)  ((y) ^ ((z) & ((x) ^ (y))))
#define H(x, y, z)  ((x) ^ (y) ^ (z))
#define I(x, y, z)  ((y) ^ ((x) | ~(z)))

/*
* The MD5 transformation for all four rounds.
*/

#define STEP(f, a, b, c, d, x, t, s)                                         \
    (a) += f((b), (c), (d)) + (x) + (t);                                     \
    (a) = (((a) << (s)) | (((a) & 0xffffffff) >> (32 - (s))));               \
    (a) += (b)

/*
* SET() reads 4 input bytes in little-endian byte order and stores them
* in a properly aligned word in host byte order.
*
* The check for little-endian architectures that tolerate unaligned
* memory accesses is just an optimization.  Nothing will break if it
* does not work.
*/

#if (NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED)

#define SET(n)      (*(uint32_t *) &p[n * 4])
#define GET(n)      (*(uint32_t *) &p[n * 4])

#else

#define SET(n)                                                               \
    (block[n] =                                                              \
    (uint32_t) p[n * 4] |                                                    \
    ((uint32_t) p[n * 4 + 1] << 8) |                                         \
    ((uint32_t) p[n * 4 + 2] << 16) |                                        \
    ((uint32_t) p[n * 4 + 3] << 24))

#define GET(n)      block[n]

#endif


/*
* This processes one or more 64-byte data blocks, but does not update
* the bit counters.  There are no alignment requirements.
*/

static const u_char *
ngx_md5_body(ngx_md5_t *ctx, const u_char *data, size_t size)
{
    uint32_t       a, b, c, d;
    uint32_t       saved_a, saved_b, saved_c, saved_d;
    const u_char  *p;
#if !(NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED)
    uint32_t       block[16];
#endif

    p = data;

```

```
    a = ctx->a;
    b = ctx->b;
    c = ctx->c;
    d = ctx->d;

    do {
        saved_a = a;
        saved_b = b;
        saved_c = c;
        saved_d = d;

        /* Round 1 */

        STEP(F, a, b, c, d, SET(0),  0xd76aa478, 7);
        STEP(F, d, a, b, c, SET(1),  0xe8c7b756, 12);
        STEP(F, c, d, a, b, SET(2),  0x242070db, 17);
        STEP(F, b, c, d, a, SET(3),  0xc1bdceee, 22);
        STEP(F, a, b, c, d, SET(4),  0xf57c0faf, 7);
        STEP(F, d, a, b, c, SET(5),  0x4787c62a, 12);
        STEP(F, c, d, a, b, SET(6),  0xa8304613, 17);
        STEP(F, b, c, d, a, SET(7),  0xfd469501, 22);
        STEP(F, a, b, c, d, SET(8),  0x698098d8, 7);
        STEP(F, d, a, b, c, SET(9),  0x8b44f7af, 12);
        STEP(F, c, d, a, b, SET(10), 0xffff5bb1, 17);
        STEP(F, b, c, d, a, SET(11), 0x895cd7be, 22);
        STEP(F, a, b, c, d, SET(12), 0x6b901122, 7);
        STEP(F, d, a, b, c, SET(13), 0xfd987193, 12);
        STEP(F, c, d, a, b, SET(14), 0xa679438e, 17);
        STEP(F, b, c, d, a, SET(15), 0x49b40821, 22);

        /* Round 2 */

        STEP(G, a, b, c, d, GET(1),  0xf61e2562, 5);
        STEP(G, d, a, b, c, GET(6),  0xc040b340, 9);
        STEP(G, c, d, a, b, GET(11), 0x265e5a51, 14);
        STEP(G, b, c, d, a, GET(0),  0xe9b6c7aa, 20);
        STEP(G, a, b, c, d, GET(5),  0xd62f105d, 5);
        STEP(G, d, a, b, c, GET(10), 0x02441453, 9);
        STEP(G, c, d, a, b, GET(15), 0xd8a1e681, 14);
        STEP(G, b, c, d, a, GET(4),  0xe7d3fbc8, 20);
        STEP(G, a, b, c, d, GET(9),  0x21e1cde6, 5);
        STEP(G, d, a, b, c, GET(14), 0xc33707d6, 9);
        STEP(G, c, d, a, b, GET(3),  0xf4d50d87, 14);
        STEP(G, b, c, d, a, GET(8),  0x455a14ed, 20);
        STEP(G, a, b, c, d, GET(13), 0xa9e3e905, 5);
        STEP(G, d, a, b, c, GET(2),  0xfcefa3f8, 9);
        STEP(G, c, d, a, b, GET(7),  0x676f02d9, 14);
        STEP(G, b, c, d, a, GET(12), 0x8d2a4c8a, 20);

        /* Round 3 */

        STEP(H, a, b, c, d, GET(5),  0xfffa3942, 4);
        STEP(H, d, a, b, c, GET(8),  0x8771f681, 11);
        STEP(H, c, d, a, b, GET(11), 0x6d9d6122, 16);
        STEP(H, b, c, d, a, GET(14), 0xfde5380c, 23);
        STEP(H, a, b, c, d, GET(1),  0xa4beea44, 4);
        STEP(H, d, a, b, c, GET(4),  0x4bdecfa9, 11);
        STEP(H, c, d, a, b, GET(7),  0xf6bb4b60, 16);
        STEP(H, b, c, d, a, GET(10), 0xbebfbc70, 23);
        STEP(H, a, b, c, d, GET(13), 0x289b7ec6, 4);
        STEP(H, d, a, b, c, GET(0),  0xeaa127fa, 11);
        STEP(H, c, d, a, b, GET(3),  0xd4ef3085, 16);
        STEP(H, b, c, d, a, GET(6),  0x04881d05, 23);
        STEP(H, a, b, c, d, GET(9),  0xd9d4d039, 4);
        STEP(H, d, a, b, c, GET(12), 0xe6db99e5, 11);
        STEP(H, c, d, a, b, GET(15), 0x1fa27cf8, 16);
        STEP(H, b, c, d, a, GET(2),  0xc4ac5665, 23);

        /* Round 4 */

        STEP(I, a, b, c, d, GET(0),  0xf4292244, 6);
        STEP(I, d, a, b, c, GET(7),  0x432aff97, 10);
        STEP(I, c, d, a, b, GET(14), 0xab9423a7, 15);
        STEP(I, b, c, d, a, GET(5),  0xfc93a039, 21);
        STEP(I, a, b, c, d, GET(12), 0x655b59c3, 6);
        STEP(I, d, a, b, c, GET(3),  0x8f0ccc92, 10);
```

```
        STEP(I, c, d, a, b, GET(10), 0xffeff47d, 15);
        STEP(I, b, c, d, a, GET(1),  0x85845dd1, 21);
        STEP(I, a, b, c, d, GET(8),  0x6fa87e4f, 6);
        STEP(I, d, a, b, c, GET(15), 0xfe2ce6e0, 10);
        STEP(I, c, d, a, b, GET(6),  0xa3014314, 15);
        STEP(I, b, c, d, a, GET(13), 0x4e0811a1, 21);
        STEP(I, a, b, c, d, GET(4),  0xf7537e82, 6);
        STEP(I, d, a, b, c, GET(11), 0xbd3af235, 10);
        STEP(I, c, d, a, b, GET(2),  0x2ad7d2bb, 15);
        STEP(I, b, c, d, a, GET(9),  0xeb86d391, 21);

        a += saved_a;
        b += saved_b;
        c += saved_c;
        d += saved_d;

        p += 64;

    } while (size -= 64);

    ctx->a = a;
    ctx->b = b;
    ctx->c = c;
    ctx->d = d;

    return p;
}

#endif
```

One Level Up Top Level

src/core/ngx_md5.h - nginx-1.7.10

Data types defined

- ngx_md5_t
- ngx_md5_t

Macros defined

- _NGX_MD5_H_INCLUDED_
- ngx_md5_final
- ngx_md5_final
- ngx_md5_init
- ngx_md5_init
- ngx_md5_update
- ngx_md5_update

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_MD5_H_INCLUDED_
9  #define _NGX_MD5_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #if (NGX_HAVE_MD5)
17
18  #if (NGX_HAVE_OPENSSL_MD5_H)
19  #include <openssl/md5.h>
20  #else
21  #include <md5.h>
22  #endif
23
24
25  typedef MD5_CTX  ngx_md5_t;
26
27
28  #if (NGX_OPENSSL_MD5)
29
30  #define ngx_md5_init    MD5_Init
31  #define ngx_md5_update  MD5_Update
32  #define ngx_md5_final   MD5_Final
33
34  #else
35
36  #define ngx_md5_init    MD5Init
37  #define ngx_md5_update  MD5Update
38  #define ngx_md5_final   MD5Final
39
40  #endif
41
```

```
42
43  #else /* !NGX_HAVE_MD5 */
44
45
46  typedef struct {
47      uint64_t  bytes;
48      uint32_t  a, b, c, d;
49      u_char    buffer[64];
50  } ngx_md5_t;
51
52
53  void ngx_md5_init(ngx_md5_t *ctx);
54  void ngx_md5_update(ngx_md5_t *ctx, const void *data, size_t size);
55  void ngx_md5_final(u_char result[16], ngx_md5_t *ctx);
56
57
58  #endif
59
60  #endif /* _NGX_MD5_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_spdy.c - nginx-1.7.10

Global variables defined

- ngx_http_spdy_dict
- ngx_http_spdy_request_headers

Data types defined

- ngx_http_spdy_request_header_t

Functions defined

- ngx_http_spdy_adjust_windows
- ngx_http_spdy_alloc_large_header_buffer
- ngx_http_spdy_close_stream
- ngx_http_spdy_close_stream_handler
- ngx_http_spdy_construct_request_line
- ngx_http_spdy_create_stream
- ngx_http_spdy_ctl_frame_handler
- ngx_http_spdy_finalize_connection
- ngx_http_spdy_get_ctl_frame
- ngx_http_spdy_get_stream_by_id
- ngx_http_spdy_handle_connection
- ngx_http_spdy_handle_connection_handler
- ngx_http_spdy_handle_request_header
- ngx_http_spdy_init
- ngx_http_spdy_init_request_body
- ngx_http_spdy_keepalive_handler
- ngx_http_spdy_parse_header
- ngx_http_spdy_parse_host
- ngx_http_spdy_parse_method
- ngx_http_spdy_parse_path
- ngx_http_spdy_parse_scheme
- ngx_http_spdy_parse_version
- ngx_http_spdy_pool_cleanup
- ngx_http_spdy_proxy_protocol

- ngx_http_spdy_read_handler
- ngx_http_spdy_read_request_body
- ngx_http_spdy_request_headers_init
- ngx_http_spdy_run_request
- ngx_http_spdy_send_output_queue
- ngx_http_spdy_send_rst_stream
- ngx_http_spdy_send_settings
- ngx_http_spdy_send_window_update
- ngx_http_spdy_settings_frame_handler
- ngx_http_spdy_state_complete
- ngx_http_spdy_state_data
- ngx_http_spdy_state_head
- ngx_http_spdy_state_headers
- ngx_http_spdy_state_headers_error
- ngx_http_spdy_state_headers_skip
- ngx_http_spdy_state_inflate_error
- ngx_http_spdy_state_internal_error
- ngx_http_spdy_state_ping
- ngx_http_spdy_state_protocol_error
- ngx_http_spdy_state_read_data
- ngx_http_spdy_state_rst_stream
- ngx_http_spdy_state_save
- ngx_http_spdy_state_settings
- ngx_http_spdy_state_skip
- ngx_http_spdy_state_syn_stream
- ngx_http_spdy_state_window_update
- ngx_http_spdy_terminate_stream
- ngx_http_spdy_write_handler
- ngx_http_spdy_zalloc
- ngx_http_spdy_zfree

Macros defined

- NGX_SPDY_CANCEL
- NGX_SPDY_CONNECTION_WINDOW

- NGX_SPDY_CTL_FRAME_BUFFER_SIZE
- NGX_SPDY_FLOW_CONTROL_ERROR
- NGX_SPDY_FRAME_TOO_LARGE
- NGX_SPDY_INIT_STREAM_WINDOW
- NGX_SPDY_INTERNAL_ERROR
- NGX_SPDY_INVALID_STREAM
- NGX_SPDY_MAX_WINDOW
- NGX_SPDY_PROTOCOL_ERROR
- NGX_SPDY_REFUSED_STREAM
- NGX_SPDY_REQUEST_HEADERS
- NGX_SPDY_SETTINGS_FLAG_PERSIST
- NGX_SPDY_SETTINGS_FLAG_PERSISTED
- NGX_SPDY_SETTINGS_INIT_WINDOW
- NGX_SPDY_SETTINGS_MAX_STREAMS
- NGX_SPDY_SKIP_HEADERS_BUFFER_SIZE
- NGX_SPDY_STREAM_ALREADY_CLOSED
- NGX_SPDY_STREAM_IN_USE
- NGX_SPDY_STREAM_WINDOW
- NGX_SPDY_UNSUPPORTED_VERSION
- ngx_http_spdy_stream_index
- ngx_http_spdy_streams_index_size
- ngx_spdy_ctl_frame_check
- ngx_spdy_ctl_frame_type
- ngx_spdy_data_frame_check
- ngx_spdy_frame_flags
- ngx_spdy_frame_id
- ngx_spdy_frame_length
- ngx_spdy_frame_parse_delta
- ngx_spdy_frame_parse_sid
- ngx_spdy_frame_parse_uint16
- ngx_spdy_frame_parse_uint16
- ngx_spdy_frame_parse_uint32
- ngx_spdy_frame_parse_uint32

- ngx_str5cmp
- ngx_str5cmp

Source code

```
1
2  /*
3   * Copyright (C) Nginx, Inc.
4   * Copyright (C) Valentin V. Bartenev
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <ngx_http_spdy_module.h>
12
13  #include <zlib.h>
14
15
16  #if (NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED)
17
18  #define ngx_str5cmp(m, c0, c1, c2, c3, c4)                                   \
19      *(uint32_t *) m == (c3 << 24 | c2 << 16 | c1 << 8 | c0)                  \
20          && m[4] == c4
21
22  #else
23
24  #define ngx_str5cmp(m, c0, c1, c2, c3, c4)                                   \
25      m[0] == c0 && m[1] == c1 && m[2] == c2 && m[3] == c3 && m[4] == c4
26
27  #endif
28
29
30  #if (NGX_HAVE_NONALIGNED)
31
32  #define ngx_spdy_frame_parse_uint16(p)  ntohs(*(uint16_t *) (p))
33  #define ngx_spdy_frame_parse_uint32(p)  ntohl(*(uint32_t *) (p))
34
35  #else
36
37  #define ngx_spdy_frame_parse_uint16(p) ((p)[0] << 8 | (p)[1])
38  #define ngx_spdy_frame_parse_uint32(p)                                       \
39      ((p)[0] << 24 | (p)[1] << 16 | (p)[2] << 8 | (p)[3])
40
41  #endif
42
43  #define ngx_spdy_frame_parse_sid(p)                                          \
44      (ngx_spdy_frame_parse_uint32(p) & 0x7fffffff)
45  #define ngx_spdy_frame_parse_delta(p)                                        \
46      (ngx_spdy_frame_parse_uint32(p) & 0x7fffffff)
47
48
49  #define ngx_spdy_ctl_frame_check(h)                                          \
50      (((h) & 0xffff0000) == ngx_spdy_ctl_frame_head(0))
51  #define ngx_spdy_data_frame_check(h)                                         \
52      (!((h) & (uint32_t) NGX_SPDY_CTL_BIT << 31))
53
54  #define ngx_spdy_ctl_frame_type(h)   ((h) & 0x0000ffff)
55  #define ngx_spdy_frame_flags(p)      ((p) >> 24)
56  #define ngx_spdy_frame_length(p)     ((p) & 0x00ffffff)
57  #define ngx_spdy_frame_id(p)         ((p) & 0x00ffffff)
58
59
60  #define NGX_SPDY_SKIP_HEADERS_BUFFER_SIZE  4096
61  #define NGX_SPDY_CTL_FRAME_BUFFER_SIZE     16
62
63  #define NGX_SPDY_PROTOCOL_ERROR            1
64  #define NGX_SPDY_INVALID_STREAM            2
65  #define NGX_SPDY_REFUSED_STREAM            3
66  #define NGX_SPDY_UNSUPPORTED_VERSION       4
67  #define NGX_SPDY_CANCEL                    5
68  #define NGX_SPDY_INTERNAL_ERROR            6
```

```
#define NGX_SPDY_FLOW_CONTROL_ERROR        7
#define NGX_SPDY_STREAM_IN_USE             8
#define NGX_SPDY_STREAM_ALREADY_CLOSED     9
/* deprecated                              10 */
#define NGX_SPDY_FRAME_TOO_LARGE           11

#define NGX_SPDY_SETTINGS_MAX_STREAMS      4
#define NGX_SPDY_SETTINGS_INIT_WINDOW      7

#define NGX_SPDY_SETTINGS_FLAG_PERSIST     0x01
#define NGX_SPDY_SETTINGS_FLAG_PERSISTED   0x02

#define NGX_SPDY_MAX_WINDOW                NGX_MAX_INT32_VALUE
#define NGX_SPDY_CONNECTION_WINDOW         65536
#define NGX_SPDY_INIT_STREAM_WINDOW        65536
#define NGX_SPDY_STREAM_WINDOW             NGX_SPDY_MAX_WINDOW

typedef struct {
    ngx_uint_t    hash;
    u_char        len;
    u_char        header[7];
    ngx_int_t   (*handler)(ngx_http_request_t *r);
} ngx_http_spdy_request_header_t;


static void ngx_http_spdy_read_handler(ngx_event_t *rev);
static void ngx_http_spdy_write_handler(ngx_event_t *wev);
static void ngx_http_spdy_handle_connection(ngx_http_spdy_connection_t *sc);

static u_char *ngx_http_spdy_proxy_protocol(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_head(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_syn_stream(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_headers(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_headers_skip(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_headers_error(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_window_update(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_data(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_read_data(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_rst_stream(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_ping(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_skip(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_settings(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_complete(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
static u_char *ngx_http_spdy_state_save(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end, ngx_http_spdy_handler_pt handler);

static u_char *ngx_http_spdy_state_inflate_error(
    ngx_http_spdy_connection_t *sc, int rc);
static u_char *ngx_http_spdy_state_protocol_error(
    ngx_http_spdy_connection_t *sc);
static u_char *ngx_http_spdy_state_internal_error(
    ngx_http_spdy_connection_t *sc);

static ngx_int_t ngx_http_spdy_send_window_update(
    ngx_http_spdy_connection_t *sc, ngx_uint_t sid, ngx_uint_t delta);
static ngx_int_t ngx_http_spdy_send_rst_stream(ngx_http_spdy_connection_t *sc,
    ngx_uint_t sid, ngx_uint_t status, ngx_uint_t priority);
static ngx_int_t ngx_http_spdy_send_settings(ngx_http_spdy_connection_t *sc);
static ngx_int_t ngx_http_spdy_settings_frame_handler(
    ngx_http_spdy_connection_t *sc, ngx_http_spdy_out_frame_t *frame);
static ngx_http_spdy_out_frame_t *ngx_http_spdy_get_ctl_frame(
    ngx_http_spdy_connection_t *sc, size_t size, ngx_uint_t priority);
```

```
static ngx_int_t ngx_http_spdy_ctl_frame_handler(
    ngx_http_spdy_connection_t *sc, ngx_http_spdy_out_frame_t *frame);

static ngx_http_spdy_stream_t *ngx_http_spdy_create_stream(
    ngx_http_spdy_connection_t *sc, ngx_uint_t id, ngx_uint_t priority);
static ngx_http_spdy_stream_t *ngx_http_spdy_get_stream_by_id(
    ngx_http_spdy_connection_t *sc, ngx_uint_t sid);
#define ngx_http_spdy_streams_index_size(sscf)  (sscf->streams_index_mask + 1)
#define ngx_http_spdy_stream_index(sscf, sid)                                 \
    ((sid >> 1) & sscf->streams_index_mask)

static ngx_int_t ngx_http_spdy_parse_header(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_alloc_large_header_buffer(ngx_http_request_t *r);

static ngx_int_t ngx_http_spdy_handle_request_header(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_parse_method(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_parse_scheme(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_parse_host(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_parse_path(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_parse_version(ngx_http_request_t *r);

static ngx_int_t ngx_http_spdy_construct_request_line(ngx_http_request_t *r);
static void ngx_http_spdy_run_request(ngx_http_request_t *r);
static ngx_int_t ngx_http_spdy_init_request_body(ngx_http_request_t *r);

static ngx_int_t ngx_http_spdy_terminate_stream(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_stream_t *stream, ngx_uint_t status);

static void ngx_http_spdy_close_stream_handler(ngx_event_t *ev);

static void ngx_http_spdy_handle_connection_handler(ngx_event_t *rev);
static void ngx_http_spdy_keepalive_handler(ngx_event_t *rev);
static void ngx_http_spdy_finalize_connection(ngx_http_spdy_connection_t *sc,
    ngx_int_t rc);

static ngx_int_t ngx_http_spdy_adjust_windows(ngx_http_spdy_connection_t *sc,
    ssize_t delta);

static void ngx_http_spdy_pool_cleanup(void *data);

static void *ngx_http_spdy_zalloc(void *opaque, u_int items, u_int size);
static void ngx_http_spdy_zfree(void *opaque, void *address);


static const u_char ngx_http_spdy_dict[] = {
    0x00, 0x00, 0x00, 0x07, 0x6f, 0x70, 0x74, 0x69,   /* - - - - o p t i */
    0x6f, 0x6e, 0x73, 0x00, 0x00, 0x00, 0x04, 0x68,   /* o n s - - - - h */
    0x65, 0x61, 0x64, 0x00, 0x00, 0x00, 0x04, 0x70,   /* e a d - - - - p */
    0x6f, 0x73, 0x74, 0x00, 0x00, 0x00, 0x03, 0x70,   /* o s t - - - - p */
    0x75, 0x74, 0x00, 0x00, 0x00, 0x06, 0x64, 0x65,   /* u t - - - - d e */
    0x6c, 0x65, 0x74, 0x65, 0x00, 0x00, 0x00, 0x05,   /* l e t e - - - - */
    0x74, 0x72, 0x61, 0x63, 0x65, 0x00, 0x00, 0x00,   /* t r a c e - - - */
    0x06, 0x61, 0x63, 0x63, 0x65, 0x70, 0x74, 0x00,   /* - a c c e p t - */
    0x00, 0x00, 0x0e, 0x61, 0x63, 0x63, 0x65, 0x70,   /* - - - a c c e p */
    0x74, 0x2d, 0x63, 0x68, 0x61, 0x72, 0x73, 0x65,   /* t - c h a r s e */
    0x74, 0x00, 0x00, 0x00, 0x0f, 0x61, 0x63, 0x63,   /* t - - - - a c c */
    0x65, 0x70, 0x74, 0x2d, 0x65, 0x6e, 0x63, 0x6f,   /* e p t - e n c o */
    0x64, 0x69, 0x6e, 0x67, 0x00, 0x00, 0x00, 0x0f,   /* d i n g - - - - */
    0x61, 0x63, 0x63, 0x65, 0x70, 0x74, 0x2d, 0x6c,   /* a c c e p t - l */
    0x61, 0x6e, 0x67, 0x75, 0x61, 0x67, 0x65, 0x00,   /* a n g u a g e - */
    0x00, 0x00, 0x0d, 0x61, 0x63, 0x63, 0x65, 0x70,   /* - - - a c c e p */
    0x74, 0x2d, 0x72, 0x61, 0x6e, 0x67, 0x65, 0x73,   /* t - r a n g e s */
    0x00, 0x00, 0x00, 0x03, 0x61, 0x67, 0x65, 0x00,   /* - - - - a g e - */
    0x00, 0x00, 0x05, 0x61, 0x6c, 0x6c, 0x6f, 0x77,   /* - - - a l l o w */
    0x00, 0x00, 0x00, 0x0d, 0x61, 0x75, 0x74, 0x68,   /* - - - - a u t h */
    0x6f, 0x72, 0x69, 0x7a, 0x61, 0x74, 0x69, 0x6f,   /* o r i z a t i o */
    0x6e, 0x00, 0x00, 0x00, 0x0d, 0x63, 0x61, 0x63,   /* n - - - - c a c */
    0x68, 0x65, 0x2d, 0x63, 0x6f, 0x6e, 0x74, 0x72,   /* h e - c o n t r */
    0x6f, 0x6c, 0x00, 0x00, 0x00, 0x0a, 0x63, 0x6f,   /* o l - - - - c o */
    0x6e, 0x6e, 0x65, 0x63, 0x74, 0x69, 0x6f, 0x6e,   /* n n e c t i o n */
    0x00, 0x00, 0x00, 0x0c, 0x63, 0x6f, 0x6e, 0x74,   /* - - - - c o n t */
    0x65, 0x6e, 0x74, 0x2d, 0x62, 0x61, 0x73, 0x65,   /* e n t - b a s e */
    0x00, 0x00, 0x00, 0x10, 0x63, 0x6f, 0x6e, 0x74,   /* - - - - c o n t */
    0x65, 0x6e, 0x74, 0x2d, 0x65, 0x6e, 0x63, 0x6f,   /* e n t - e n c o */
    0x64, 0x69, 0x6e, 0x67, 0x00, 0x00, 0x00, 0x10,   /* d i n g - - - - */
    0x63, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74, 0x2d,   /* c o n t e n t - */
```

```
    0x6c, 0x61, 0x6e, 0x67, 0x75, 0x61, 0x67, 0x65,   /* l a n g u a g e */
    0x00, 0x00, 0x00, 0x0e, 0x63, 0x6f, 0x6e, 0x74,   /* - - - - c o n t */
    0x65, 0x6e, 0x74, 0x2d, 0x6c, 0x65, 0x6e, 0x67,   /* e n t - l e n g */
    0x74, 0x68, 0x00, 0x00, 0x00, 0x10, 0x63, 0x6f,   /* t h - - - - c o */
    0x6e, 0x74, 0x65, 0x6e, 0x74, 0x2d, 0x6c, 0x6f,   /* n t e n t - l o */
    0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x00, 0x00,   /* c a t i o n - - */
    0x00, 0x0b, 0x63, 0x6f, 0x6e, 0x74, 0x65, 0x6e,   /* - - c o n t e n */
    0x74, 0x2d, 0x6d, 0x64, 0x35, 0x00, 0x00, 0x00,   /* t - m d 5 - - - */
    0x0d, 0x63, 0x6f, 0x6e, 0x74, 0x65, 0x6e, 0x74,   /* - c o n t e n t */
    0x2d, 0x72, 0x61, 0x6e, 0x67, 0x65, 0x00, 0x00,   /* - r a n g e - - */
    0x00, 0x0c, 0x63, 0x6f, 0x6e, 0x74, 0x65, 0x6e,   /* - - c o n t e n */
    0x74, 0x2d, 0x74, 0x79, 0x70, 0x65, 0x00, 0x00,   /* t - t y p e - - */
    0x00, 0x04, 0x64, 0x61, 0x74, 0x65, 0x00, 0x00,   /* - - d a t e - - */
    0x00, 0x04, 0x65, 0x74, 0x61, 0x67, 0x00, 0x00,   /* - - e t a g - - */
    0x00, 0x06, 0x65, 0x78, 0x70, 0x65, 0x63, 0x74,   /* - - e x p e c t */
    0x00, 0x00, 0x00, 0x07, 0x65, 0x78, 0x70, 0x69,   /* - - - - e x p i */
    0x72, 0x65, 0x73, 0x00, 0x00, 0x00, 0x04, 0x66,   /* r e s - - - - f */
    0x72, 0x6f, 0x6d, 0x00, 0x00, 0x00, 0x04, 0x68,   /* r o m - - - - h */
    0x6f, 0x73, 0x74, 0x00, 0x00, 0x00, 0x08, 0x69,   /* o s t - - - - i */
    0x66, 0x2d, 0x6d, 0x61, 0x74, 0x63, 0x68, 0x00,   /* f - m a t c h - */
    0x00, 0x00, 0x11, 0x69, 0x66, 0x2d, 0x6d, 0x6f,   /* - - - i f - m o */
    0x64, 0x69, 0x66, 0x69, 0x65, 0x64, 0x2d, 0x73,   /* d i f i e d - s */
    0x69, 0x6e, 0x63, 0x65, 0x00, 0x00, 0x00, 0x0d,   /* i n c e - - - - */
    0x69, 0x66, 0x2d, 0x6e, 0x6f, 0x6e, 0x65, 0x2d,   /* i f - n o n e - */
    0x6d, 0x61, 0x74, 0x63, 0x68, 0x00, 0x00, 0x00,   /* m a t c h - - - */
    0x08, 0x69, 0x66, 0x2d, 0x72, 0x61, 0x6e, 0x67,   /* - i f - r a n g */
    0x65, 0x00, 0x00, 0x00, 0x13, 0x69, 0x66, 0x2d,   /* e - - - - i f - */
    0x75, 0x6e, 0x6d, 0x6f, 0x64, 0x69, 0x66, 0x69,   /* u n m o d i f i */
    0x65, 0x64, 0x2d, 0x73, 0x69, 0x6e, 0x63, 0x65,   /* e d - s i n c e */
    0x00, 0x00, 0x00, 0x0d, 0x6c, 0x61, 0x73, 0x74,   /* - - - - l a s t */
    0x2d, 0x6d, 0x6f, 0x64, 0x69, 0x66, 0x69, 0x65,   /* - m o d i f i e */
    0x64, 0x00, 0x00, 0x00, 0x08, 0x6c, 0x6f, 0x63,   /* d - - - - l o c */
    0x61, 0x74, 0x69, 0x6f, 0x6e, 0x00, 0x00, 0x00,   /* a t i o n - - - */
    0x0c, 0x6d, 0x61, 0x78, 0x2d, 0x66, 0x6f, 0x72,   /* - m a x - f o r */
    0x77, 0x61, 0x72, 0x64, 0x73, 0x00, 0x00, 0x00,   /* w a r d s - - - */
    0x06, 0x70, 0x72, 0x61, 0x67, 0x6d, 0x61, 0x00,   /* - p r a g m a - */
    0x00, 0x00, 0x12, 0x70, 0x72, 0x6f, 0x78, 0x79,   /* - - - p r o x y */
    0x2d, 0x61, 0x75, 0x74, 0x68, 0x65, 0x6e, 0x74,   /* - a u t h e n t */
    0x69, 0x63, 0x61, 0x74, 0x65, 0x00, 0x00, 0x00,   /* i c a t e - - - */
    0x13, 0x70, 0x72, 0x6f, 0x78, 0x79, 0x2d, 0x61,   /* - p r o x y - a */
    0x75, 0x74, 0x68, 0x6f, 0x72, 0x69, 0x7a, 0x61,   /* u t h o r i z a */
    0x74, 0x69, 0x6f, 0x6e, 0x00, 0x00, 0x00, 0x05,   /* t i o n - - - - */
    0x72, 0x61, 0x6e, 0x67, 0x65, 0x00, 0x00, 0x00,   /* r a n g e - - - */
    0x07, 0x72, 0x65, 0x66, 0x65, 0x72, 0x65, 0x72,   /* - r e f e r e r */
    0x00, 0x00, 0x00, 0x0b, 0x72, 0x65, 0x74, 0x72,   /* - - - - r e t r */
    0x79, 0x2d, 0x61, 0x66, 0x74, 0x65, 0x72, 0x00,   /* y - a f t e r - */
    0x00, 0x00, 0x06, 0x73, 0x65, 0x72, 0x76, 0x65,   /* - - - s e r v e */
    0x72, 0x00, 0x00, 0x00, 0x02, 0x74, 0x65, 0x00,   /* r - - - - t e - */
    0x00, 0x00, 0x07, 0x74, 0x72, 0x61, 0x69, 0x6c,   /* - - - t r a i l */
    0x65, 0x72, 0x00, 0x00, 0x00, 0x11, 0x74, 0x72,   /* e r - - - - t r */
    0x61, 0x6e, 0x73, 0x66, 0x65, 0x72, 0x2d, 0x65,   /* a n s f e r - e */
    0x6e, 0x63, 0x6f, 0x64, 0x69, 0x6e, 0x67, 0x00,   /* n c o d i n g - */
    0x00, 0x00, 0x07, 0x75, 0x70, 0x67, 0x72, 0x61,   /* - - - u p g r a */
    0x64, 0x65, 0x00, 0x00, 0x00, 0x0a, 0x75, 0x73,   /* d e - - - - u s */
    0x65, 0x72, 0x2d, 0x61, 0x67, 0x65, 0x6e, 0x74,   /* e r - a g e n t */
    0x00, 0x00, 0x00, 0x04, 0x76, 0x61, 0x72, 0x79,   /* - - - - v a r y */
    0x00, 0x00, 0x00, 0x03, 0x76, 0x69, 0x61, 0x00,   /* - - - - v i a - */
    0x00, 0x00, 0x07, 0x77, 0x61, 0x72, 0x6e, 0x69,   /* - - - w a r n i */
    0x6e, 0x67, 0x00, 0x00, 0x00, 0x10, 0x77, 0x77,   /* n g - - - - w w */
    0x77, 0x2d, 0x61, 0x75, 0x74, 0x68, 0x65, 0x6e,   /* w - a u t h e n */
    0x74, 0x69, 0x63, 0x61, 0x74, 0x65, 0x00, 0x00,   /* t i c a t e - - */
    0x00, 0x06, 0x6d, 0x65, 0x74, 0x68, 0x6f, 0x64,   /* - - m e t h o d */
    0x00, 0x00, 0x00, 0x03, 0x67, 0x65, 0x74, 0x00,   /* - - - - g e t - */
    0x00, 0x00, 0x06, 0x73, 0x74, 0x61, 0x74, 0x75,   /* - - - s t a t u */
    0x73, 0x00, 0x00, 0x00, 0x06, 0x32, 0x30, 0x30,   /* s - - - - 2 0 0 */
    0x20, 0x4f, 0x4b, 0x00, 0x00, 0x00, 0x07, 0x76,   /* - O K - - - - v */
    0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00, 0x00,   /* e r s i o n - - */
    0x00, 0x08, 0x48, 0x54, 0x54, 0x50, 0x2f, 0x31,   /* - - H T T P - 1 */
    0x2e, 0x31, 0x00, 0x00, 0x00, 0x03, 0x75, 0x72,   /* - 1 - - - - u r */
    0x6c, 0x00, 0x00, 0x00, 0x06, 0x70, 0x75, 0x62,   /* l - - - - p u b */
    0x6c, 0x69, 0x63, 0x00, 0x00, 0x00, 0x0a, 0x73,   /* l i c - - - - s */
    0x65, 0x74, 0x2d, 0x63, 0x6f, 0x6f, 0x6b, 0x69,   /* e t - c o o k i */
    0x65, 0x00, 0x00, 0x00, 0x0a, 0x6b, 0x65, 0x65,   /* e - - - - k e e */
    0x70, 0x2d, 0x61, 0x6c, 0x69, 0x76, 0x65, 0x00,   /* p - a l i v e - */
    0x00, 0x00, 0x06, 0x6f, 0x72, 0x69, 0x67, 0x69,   /* - - - o r i g i */
    0x6e, 0x31, 0x30, 0x30, 0x31, 0x30, 0x31, 0x32,   /* n 1 0 0 1 0 1 2 */
```

```
    0x30, 0x31, 0x32, 0x30, 0x32, 0x32, 0x30, 0x35,   /* 0 1 2 0 2 2 0 5 */
    0x32, 0x30, 0x36, 0x33, 0x30, 0x30, 0x33, 0x30,   /* 2 0 6 3 0 0 3 0 */
    0x32, 0x33, 0x30, 0x33, 0x33, 0x30, 0x34, 0x33,   /* 2 3 0 3 3 0 4 3 */
    0x30, 0x35, 0x33, 0x30, 0x36, 0x33, 0x30, 0x37,   /* 0 5 3 0 6 3 0 7 */
    0x34, 0x30, 0x32, 0x34, 0x30, 0x35, 0x34, 0x30,   /* 4 0 2 4 0 5 4 0 */
    0x36, 0x34, 0x30, 0x37, 0x34, 0x30, 0x38, 0x34,   /* 6 4 0 7 4 0 8 4 */
    0x30, 0x39, 0x34, 0x31, 0x30, 0x34, 0x31, 0x31,   /* 0 9 4 1 0 4 1 1 */
    0x34, 0x31, 0x32, 0x34, 0x31, 0x33, 0x34, 0x31,   /* 4 1 2 4 1 3 4 1 */
    0x34, 0x34, 0x31, 0x35, 0x34, 0x31, 0x36, 0x34,   /* 4 4 1 5 4 1 6 4 */
    0x31, 0x37, 0x35, 0x30, 0x32, 0x35, 0x30, 0x34,   /* 1 7 5 0 2 5 0 4 */
    0x35, 0x30, 0x35, 0x32, 0x30, 0x33, 0x20, 0x4e,   /* 5 0 5 2 0 3 - N */
    0x6f, 0x6e, 0x2d, 0x41, 0x75, 0x74, 0x68, 0x6f,   /* o n - A u t h o */
    0x72, 0x69, 0x74, 0x61, 0x74, 0x69, 0x76, 0x65,   /* r i t a t i v e */
    0x20, 0x49, 0x6e, 0x66, 0x6f, 0x72, 0x6d, 0x61,   /* - I n f o r m a */
    0x74, 0x69, 0x6f, 0x6e, 0x32, 0x30, 0x34, 0x20,   /* t i o n 2 0 4 - */
    0x4e, 0x6f, 0x20, 0x43, 0x6f, 0x6e, 0x74, 0x65,   /* N o - C o n t e */
    0x6e, 0x74, 0x33, 0x30, 0x31, 0x20, 0x4d, 0x6f,   /* n t 3 0 1 - M o */
    0x76, 0x65, 0x64, 0x20, 0x50, 0x65, 0x72, 0x6d,   /* v e d - P e r m */
    0x61, 0x6e, 0x65, 0x6e, 0x74, 0x6c, 0x79, 0x34,   /* a n e n t l y 4 */
    0x30, 0x30, 0x20, 0x42, 0x61, 0x64, 0x20, 0x52,   /* 0 0 - B a d - R */
    0x65, 0x71, 0x75, 0x65, 0x73, 0x74, 0x34, 0x30,   /* e q u e s t 4 0 */
    0x31, 0x20, 0x55, 0x6e, 0x61, 0x75, 0x74, 0x68,   /* 1 - U n a u t h */
    0x6f, 0x72, 0x69, 0x7a, 0x65, 0x64, 0x34, 0x30,   /* o r i z e d 4 0 */
    0x33, 0x20, 0x46, 0x6f, 0x72, 0x62, 0x69, 0x64,   /* 3 - F o r b i d */
    0x64, 0x65, 0x6e, 0x34, 0x30, 0x34, 0x20, 0x4e,   /* d e n 4 0 4 - N */
    0x6f, 0x74, 0x20, 0x46, 0x6f, 0x75, 0x6e, 0x64,   /* o t - F o u n d */
    0x35, 0x30, 0x30, 0x20, 0x49, 0x6e, 0x74, 0x65,   /* 5 0 0 - I n t e */
    0x72, 0x6e, 0x61, 0x6c, 0x20, 0x53, 0x65, 0x72,   /* r n a l - S e r */
    0x76, 0x65, 0x72, 0x20, 0x45, 0x72, 0x72, 0x6f,   /* v e r - E r r o */
    0x72, 0x35, 0x30, 0x31, 0x20, 0x4e, 0x6f, 0x74,   /* r 5 0 1 - N o t */
    0x20, 0x49, 0x6d, 0x70, 0x6c, 0x65, 0x6d, 0x65,   /* - I m p l e m e */
    0x6e, 0x74, 0x65, 0x64, 0x35, 0x30, 0x33, 0x20,   /* n t e d 5 0 3 - */
    0x53, 0x65, 0x72, 0x76, 0x69, 0x63, 0x65, 0x20,   /* S e r v i c e - */
    0x55, 0x6e, 0x61, 0x76, 0x61, 0x69, 0x6c, 0x61,   /* U n a v a i l a */
    0x62, 0x6c, 0x65, 0x4a, 0x61, 0x6e, 0x20, 0x46,   /* b l e J a n - F */
    0x65, 0x62, 0x20, 0x4d, 0x61, 0x72, 0x20, 0x41,   /* e b - M a r - A */
    0x70, 0x72, 0x20, 0x4d, 0x61, 0x79, 0x20, 0x4a,   /* p r - M a y - J */
    0x75, 0x6e, 0x20, 0x4a, 0x75, 0x6c, 0x20, 0x41,   /* u n - J u l - A */
    0x75, 0x67, 0x20, 0x53, 0x65, 0x70, 0x74, 0x20,   /* u g - S e p t - */
    0x4f, 0x63, 0x74, 0x20, 0x4e, 0x6f, 0x76, 0x20,   /* O c t - N o v - */
    0x44, 0x65, 0x63, 0x20, 0x30, 0x30, 0x3a, 0x30,   /* D e c - 0 0 - 0 */
    0x30, 0x3a, 0x30, 0x30, 0x20, 0x4d, 0x6f, 0x6e,   /* 0 - 0 0 - M o n */
    0x2c, 0x20, 0x54, 0x75, 0x65, 0x2c, 0x20, 0x57,   /* - - T u e - - W */
    0x65, 0x64, 0x2c, 0x20, 0x54, 0x68, 0x75, 0x2c,   /* e d - - T h u - */
    0x20, 0x46, 0x72, 0x69, 0x2c, 0x20, 0x53, 0x61,   /* - F r i - - S a */
    0x74, 0x2c, 0x20, 0x53, 0x75, 0x6e, 0x2c, 0x20,   /* t - - S u n - - */
    0x47, 0x4d, 0x54, 0x63, 0x68, 0x75, 0x6e, 0x6b,   /* G M T c h u n k */
    0x65, 0x64, 0x2c, 0x74, 0x65, 0x78, 0x74, 0x2f,   /* e d - t e x t - */
    0x68, 0x74, 0x6d, 0x6c, 0x2c, 0x69, 0x6d, 0x61,   /* h t m l - i m a */
    0x67, 0x65, 0x2f, 0x70, 0x6e, 0x67, 0x2c, 0x69,   /* g e - p n g - i */
    0x6d, 0x61, 0x67, 0x65, 0x2f, 0x6a, 0x70, 0x67,   /* m a g e - j p g */
    0x2c, 0x69, 0x6d, 0x61, 0x67, 0x65, 0x2f, 0x67,   /* - i m a g e - g */
    0x69, 0x66, 0x2c, 0x61, 0x70, 0x70, 0x6c, 0x69,   /* i f - a p p l i */
    0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x2f, 0x78,   /* c a t i o n - x */
    0x6d, 0x6c, 0x2c, 0x61, 0x70, 0x70, 0x6c, 0x69,   /* m l - a p p l i */
    0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x2f, 0x78,   /* c a t i o n - x */
    0x68, 0x74, 0x6d, 0x6c, 0x2b, 0x78, 0x6d, 0x6c,   /* h t m l - x m l */
    0x2c, 0x74, 0x65, 0x78, 0x74, 0x2f, 0x70, 0x6c,   /* - t e x t - p l */
    0x61, 0x69, 0x6e, 0x2c, 0x74, 0x65, 0x78, 0x74,   /* a i n - t e x t */
    0x2f, 0x6a, 0x61, 0x76, 0x61, 0x73, 0x63, 0x72,   /* - j a v a s c r */
    0x69, 0x70, 0x74, 0x2c, 0x70, 0x75, 0x62, 0x6c,   /* i p t - p u b l */
    0x69, 0x63, 0x70, 0x72, 0x69, 0x76, 0x61, 0x74,   /* i c p r i v a t */
    0x65, 0x6d, 0x61, 0x78, 0x2d, 0x61, 0x67, 0x65,   /* e m a x - a g e */
    0x3d, 0x67, 0x7a, 0x69, 0x70, 0x2c, 0x64, 0x65,   /* - g z i p - d e */
    0x66, 0x6c, 0x61, 0x74, 0x65, 0x2c, 0x73, 0x64,   /* f l a t e - s d */
    0x63, 0x68, 0x63, 0x68, 0x61, 0x72, 0x73, 0x65,   /* c h c h a r s e */
    0x74, 0x3d, 0x75, 0x74, 0x66, 0x2d, 0x38, 0x63,   /* t - u t f - 8 c */
    0x68, 0x61, 0x72, 0x73, 0x65, 0x74, 0x3d, 0x69,   /* h a r s e t - i */
    0x73, 0x6f, 0x2d, 0x38, 0x38, 0x35, 0x39, 0x2d,   /* s o - 8 8 5 9 - */
    0x31, 0x2c, 0x75, 0x74, 0x66, 0x2d, 0x2c, 0x2a,   /* 1 - u t f - - - */
    0x2c, 0x65, 0x6e, 0x71, 0x3d, 0x30, 0x2e          /* - e n q - 0 -   */
};


static ngx_http_spdy_request_header_t ngx_http_spdy_request_headers[] = {
    { 0, 6, "method", ngx_http_spdy_parse_method },
```

```
    { 0, 6, "scheme", ngx_http_spdy_parse_scheme },
    { 0, 4, "host", ngx_http_spdy_parse_host },
    { 0, 4, "path", ngx_http_spdy_parse_path },
    { 0, 7, "version", ngx_http_spdy_parse_version },
};

#define NGX_SPDY_REQUEST_HEADERS                                             \
    (sizeof(ngx_http_spdy_request_headers)                                   \
     / sizeof(ngx_http_spdy_request_header_t))


void
ngx_http_spdy_init(ngx_event_t *rev)
{
    int                        rc;
    ngx_connection_t           *c;
    ngx_pool_cleanup_t         *cln;
    ngx_http_connection_t      *hc;
    ngx_http_spdy_srv_conf_t   *sscf;
    ngx_http_spdy_main_conf_t  *smcf;
    ngx_http_spdy_connection_t  *sc;

    c = rev->data;
    hc = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "init spdy request");

    c->log->action = "processing SPDY";

    smcf = ngx_http_get_module_main_conf(hc->conf_ctx, ngx_http_spdy_module);

    if (smcf->recv_buffer == NULL) {
        smcf->recv_buffer = ngx_palloc(ngx_cycle->pool, smcf->recv_buffer_size);
        if (smcf->recv_buffer == NULL) {
            ngx_http_close_connection(c);
            return;
        }
    }

    sc = ngx_pcalloc(c->pool, sizeof(ngx_http_spdy_connection_t));
    if (sc == NULL) {
        ngx_http_close_connection(c);
        return;
    }

    sc->connection = c;
    sc->http_connection = hc;

    sc->send_window = NGX_SPDY_CONNECTION_WINDOW;
    sc->recv_window = NGX_SPDY_CONNECTION_WINDOW;

    sc->init_window = NGX_SPDY_INIT_STREAM_WINDOW;

    sc->handler = hc->proxy_protocol ? ngx_http_spdy_proxy_protocol
                                     : ngx_http_spdy_state_head;

    sc->zstream_in.zalloc = ngx_http_spdy_zalloc;
    sc->zstream_in.zfree = ngx_http_spdy_zfree;
    sc->zstream_in.opaque = sc;

    rc = inflateInit(&sc->zstream_in);
    if (rc != Z_OK) {
        ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                      "inflateInit() failed: %d", rc);
        ngx_http_close_connection(c);
        return;
    }

    sc->zstream_out.zalloc = ngx_http_spdy_zalloc;
    sc->zstream_out.zfree = ngx_http_spdy_zfree;
    sc->zstream_out.opaque = sc;

    sscf = ngx_http_get_module_srv_conf(hc->conf_ctx, ngx_http_spdy_module);

    rc = deflateInit2(&sc->zstream_out, (int) sscf->headers_comp,
                      Z_DEFLATED, 11, 4, Z_DEFAULT_STRATEGY);
```

```
449
450     if (rc != Z_OK) {
451         ngx_log_error(NGX_LOG_ALERT, c->log, 0,
452                       "deflateInit2() failed: %d", rc);
453         ngx_http_close_connection(c);
454         return;
455     }
456
457     rc = deflateSetDictionary(&sc->zstream_out, ngx_http_spdy_dict,
458                               sizeof(ngx_http_spdy_dict));
459     if (rc != Z_OK) {
460         ngx_log_error(NGX_LOG_ALERT, c->log, 0,
461                       "deflateSetDictionary() failed: %d", rc);
462         ngx_http_close_connection(c);
463         return;
464     }
465
466     sc->pool = ngx_create_pool(sscf->pool_size, sc->connection->log);
467     if (sc->pool == NULL) {
468         ngx_http_close_connection(c);
469         return;
470     }
471
472     cln = ngx_pool_cleanup_add(c->pool, sizeof(ngx_pool_cleanup_file_t));
473     if (cln == NULL) {
474         ngx_http_close_connection(c);
475         return;
476     }
477
478     cln->handler = ngx_http_spdy_pool_cleanup;
479     cln->data = sc;
480
481     sc->streams_index = ngx_pcalloc(sc->pool,
482                                     ngx_http_spdy_streams_index_size(sscf)
483                                     * sizeof(ngx_http_spdy_stream_t *));
484     if (sc->streams_index == NULL) {
485         ngx_http_close_connection(c);
486         return;
487     }
488
489     if (ngx_http_spdy_send_settings(sc) == NGX_ERROR) {
490         ngx_http_close_connection(c);
491         return;
492     }
493
494     if (ngx_http_spdy_send_window_update(sc, 0, NGX_SPDY_MAX_WINDOW
495                                                 - sc->recv_window)
496         == NGX_ERROR)
497     {
498         ngx_http_close_connection(c);
499         return;
500     }
501
502     sc->recv_window = NGX_SPDY_MAX_WINDOW;
503
504     ngx_queue_init(&sc->waiting);
505     ngx_queue_init(&sc->posted);
506
507     c->data = sc;
508
509     rev->handler = ngx_http_spdy_read_handler;
510     c->write->handler = ngx_http_spdy_write_handler;
511
512     ngx_http_spdy_read_handler(rev);
513 }
514
515
516 static void
517 ngx_http_spdy_read_handler(ngx_event_t *rev)
518 {
519     u_char                     *p, *end;
520     size_t                      available;
521     ssize_t                     n;
522     ngx_connection_t           *c;
523     ngx_http_spdy_main_conf_t  *smcf;
524     ngx_http_spdy_connection_t  *sc;
```

```
525
526     c = rev->data;
527     sc = c->data;
528
529     if (rev->timedout) {
530         ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
531         ngx_http_spdy_finalize_connection(sc, NGX_HTTP_REQUEST_TIME_OUT);
532         return;
533     }
534
535     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "spdy read handler");
536
537     sc->blocked = 1;
538
539     smcf = ngx_http_get_module_main_conf(sc->http_connection->conf_ctx,
540                                          ngx_http_spdy_module);
541
542     available = smcf->recv_buffer_size - 2 * NGX_SPDY_STATE_BUFFER_SIZE;
543
544     do {
545         p = smcf->recv_buffer;
546
547         ngx_memcpy(p, sc->buffer, NGX_SPDY_STATE_BUFFER_SIZE);
548         end = p + sc->buffer_used;
549
550         n = c->recv(c, end, available);
551
552         if (n == NGX_AGAIN) {
553             break;
554         }
555
556         if (n == 0 && (sc->incomplete || sc->processing)) {
557             ngx_log_error(NGX_LOG_INFO, c->log, 0,
558                           "client prematurely closed connection");
559         }
560
561         if (n == 0 || n == NGX_ERROR) {
562             ngx_http_spdy_finalize_connection(sc,
563                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
564             return;
565         }
566
567         end += n;
568
569         sc->buffer_used = 0;
570         sc->incomplete = 0;
571
572         do {
573             p = sc->handler(sc, p, end);
574
575             if (p == NULL) {
576                 return;
577             }
578
579         } while (p != end);
580
581     } while (rev->ready);
582
583     if (ngx_handle_read_event(rev, 0) != NGX_OK) {
584         ngx_http_spdy_finalize_connection(sc, NGX_HTTP_INTERNAL_SERVER_ERROR);
585         return;
586     }
587
588     if (sc->last_out && ngx_http_spdy_send_output_queue(sc) == NGX_ERROR) {
589         ngx_http_spdy_finalize_connection(sc, NGX_HTTP_CLIENT_CLOSED_REQUEST);
590         return;
591     }
592
593     sc->blocked = 0;
594
595     if (sc->processing) {
596         if (rev->timer_set) {
597             ngx_del_timer(rev);
598         }
599         return;
600     }
```

```
601
602     ngx_http_spdy_handle_connection(sc);
603 }
604
605
606 static void
607 ngx_http_spdy_write_handler(ngx_event_t *wev)
608 {
609     ngx_int_t                    rc;
610     ngx_queue_t                 *q;
611     ngx_connection_t            *c;
612     ngx_http_spdy_stream_t      *stream;
613     ngx_http_spdy_connection_t  *sc;
614
615     c = wev->data;
616     sc = c->data;
617
618     if (wev->timedout) {
619         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
620                        "spdy write event timed out");
621         ngx_http_spdy_finalize_connection(sc, NGX_HTTP_CLIENT_CLOSED_REQUEST);
622         return;
623     }
624
625     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "spdy write handler");
626
627     sc->blocked = 1;
628
629     rc = ngx_http_spdy_send_output_queue(sc);
630
631     if (rc == NGX_ERROR) {
632         ngx_http_spdy_finalize_connection(sc, NGX_HTTP_CLIENT_CLOSED_REQUEST);
633         return;
634     }
635
636     while (!ngx_queue_empty(&sc->posted)) {
637         q = ngx_queue_head(&sc->posted);
638
639         ngx_queue_remove(q);
640
641         stream = ngx_queue_data(q, ngx_http_spdy_stream_t, queue);
642
643         stream->handled = 0;
644
645         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
646                        "run spdy stream %ui", stream->id);
647
648         wev = stream->request->connection->write;
649         wev->handler(wev);
650     }
651
652     sc->blocked = 0;
653
654     if (rc == NGX_AGAIN) {
655         return;
656     }
657
658     ngx_http_spdy_handle_connection(sc);
659 }
660
661
662 ngx_int_t
663 ngx_http_spdy_send_output_queue(ngx_http_spdy_connection_t *sc)
664 {
665     ngx_chain_t                *cl;
666     ngx_event_t                *wev;
667     ngx_connection_t           *c;
668     ngx_http_core_loc_conf_t   *clcf;
669     ngx_http_spdy_out_frame_t  *out, *frame, *fn;
670
671     c = sc->connection;
672
673     if (c->error) {
674         return NGX_ERROR;
675     }
676
```

```
677     wev = c->write;
678
679     if (!wev->ready) {
680         return NGX_OK;
681     }
682
683     cl = NULL;
684     out = NULL;
685
686     for (frame = sc->last_out; frame; frame = fn) {
687         frame->last->next = cl;
688         cl = frame->first;
689
690         fn = frame->next;
691         frame->next = out;
692         out = frame;
693
694         ngx_log_debug5(NGX_LOG_DEBUG_HTTP, c->log, 0,
695                        "spdy frame out: %p sid:%ui prio:%ui bl:%d len:%uz",
696                        out, out->stream ? out->stream->id : 0, out->priority,
697                        out->blocked, out->length);
698     }
699
700     cl = c->send_chain(c, cl, 0);
701
702     if (cl == NGX_CHAIN_ERROR) {
703         c->error = 1;
704
705         if (!sc->blocked) {
706             ngx_post_event(wev, &ngx_posted_events);
707         }
708
709         return NGX_ERROR;
710     }
711
712     clcf = ngx_http_get_module_loc_conf(sc->http_connection->conf_ctx,
713                                         ngx_http_core_module);
714
715     if (ngx_handle_write_event(wev, clcf->send_lowat) != NGX_OK) {
716         return NGX_ERROR; /* FIXME */
717     }
718
719     if (cl) {
720         ngx_add_timer(wev, clcf->send_timeout);
721
722     } else {
723         if (wev->timer_set) {
724             ngx_del_timer(wev);
725         }
726     }
727
728     for ( /* void */ ; out; out = fn) {
729         fn = out->next;
730
731         if (out->handler(sc, out) != NGX_OK) {
732             out->blocked = 1;
733             out->priority = NGX_SPDY_HIGHEST_PRIORITY;
734             break;
735         }
736
737         ngx_log_debug4(NGX_LOG_DEBUG_HTTP, c->log, 0,
738                        "spdy frame sent: %p sid:%ui bl:%d len:%uz",
739                        out, out->stream ? out->stream->id : 0,
740                        out->blocked, out->length);
741     }
742
743     frame = NULL;
744
745     for ( /* void */ ; out; out = fn) {
746         fn = out->next;
747         out->next = frame;
748         frame = out;
749     }
750
751     sc->last_out = frame;
752
```

```
753     return NGX_OK;
754 }
755
756
757 static void
758 ngx_http_spdy_handle_connection(ngx_http_spdy_connection_t *sc)
759 {
760     ngx_connection_t          *c;
761     ngx_http_spdy_srv_conf_t  *sscf;
762
763     if (sc->last_out || sc->processing) {
764         return;
765     }
766
767     c = sc->connection;
768
769     if (c->error) {
770         ngx_http_close_connection(c);
771         return;
772     }
773
774     if (c->buffered) {
775         return;
776     }
777
778     sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
779                                         ngx_http_spdy_module);
780     if (sc->incomplete) {
781         ngx_add_timer(c->read, sscf->recv_timeout);
782         return;
783     }
784
785     if (ngx_terminate || ngx_exiting) {
786         ngx_http_close_connection(c);
787         return;
788     }
789
790     ngx_destroy_pool(sc->pool);
791
792     sc->pool = NULL;
793     sc->free_ctl_frames = NULL;
794     sc->free_fake_connections = NULL;
795
796 #if (NGX_HTTP_SSL)
797     if (c->ssl) {
798         ngx_ssl_free_buffer(c);
799     }
800 #endif
801
802     c->destroyed = 1;
803     c->idle = 1;
804     ngx_reusable_connection(c, 1);
805
806     c->write->handler = ngx_http_empty_handler;
807     c->read->handler = ngx_http_spdy_keepalive_handler;
808
809     if (c->write->timer_set) {
810         ngx_del_timer(c->write);
811     }
812
813     ngx_add_timer(c->read, sscf->keepalive_timeout);
814 }
815
816
817 static u_char *
818 ngx_http_spdy_proxy_protocol(ngx_http_spdy_connection_t *sc, u_char *pos,
819     u_char *end)
820 {
821     ngx_log_t  *log;
822
823     log = sc->connection->log;
824     log->action = "reading PROXY protocol";
825
826     pos = ngx_proxy_protocol_parse(sc->connection, pos, end);
827
828     log->action = "processing SPDY";
```

```
829
830     if (pos == NULL) {
831         return ngx_http_spdy_state_protocol_error(sc);
832     }
833
834     return ngx_http_spdy_state_complete(sc, pos, end);
835 }
836
837
838 static u_char *
839 ngx_http_spdy_state_head(ngx_http_spdy_connection_t *sc, u_char *pos,
840     u_char *end)
841 {
842     uint32_t    head, flen;
843     ngx_uint_t  type;
844
845     if (end - pos < NGX_SPDY_FRAME_HEADER_SIZE) {
846         return ngx_http_spdy_state_save(sc, pos, end,
847                                         ngx_http_spdy_state_head);
848     }
849
850     head = ngx_spdy_frame_parse_uint32(pos);
851
852     pos += sizeof(uint32_t);
853
854     flen = ngx_spdy_frame_parse_uint32(pos);
855
856     sc->flags = ngx_spdy_frame_flags(flen);
857     sc->length = ngx_spdy_frame_length(flen);
858
859     pos += sizeof(uint32_t);
860
861     ngx_log_debug3(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
862                    "process spdy frame head:%08XD f:%Xd l:%uz",
863                    head, sc->flags, sc->length);
864
865     if (ngx_spdy_ctl_frame_check(head)) {
866         type = ngx_spdy_ctl_frame_type(head);
867
868         switch (type) {
869
870         case NGX_SPDY_SYN_STREAM:
871             return ngx_http_spdy_state_syn_stream(sc, pos, end);
872
873         case NGX_SPDY_SYN_REPLY:
874             ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
875                           "client sent unexpected SYN_REPLY frame");
876             return ngx_http_spdy_state_protocol_error(sc);
877
878         case NGX_SPDY_RST_STREAM:
879             return ngx_http_spdy_state_rst_stream(sc, pos, end);
880
881         case NGX_SPDY_SETTINGS:
882             return ngx_http_spdy_state_settings(sc, pos, end);
883
884         case NGX_SPDY_PING:
885             return ngx_http_spdy_state_ping(sc, pos, end);
886
887         case NGX_SPDY_GOAWAY:
888             return ngx_http_spdy_state_skip(sc, pos, end); /* TODO */
889
890         case NGX_SPDY_HEADERS:
891             ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
892                           "client sent unexpected HEADERS frame");
893             return ngx_http_spdy_state_protocol_error(sc);
894
895         case NGX_SPDY_WINDOW_UPDATE:
896             return ngx_http_spdy_state_window_update(sc, pos, end);
897
898         default:
899             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
900                            "spdy control frame with unknown type %ui", type);
901             return ngx_http_spdy_state_skip(sc, pos, end);
902         }
903     }
904
```

```
    if (ngx_spdy_data_frame_check(head)) {
        sc->stream = ngx_http_spdy_get_stream_by_id(sc, head);
        return ngx_http_spdy_state_data(sc, pos, end);
    }

    ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                  "client sent invalid frame");

    return ngx_http_spdy_state_protocol_error(sc);
}


static u_char *
ngx_http_spdy_state_syn_stream(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    ngx_uint_t                 sid, prio;
    ngx_http_spdy_stream_t    *stream;
    ngx_http_spdy_srv_conf_t  *sscf;

    if (end - pos < NGX_SPDY_SYN_STREAM_SIZE) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_syn_stream);
    }

    if (sc->length <= NGX_SPDY_SYN_STREAM_SIZE) {
        ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                      "client sent SYN_STREAM frame with incorrect length %uz",
                      sc->length);

        return ngx_http_spdy_state_protocol_error(sc);
    }

    sc->length -= NGX_SPDY_SYN_STREAM_SIZE;

    sid = ngx_spdy_frame_parse_sid(pos);
    prio = pos[8] >> 5;

    pos += NGX_SPDY_SYN_STREAM_SIZE;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy SYN_STREAM frame sid:%ui prio:%ui", sid, prio);

    if (sid % 2 == 0 || sid <= sc->last_sid) {
        ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                      "client sent SYN_STREAM frame "
                      "with invalid Stream-ID %ui", sid);

        stream = ngx_http_spdy_get_stream_by_id(sc, sid);

        if (stream) {
            if (ngx_http_spdy_terminate_stream(sc, stream,
                                               NGX_SPDY_PROTOCOL_ERROR)
                != NGX_OK)
            {
                return ngx_http_spdy_state_internal_error(sc);
            }
        }

        return ngx_http_spdy_state_protocol_error(sc);
    }

    sc->last_sid = sid;

    sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
                                        ngx_http_spdy_module);

    if (sc->processing >= sscf->concurrent_streams) {

        ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                      "concurrent streams exceeded %ui", sc->processing);

        if (ngx_http_spdy_send_rst_stream(sc, sid, NGX_SPDY_REFUSED_STREAM,
                                          prio)
            != NGX_OK)
        {
```

```
            return ngx_http_spdy_state_internal_error(sc);
        }

        return ngx_http_spdy_state_headers_skip(sc, pos, end);
    }

    stream = ngx_http_spdy_create_stream(sc, sid, prio);
    if (stream == NULL) {
        return ngx_http_spdy_state_internal_error(sc);
    }

    stream->in_closed = (sc->flags & NGX_SPDY_FLAG_FIN) ? 1 : 0;

    stream->request->request_length = NGX_SPDY_FRAME_HEADER_SIZE
                                      + NGX_SPDY_SYN_STREAM_SIZE
                                      + sc->length;

    sc->stream = stream;

     return ngx_http_spdy_state_headers(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_headers(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    int                 z;
    size_t              size;
    ngx_buf_t          *buf;
    ngx_int_t           rc;
    ngx_http_request_t  *r;

    size = end - pos;

    if (size == 0) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_headers);
    }

    if (size > sc->length) {
        size = sc->length;
    }

    r = sc->stream->request;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "process spdy header block %uz of %uz", size, sc->length);

    buf = r->header_in;

    sc->zstream_in.next_in = pos;
    sc->zstream_in.avail_in = size;
    sc->zstream_in.next_out = buf->last;

    /* one byte is reserved for null-termination of the last header value */
    sc->zstream_in.avail_out = buf->end - buf->last - 1;

    z = inflate(&sc->zstream_in, Z_NO_FLUSH);

    if (z == Z_NEED_DICT) {
        z = inflateSetDictionary(&sc->zstream_in, ngx_http_spdy_dict,
                                 sizeof(ngx_http_spdy_dict));

        if (z != Z_OK) {
            if (z == Z_DATA_ERROR) {
                ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                              "client sent SYN_STREAM frame with header "
                              "block encoded using wrong dictionary: %ul",
                              (u_long) sc->zstream_in.adler);

                return ngx_http_spdy_state_protocol_error(sc);
            }

            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "inflateSetDictionary() failed: %d", z);
```

```
            return ngx_http_spdy_state_internal_error(sc);
        }

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "spdy inflateSetDictionary(): %d", z);

        z = sc->zstream_in.avail_in ? inflate(&sc->zstream_in, Z_NO_FLUSH)
                                    : Z_OK;
    }

    ngx_log_debug5(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "spdy inflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
                   sc->zstream_in.next_in, sc->zstream_in.next_out,
                   sc->zstream_in.avail_in, sc->zstream_in.avail_out,
                   z);

    if (z != Z_OK) {
        return ngx_http_spdy_state_inflate_error(sc, z);
    }

    sc->length -= sc->zstream_in.next_in - pos;
    pos = sc->zstream_in.next_in;

    buf->last = sc->zstream_in.next_out;

    if (r->headers_in.headers.part.elts == NULL) {

        if (buf->last - buf->pos < NGX_SPDY_NV_NUM_SIZE) {

            if (sc->length == 0) {
                ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                               "premature end of spdy header block");

                return ngx_http_spdy_state_headers_error(sc, pos, end);
            }

            return ngx_http_spdy_state_save(sc, pos, end,
                                            ngx_http_spdy_state_headers);
        }

        sc->entries = ngx_spdy_frame_parse_uint32(buf->pos);

        buf->pos += NGX_SPDY_NV_NUM_SIZE;

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "spdy header block has %ui entries",
                       sc->entries);

        if (ngx_list_init(&r->headers_in.headers, r->pool, 20,
                          sizeof(ngx_table_elt_t))
            != NGX_OK)
        {
            ngx_http_spdy_close_stream(sc->stream,
                                       NGX_HTTP_INTERNAL_SERVER_ERROR);
            return ngx_http_spdy_state_headers_skip(sc, pos, end);
        }

        if (ngx_array_init(&r->headers_in.cookies, r->pool, 2,
                           sizeof(ngx_table_elt_t *))
            != NGX_OK)
        {
            ngx_http_spdy_close_stream(sc->stream,
                                       NGX_HTTP_INTERNAL_SERVER_ERROR);
            return ngx_http_spdy_state_headers_skip(sc, pos, end);
        }
    }

    while (sc->entries) {

        rc = ngx_http_spdy_parse_header(r);

        switch (rc) {

        case NGX_DONE:
            sc->entries--;
```

```
1133
1134        case NGX_OK:
1135            break;
1136
1137        case NGX_AGAIN:
1138
1139            if (sc->zstream_in.avail_in) {
1140
1141                rc = ngx_http_spdy_alloc_large_header_buffer(r);
1142
1143                if (rc == NGX_DECLINED) {
1144                    ngx_http_finalize_request(r,
1145                                              NGX_HTTP_REQUEST_HEADER_TOO_LARGE);
1146                    return ngx_http_spdy_state_headers_skip(sc, pos, end);
1147                }
1148
1149                if (rc != NGX_OK) {
1150                    ngx_http_spdy_close_stream(sc->stream,
1151                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
1152                    return ngx_http_spdy_state_headers_skip(sc, pos, end);
1153                }
1154
1155                /* null-terminate the last processed header name or value */
1156                *buf->pos = '\0';
1157
1158                buf = r->header_in;
1159
1160                sc->zstream_in.next_out = buf->last;
1161
1162                /* one byte is reserved for null-termination */
1163                sc->zstream_in.avail_out = buf->end - buf->last - 1;
1164
1165                z = inflate(&sc->zstream_in, Z_NO_FLUSH);
1166
1167                ngx_log_debug5(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1168                               "spdy inflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
1169                               sc->zstream_in.next_in, sc->zstream_in.next_out,
1170                               sc->zstream_in.avail_in, sc->zstream_in.avail_out,
1171                               z);
1172
1173                if (z != Z_OK) {
1174                    return ngx_http_spdy_state_inflate_error(sc, z);
1175                }
1176
1177                sc->length -= sc->zstream_in.next_in - pos;
1178                pos = sc->zstream_in.next_in;
1179
1180                buf->last = sc->zstream_in.next_out;
1181
1182                continue;
1183            }
1184
1185            if (sc->length == 0) {
1186                ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1187                               "premature end of spdy header block");
1188
1189                return ngx_http_spdy_state_headers_error(sc, pos, end);
1190            }
1191
1192            return ngx_http_spdy_state_save(sc, pos, end,
1193                                            ngx_http_spdy_state_headers);
1194
1195        case NGX_HTTP_PARSE_INVALID_HEADER:
1196            ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
1197            return ngx_http_spdy_state_headers_skip(sc, pos, end);
1198
1199        default: /* NGX_ERROR */
1200            return ngx_http_spdy_state_headers_error(sc, pos, end);
1201        }
1202
1203        /* a header line has been parsed successfully */
1204
1205        rc = ngx_http_spdy_handle_request_header(r);
1206
1207        if (rc != NGX_OK) {
1208            if (rc == NGX_HTTP_PARSE_INVALID_HEADER) {
```

```
                ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
                return ngx_http_spdy_state_headers_skip(sc, pos, end);
            }

            if (rc != NGX_ABORT) {
                ngx_http_spdy_close_stream(sc->stream,
                                           NGX_HTTP_INTERNAL_SERVER_ERROR);
            }

            return ngx_http_spdy_state_headers_skip(sc, pos, end);
        }
    }

    if (buf->pos != buf->last || sc->zstream_in.avail_in) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "incorrect number of spdy header block entries");

        return ngx_http_spdy_state_headers_error(sc, pos, end);
    }

    if (sc->length) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_headers);
    }

    /* null-terminate the last header value */
    *buf->pos = '\0';

    ngx_http_spdy_run_request(r);

    return ngx_http_spdy_state_complete(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_headers_skip(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    int     n;
    size_t  size;
    u_char  buffer[NGX_SPDY_SKIP_HEADERS_BUFFER_SIZE];

    if (sc->length == 0) {
        return ngx_http_spdy_state_complete(sc, pos, end);
    }

    size = end - pos;

    if (size == 0) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_headers_skip);
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy header block skip %uz of %uz", size, sc->length);

    sc->zstream_in.next_in = pos;
    sc->zstream_in.avail_in = (size < sc->length) ? size : sc->length;

    while (sc->zstream_in.avail_in) {
        sc->zstream_in.next_out = buffer;
        sc->zstream_in.avail_out = NGX_SPDY_SKIP_HEADERS_BUFFER_SIZE;

        n = inflate(&sc->zstream_in, Z_NO_FLUSH);

        ngx_log_debug5(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                       "spdy inflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
                       sc->zstream_in.next_in, sc->zstream_in.next_out,
                       sc->zstream_in.avail_in, sc->zstream_in.avail_out,
                       n);

        if (n != Z_OK) {
            return ngx_http_spdy_state_inflate_error(sc, n);
        }
    }

```

```
    pos = sc->zstream_in.next_in;

    if (size < sc->length) {
        sc->length -= size;
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_headers_skip);
    }

    return ngx_http_spdy_state_complete(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_headers_error(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    ngx_http_spdy_stream_t  *stream;

    stream = sc->stream;

    ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                  "client sent SYN_STREAM frame for stream %ui "
                  "with invalid header block", stream->id);

    if (ngx_http_spdy_send_rst_stream(sc, stream->id, NGX_SPDY_PROTOCOL_ERROR,
                                      stream->priority)
        != NGX_OK)
    {
        return ngx_http_spdy_state_internal_error(sc);
    }

    stream->out_closed = 1;

    ngx_http_spdy_close_stream(stream, NGX_HTTP_BAD_REQUEST);

    return ngx_http_spdy_state_headers_skip(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_window_update(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    size_t                   delta;
    ngx_uint_t               sid;
    ngx_event_t             *wev;
    ngx_queue_t             *q;
    ngx_http_spdy_stream_t  *stream;

    if (end - pos < NGX_SPDY_WINDOW_UPDATE_SIZE) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_window_update);
    }

    if (sc->length != NGX_SPDY_WINDOW_UPDATE_SIZE) {
        ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                      "client sent WINDOW_UPDATE frame "
                      "with incorrect length %uz", sc->length);

        return ngx_http_spdy_state_protocol_error(sc);
    }

    sid = ngx_spdy_frame_parse_sid(pos);

    pos += NGX_SPDY_SID_SIZE;

    delta = ngx_spdy_frame_parse_delta(pos);

    pos += NGX_SPDY_DELTA_SIZE;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy WINDOW_UPDATE sid:%ui delta:%ui", sid, delta);

    if (sid) {
        stream = ngx_http_spdy_get_stream_by_id(sc, sid);

```

```
1361         if (stream == NULL) {
1362             ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1363                            "unknown spdy stream");
1364 
1365             return ngx_http_spdy_state_complete(sc, pos, end);
1366         }
1367 
1368         if (stream->send_window > (ssize_t) (NGX_SPDY_MAX_WINDOW - delta)) {
1369 
1370             ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1371                           "client violated flow control for stream %ui: "
1372                           "received WINDOW_UPDATE frame with delta %uz "
1373                           "not allowed for window %z",
1374                           sid, delta, stream->send_window);
1375 
1376             if (ngx_http_spdy_terminate_stream(sc, stream,
1377                                                NGX_SPDY_FLOW_CONTROL_ERROR)
1378                 == NGX_ERROR)
1379             {
1380                 return ngx_http_spdy_state_internal_error(sc);
1381             }
1382 
1383             return ngx_http_spdy_state_complete(sc, pos, end);
1384         }
1385 
1386         stream->send_window += delta;
1387 
1388         if (stream->exhausted) {
1389             stream->exhausted = 0;
1390 
1391             wev = stream->request->connection->write;
1392 
1393             if (!wev->timer_set) {
1394                 wev->delayed = 0;
1395                 wev->handler(wev);
1396             }
1397         }
1398 
1399     } else {
1400         sc->send_window += delta;
1401 
1402         if (sc->send_window > NGX_SPDY_MAX_WINDOW) {
1403             ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1404                           "client violated connection flow control: "
1405                           "received WINDOW_UPDATE frame with delta %uz "
1406                           "not allowed for window %uz",
1407                           delta, sc->send_window);
1408 
1409             return ngx_http_spdy_state_protocol_error(sc);
1410         }
1411 
1412         while (!ngx_queue_empty(&sc->waiting)) {
1413             q = ngx_queue_head(&sc->waiting);
1414 
1415             ngx_queue_remove(q);
1416 
1417             stream = ngx_queue_data(q, ngx_http_spdy_stream_t, queue);
1418 
1419             stream->handled = 0;
1420 
1421             wev = stream->request->connection->write;
1422 
1423             if (!wev->timer_set) {
1424                 wev->delayed = 0;
1425                 wev->handler(wev);
1426 
1427                 if (sc->send_window == 0) {
1428                     break;
1429                 }
1430             }
1431         }
1432     }
1433 
1434     return ngx_http_spdy_state_complete(sc, pos, end);
1435 }
1436 
```

```
1437
1438 static u_char *
1439 ngx_http_spdy_state_data(ngx_http_spdy_connection_t *sc, u_char *pos,
1440     u_char *end)
1441 {
1442     ngx_http_spdy_stream_t  *stream;
1443
1444     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1445                    "spdy DATA frame");
1446
1447     if (sc->length > sc->recv_window) {
1448         ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1449                       "client violated connection flow control: "
1450                       "received DATA frame length %uz, available window %uz",
1451                       sc->length, sc->recv_window);
1452
1453         return ngx_http_spdy_state_protocol_error(sc);
1454     }
1455
1456     sc->recv_window -= sc->length;
1457
1458     if (sc->recv_window < NGX_SPDY_MAX_WINDOW / 4) {
1459
1460         if (ngx_http_spdy_send_window_update(sc, 0,
1461                                              NGX_SPDY_MAX_WINDOW
1462                                              - sc->recv_window)
1463             == NGX_ERROR)
1464         {
1465             return ngx_http_spdy_state_internal_error(sc);
1466         }
1467
1468         sc->recv_window = NGX_SPDY_MAX_WINDOW;
1469     }
1470
1471     stream = sc->stream;
1472
1473     if (stream == NULL) {
1474         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1475                        "unknown spdy stream");
1476
1477         return ngx_http_spdy_state_skip(sc, pos, end);
1478     }
1479
1480     if (sc->length > stream->recv_window) {
1481         ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1482                       "client violated flow control for stream %ui: "
1483                       "received DATA frame length %uz, available window %uz",
1484                       stream->id, sc->length, stream->recv_window);
1485
1486         if (ngx_http_spdy_terminate_stream(sc, stream,
1487                                            NGX_SPDY_FLOW_CONTROL_ERROR)
1488             == NGX_ERROR)
1489         {
1490             return ngx_http_spdy_state_internal_error(sc);
1491         }
1492
1493         return ngx_http_spdy_state_skip(sc, pos, end);
1494     }
1495
1496     stream->recv_window -= sc->length;
1497
1498     if (stream->recv_window < NGX_SPDY_STREAM_WINDOW / 4) {
1499
1500         if (ngx_http_spdy_send_window_update(sc, stream->id,
1501                                              NGX_SPDY_STREAM_WINDOW
1502                                              - stream->recv_window)
1503             == NGX_ERROR)
1504         {
1505             return ngx_http_spdy_state_internal_error(sc);
1506         }
1507
1508         stream->recv_window = NGX_SPDY_STREAM_WINDOW;
1509     }
1510
1511     if (stream->in_closed) {
1512         ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
```

```
                      "client sent DATA frame for half-closed stream %ui",
                      stream->id);

        if (ngx_http_spdy_terminate_stream(sc, stream,
                                           NGX_SPDY_STREAM_ALREADY_CLOSED)
            == NGX_ERROR)
        {
            return ngx_http_spdy_state_internal_error(sc);
        }

        return ngx_http_spdy_state_skip(sc, pos, end);
    }

    return ngx_http_spdy_state_read_data(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_read_data(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    size_t                     size;
    ssize_t                    n;
    ngx_buf_t                 *buf;
    ngx_int_t                  rc;
    ngx_temp_file_t           *tf;
    ngx_http_request_t        *r;
    ngx_http_spdy_stream_t    *stream;
    ngx_http_request_body_t   *rb;
    ngx_http_core_loc_conf_t  *clcf;

    stream = sc->stream;

    if (stream == NULL) {
        return ngx_http_spdy_state_skip(sc, pos, end);
    }

    if (stream->skip_data) {

        if (sc->flags & NGX_SPDY_FLAG_FIN) {
            stream->in_closed = 1;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                       "skipping spdy DATA frame, reason: %d",
                       stream->skip_data);

        return ngx_http_spdy_state_skip(sc, pos, end);
    }

    size = end - pos;

    if (size > sc->length) {
        size = sc->length;
    }

    r = stream->request;

    if (r->request_body == NULL
        && ngx_http_spdy_init_request_body(r) != NGX_OK)
    {
        stream->skip_data = NGX_SPDY_DATA_INTERNAL_ERROR;
        return ngx_http_spdy_state_skip(sc, pos, end);
    }

    rb = r->request_body;
    tf = rb->temp_file;
    buf = rb->buf;

    if (size) {
        rb->rest += size;

        if (r->headers_in.content_length_n != -1
            && r->headers_in.content_length_n < rb->rest)
        {
            ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
```

```
                          "client intended to send body data "
                          "larger than declared");

            stream->skip_data = NGX_SPDY_DATA_ERROR;
            goto error;

        } else {
            clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

            if (clcf->client_max_body_size
                && clcf->client_max_body_size < rb->rest)
            {
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "client intended to send "
                              "too large chunked body: %O bytes", rb->rest);

                stream->skip_data = NGX_SPDY_DATA_ERROR;
                goto error;
            }
        }

        sc->length -= size;

        if (tf) {
            buf->start = pos;
            buf->pos = pos;

            pos += size;

            buf->end = pos;
            buf->last = pos;

            n = ngx_write_chain_to_temp_file(tf, rb->bufs);

            /* TODO: n == 0 or not complete and level event */

            if (n == NGX_ERROR) {
                stream->skip_data = NGX_SPDY_DATA_INTERNAL_ERROR;
                goto error;
            }

            tf->offset += n;

        } else {
            buf->last = ngx_cpymem(buf->last, pos, size);
            pos += size;
        }

        r->request_length += size;
    }

    if (sc->length) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_read_data);
    }

    if (sc->flags & NGX_SPDY_FLAG_FIN) {

        stream->in_closed = 1;

        if (r->headers_in.content_length_n < 0) {
            r->headers_in.content_length_n = rb->rest;

        } else if (r->headers_in.content_length_n != rb->rest) {
            ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                          "client prematurely closed stream: "
                          "only %O out of %O bytes of request body received",
                          rb->rest, r->headers_in.content_length_n);

            stream->skip_data = NGX_SPDY_DATA_ERROR;
            goto error;
        }

        if (tf) {
            ngx_memzero(buf, sizeof(ngx_buf_t));

```

```
            buf->in_file = 1;
            buf->file_last = tf->file.offset;
            buf->file = &tf->file;

            rb->buf = NULL;
        }

        if (rb->post_handler) {
            r->read_event_handler = ngx_http_block_reading;
            rb->post_handler(r);
        }
    }

    return ngx_http_spdy_state_complete(sc, pos, end);

error:

    if (rb->post_handler) {

        if (stream->skip_data == NGX_SPDY_DATA_ERROR) {
            rc = (r->headers_in.content_length_n == -1)
                 ? NGX_HTTP_REQUEST_ENTITY_TOO_LARGE
                 : NGX_HTTP_BAD_REQUEST;

        } else {
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        ngx_http_finalize_request(r, rc);
    }

    return ngx_http_spdy_state_skip(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_rst_stream(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    ngx_uint_t               sid, status;
    ngx_event_t             *ev;
    ngx_connection_t        *fc;
    ngx_http_spdy_stream_t  *stream;

    if (end - pos < NGX_SPDY_RST_STREAM_SIZE) {
        return ngx_http_spdy_state_save(sc, pos, end,
                                        ngx_http_spdy_state_rst_stream);
    }

    if (sc->length != NGX_SPDY_RST_STREAM_SIZE) {
        ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                      "client sent RST_STREAM frame with incorrect length %uz",
                      sc->length);

        return ngx_http_spdy_state_protocol_error(sc);
    }

    sid = ngx_spdy_frame_parse_sid(pos);

    pos += NGX_SPDY_SID_SIZE;

    status = ngx_spdy_frame_parse_uint32(pos);

    pos += sizeof(uint32_t);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy RST_STREAM sid:%ui st:%ui", sid, status);

    stream = ngx_http_spdy_get_stream_by_id(sc, sid);

    if (stream == NULL) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                       "unknown spdy stream");

        return ngx_http_spdy_state_complete(sc, pos, end);
    }
```

```
1741
1742        stream->in_closed = 1;
1743        stream->out_closed = 1;
1744
1745        fc = stream->request->connection;
1746        fc->error = 1;
1747
1748        switch (status) {
1749
1750        case NGX_SPDY_CANCEL:
1751            ngx_log_error(NGX_LOG_INFO, fc->log, 0,
1752                          "client canceled stream %ui", sid);
1753            break;
1754
1755        case NGX_SPDY_INTERNAL_ERROR:
1756            ngx_log_error(NGX_LOG_INFO, fc->log, 0,
1757                          "client terminated stream %ui due to internal error",
1758                          sid);
1759            break;
1760
1761        default:
1762            ngx_log_error(NGX_LOG_INFO, fc->log, 0,
1763                          "client terminated stream %ui with status %ui",
1764                          sid, status);
1765            break;
1766        }
1767
1768        ev = fc->read;
1769        ev->handler(ev);
1770
1771        return ngx_http_spdy_state_complete(sc, pos, end);
1772    }
1773
1774
1775    static u_char *
1776    ngx_http_spdy_state_ping(ngx_http_spdy_connection_t *sc, u_char *pos,
1777        u_char *end)
1778    {
1779        u_char                     *p;
1780        ngx_buf_t                  *buf;
1781        ngx_http_spdy_out_frame_t  *frame;
1782
1783        if (end - pos < NGX_SPDY_PING_SIZE) {
1784            return ngx_http_spdy_state_save(sc, pos, end,
1785                                            ngx_http_spdy_state_ping);
1786        }
1787
1788        if (sc->length != NGX_SPDY_PING_SIZE) {
1789            ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1790                          "client sent PING frame with incorrect length %uz",
1791                          sc->length);
1792
1793            return ngx_http_spdy_state_protocol_error(sc);
1794        }
1795
1796        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1797                       "spdy PING frame");
1798
1799        frame = ngx_http_spdy_get_ctl_frame(sc, NGX_SPDY_PING_SIZE,
1800                                            NGX_SPDY_HIGHEST_PRIORITY);
1801        if (frame == NULL) {
1802            return ngx_http_spdy_state_internal_error(sc);
1803        }
1804
1805        buf = frame->first->buf;
1806
1807        p = buf->pos;
1808
1809        p = ngx_spdy_frame_write_head(p, NGX_SPDY_PING);
1810        p = ngx_spdy_frame_write_flags_and_len(p, 0, NGX_SPDY_PING_SIZE);
1811
1812        p = ngx_cpymem(p, pos, NGX_SPDY_PING_SIZE);
1813
1814        buf->last = p;
1815
1816        ngx_http_spdy_queue_frame(sc, frame);
```

```
1817
1818        pos += NGX_SPDY_PING_SIZE;
1819
1820        return ngx_http_spdy_state_complete(sc, pos, end);
1821    }
1822
1823
1824    static u_char *
1825    ngx_http_spdy_state_skip(ngx_http_spdy_connection_t *sc, u_char *pos,
1826        u_char *end)
1827    {
1828        size_t  size;
1829
1830        size = end - pos;
1831
1832        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1833                       "spdy frame skip %uz of %uz", size, sc->length);
1834
1835        if (size < sc->length) {
1836            sc->length -= size;
1837            return ngx_http_spdy_state_save(sc, end, end,
1838                                            ngx_http_spdy_state_skip);
1839        }
1840
1841        return ngx_http_spdy_state_complete(sc, pos + sc->length, end);
1842    }
1843
1844
1845    static u_char *
1846    ngx_http_spdy_state_settings(ngx_http_spdy_connection_t *sc, u_char *pos,
1847        u_char *end)
1848    {
1849        ngx_uint_t  fid, val;
1850
1851        if (sc->entries == 0) {
1852
1853            if (end - pos < NGX_SPDY_SETTINGS_NUM_SIZE) {
1854                return ngx_http_spdy_state_save(sc, pos, end,
1855                                                ngx_http_spdy_state_settings);
1856            }
1857
1858            sc->entries = ngx_spdy_frame_parse_uint32(pos);
1859
1860            pos += NGX_SPDY_SETTINGS_NUM_SIZE;
1861            sc->length -= NGX_SPDY_SETTINGS_NUM_SIZE;
1862
1863            if (sc->length < sc->entries * NGX_SPDY_SETTINGS_PAIR_SIZE) {
1864                ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1865                              "client sent SETTINGS frame with incorrect "
1866                              "length %uz or number of entries %ui",
1867                              sc->length, sc->entries);
1868
1869                return ngx_http_spdy_state_protocol_error(sc);
1870            }
1871
1872            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1873                           "spdy SETTINGS frame has %ui entries", sc->entries);
1874        }
1875
1876        while (sc->entries) {
1877            if (end - pos < NGX_SPDY_SETTINGS_PAIR_SIZE) {
1878                return ngx_http_spdy_state_save(sc, pos, end,
1879                                                ngx_http_spdy_state_settings);
1880            }
1881
1882            sc->entries--;
1883            sc->length -= NGX_SPDY_SETTINGS_PAIR_SIZE;
1884
1885            fid = ngx_spdy_frame_parse_uint32(pos);
1886
1887            pos += NGX_SPDY_SETTINGS_FID_SIZE;
1888
1889            val = ngx_spdy_frame_parse_uint32(pos);
1890
1891            pos += NGX_SPDY_SETTINGS_VAL_SIZE;
1892
```

```
        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                       "spdy SETTINGS entry fl:%ui id:%ui val:%ui",
                       ngx_spdy_frame_flags(fid), ngx_spdy_frame_id(fid), val);

        if (ngx_spdy_frame_flags(fid) == NGX_SPDY_SETTINGS_FLAG_PERSISTED) {
            continue;
        }

        switch (ngx_spdy_frame_id(fid)) {

        case NGX_SPDY_SETTINGS_INIT_WINDOW:

            if (val > NGX_SPDY_MAX_WINDOW) {
                ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
                              "client sent SETTINGS frame with "
                              "incorrect INIT_WINDOW value: %ui", val);

                return ngx_http_spdy_state_protocol_error(sc);
            }

            if (ngx_http_spdy_adjust_windows(sc, val - sc->init_window)
                != NGX_OK)
            {
                return ngx_http_spdy_state_internal_error(sc);
            }

            sc->init_window = val;

            continue;
        }
    }

    return ngx_http_spdy_state_complete(sc, pos, end);
}


static u_char *
ngx_http_spdy_state_complete(ngx_http_spdy_connection_t *sc, u_char *pos,
    u_char *end)
{
    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy frame complete pos:%p end:%p", pos, end);

    if (pos > end) {
        ngx_log_error(NGX_LOG_ALERT, sc->connection->log, 0,
                      "receive buffer overrun");

        ngx_debug_point();
        return ngx_http_spdy_state_internal_error(sc);
    }

    sc->handler = ngx_http_spdy_state_head;
    sc->stream = NULL;

    return pos;
}


static u_char *
ngx_http_spdy_state_save(ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end, ngx_http_spdy_handler_pt handler)
{
    size_t  size;

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy frame state save pos:%p end:%p handler:%p",
                   pos, end, handler);

    size = end - pos;

    if (size > NGX_SPDY_STATE_BUFFER_SIZE) {
        ngx_log_error(NGX_LOG_ALERT, sc->connection->log, 0,
                      "state buffer overflow: %uz bytes required", size);

        ngx_debug_point();
        return ngx_http_spdy_state_internal_error(sc);
```

```
1969      }
1970
1971      ngx_memcpy(sc->buffer, pos, NGX_SPDY_STATE_BUFFER_SIZE);
1972
1973      sc->buffer_used = size;
1974      sc->handler = handler;
1975      sc->incomplete = 1;
1976
1977      return end;
1978  }
1979
1980
1981  static u_char *
1982  ngx_http_spdy_state_inflate_error(ngx_http_spdy_connection_t *sc, int rc)
1983  {
1984      if (rc == Z_DATA_ERROR || rc == Z_STREAM_END) {
1985          ngx_log_error(NGX_LOG_INFO, sc->connection->log, 0,
1986                        "client sent SYN_STREAM frame with "
1987                        "corrupted header block, inflate() failed: %d", rc);
1988
1989          return ngx_http_spdy_state_protocol_error(sc);
1990      }
1991
1992      ngx_log_error(NGX_LOG_ERR, sc->connection->log, 0,
1993                    "inflate() failed: %d", rc);
1994
1995      return ngx_http_spdy_state_internal_error(sc);
1996  }
1997
1998
1999  static u_char *
2000  ngx_http_spdy_state_protocol_error(ngx_http_spdy_connection_t *sc)
2001  {
2002      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
2003                     "spdy state protocol error");
2004
2005      if (sc->stream) {
2006          sc->stream->out_closed = 1;
2007          ngx_http_spdy_close_stream(sc->stream, NGX_HTTP_BAD_REQUEST);
2008      }
2009
2010      ngx_http_spdy_finalize_connection(sc, NGX_HTTP_CLIENT_CLOSED_REQUEST);
2011
2012      return NULL;
2013  }
2014
2015
2016  static u_char *
2017  ngx_http_spdy_state_internal_error(ngx_http_spdy_connection_t *sc)
2018  {
2019      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
2020                     "spdy state internal error");
2021
2022      if (sc->stream) {
2023          sc->stream->out_closed = 1;
2024          ngx_http_spdy_close_stream(sc->stream, NGX_HTTP_INTERNAL_SERVER_ERROR);
2025      }
2026
2027      ngx_http_spdy_finalize_connection(sc, NGX_HTTP_INTERNAL_SERVER_ERROR);
2028
2029      return NULL;
2030  }
2031
2032
2033  static ngx_int_t
2034  ngx_http_spdy_send_window_update(ngx_http_spdy_connection_t *sc, ngx_uint_t sid,
2035      ngx_uint_t delta)
2036  {
2037      u_char                     *p;
2038      ngx_buf_t                  *buf;
2039      ngx_http_spdy_out_frame_t  *frame;
2040
2041      ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
2042                     "spdy send WINDOW_UPDATE sid:%ui delta:%ui", sid, delta);
2043
2044      frame = ngx_http_spdy_get_ctl_frame(sc, NGX_SPDY_WINDOW_UPDATE_SIZE,
```

```
                                       NGX_SPDY_HIGHEST_PRIORITY);
    if (frame == NULL) {
        return NGX_ERROR;
    }

    buf = frame->first->buf;

    p = buf->pos;

    p = ngx_spdy_frame_write_head(p, NGX_SPDY_WINDOW_UPDATE);
    p = ngx_spdy_frame_write_flags_and_len(p, 0, NGX_SPDY_WINDOW_UPDATE_SIZE);

    p = ngx_spdy_frame_write_sid(p, sid);
    p = ngx_spdy_frame_aligned_write_uint32(p, delta);

    buf->last = p;

    ngx_http_spdy_queue_frame(sc, frame);

    return NGX_OK;
}


static ngx_int_t
ngx_http_spdy_send_rst_stream(ngx_http_spdy_connection_t *sc, ngx_uint_t sid,
    ngx_uint_t status, ngx_uint_t priority)
{
    u_char                     *p;
    ngx_buf_t                  *buf;
    ngx_http_spdy_out_frame_t  *frame;

    if (sc->connection->error) {
        return NGX_OK;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy send RST_STREAM sid:%ui st:%ui", sid, status);

    frame = ngx_http_spdy_get_ctl_frame(sc, NGX_SPDY_RST_STREAM_SIZE,
                                        priority);
    if (frame == NULL) {
        return NGX_ERROR;
    }

    buf = frame->first->buf;

    p = buf->pos;

    p = ngx_spdy_frame_write_head(p, NGX_SPDY_RST_STREAM);
    p = ngx_spdy_frame_write_flags_and_len(p, 0, NGX_SPDY_RST_STREAM_SIZE);

    p = ngx_spdy_frame_write_sid(p, sid);
    p = ngx_spdy_frame_aligned_write_uint32(p, status);

    buf->last = p;

    ngx_http_spdy_queue_frame(sc, frame);

    return NGX_OK;
}


#if 0
static ngx_int_t
ngx_http_spdy_send_goaway(ngx_http_spdy_connection_t *sc)
{
    u_char                     *p;
    ngx_buf_t                  *buf;
    ngx_http_spdy_out_frame_t  *frame;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy send GOAWAY sid:%ui", sc->last_sid);

    frame = ngx_http_spdy_get_ctl_frame(sc, NGX_SPDY_GOAWAY_SIZE,
                                        NGX_SPDY_HIGHEST_PRIORITY);
    if (frame == NULL) {
```

```
        return NGX_ERROR;
    }

    buf = frame->first->buf;

    p = buf->pos;

    p = ngx_spdy_frame_write_head(p, NGX_SPDY_GOAWAY);
    p = ngx_spdy_frame_write_flags_and_len(p, 0, NGX_SPDY_GOAWAY_SIZE);

    p = ngx_spdy_frame_write_sid(p, sc->last_sid);

    buf->last = p;

    ngx_http_spdy_queue_frame(sc, frame);

    return NGX_OK;
}
#endif


static ngx_int_t
ngx_http_spdy_send_settings(ngx_http_spdy_connection_t *sc)
{
    u_char                     *p;
    ngx_buf_t                  *buf;
    ngx_chain_t                *cl;
    ngx_http_spdy_srv_conf_t   *sscf;
    ngx_http_spdy_out_frame_t  *frame;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy send SETTINGS frame");

    frame = ngx_palloc(sc->pool, sizeof(ngx_http_spdy_out_frame_t));
    if (frame == NULL) {
        return NGX_ERROR;
    }

    cl = ngx_alloc_chain_link(sc->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    buf = ngx_create_temp_buf(sc->pool, NGX_SPDY_FRAME_HEADER_SIZE
                                        + NGX_SPDY_SETTINGS_NUM_SIZE
                                        + 2 * NGX_SPDY_SETTINGS_PAIR_SIZE);
    if (buf == NULL) {
        return NGX_ERROR;
    }

    buf->last_buf = 1;

    cl->buf = buf;
    cl->next = NULL;

    frame->first = cl;
    frame->last = cl;
    frame->handler = ngx_http_spdy_settings_frame_handler;
    frame->stream = NULL;
#if (NGX_DEBUG)
    frame->length = NGX_SPDY_SETTINGS_NUM_SIZE
                    + 2 * NGX_SPDY_SETTINGS_PAIR_SIZE;
#endif
    frame->priority = NGX_SPDY_HIGHEST_PRIORITY;
    frame->blocked = 0;

    p = buf->pos;

    p = ngx_spdy_frame_write_head(p, NGX_SPDY_SETTINGS);
    p = ngx_spdy_frame_write_flags_and_len(p, NGX_SPDY_FLAG_CLEAR_SETTINGS,
                                           NGX_SPDY_SETTINGS_NUM_SIZE
                                           + 2 * NGX_SPDY_SETTINGS_PAIR_SIZE);

    p = ngx_spdy_frame_aligned_write_uint32(p, 2);

    sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
```

```
                                       ngx_http_spdy_module);

    p = ngx_spdy_frame_write_flags_and_id(p, 0, NGX_SPDY_SETTINGS_MAX_STREAMS);
    p = ngx_spdy_frame_aligned_write_uint32(p, sscf->concurrent_streams);

    p = ngx_spdy_frame_write_flags_and_id(p, 0, NGX_SPDY_SETTINGS_INIT_WINDOW);
    p = ngx_spdy_frame_aligned_write_uint32(p, NGX_SPDY_STREAM_WINDOW);

    buf->last = p;

    ngx_http_spdy_queue_frame(sc, frame);

    return NGX_OK;
}


ngx_int_t
ngx_http_spdy_settings_frame_handler(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_out_frame_t *frame)
{
    ngx_buf_t  *buf;

    buf = frame->first->buf;

    if (buf->pos != buf->last) {
        return NGX_AGAIN;
    }

    ngx_free_chain(sc->pool, frame->first);

    return NGX_OK;
}


static ngx_http_spdy_out_frame_t *
ngx_http_spdy_get_ctl_frame(ngx_http_spdy_connection_t *sc, size_t length,
    ngx_uint_t priority)
{
    ngx_chain_t               *cl;
    ngx_http_spdy_out_frame_t  *frame;

    frame = sc->free_ctl_frames;

    if (frame) {
        sc->free_ctl_frames = frame->next;

        cl = frame->first;
        cl->buf->pos = cl->buf->start;

    } else {
        frame = ngx_palloc(sc->pool, sizeof(ngx_http_spdy_out_frame_t));
        if (frame == NULL) {
            return NULL;
        }

        cl = ngx_alloc_chain_link(sc->pool);
        if (cl == NULL) {
            return NULL;
        }

        cl->buf = ngx_create_temp_buf(sc->pool,
                                      NGX_SPDY_CTL_FRAME_BUFFER_SIZE);
        if (cl->buf == NULL) {
            return NULL;
        }

        cl->buf->last_buf = 1;

        frame->first = cl;
        frame->last = cl;
        frame->handler = ngx_http_spdy_ctl_frame_handler;
        frame->stream = NULL;
    }

#if (NGX_DEBUG)
    if (length > NGX_SPDY_CTL_FRAME_BUFFER_SIZE - NGX_SPDY_FRAME_HEADER_SIZE) {
```

```
2273         ngx_log_error(NGX_LOG_ALERT, sc->pool->log, 0,
2274                       "requested control frame is too large: %uz", length);
2275         return NULL;
2276     }
2277
2278     frame->length = length;
2279 #endif
2280
2281     frame->priority = priority;
2282     frame->blocked = 0;
2283
2284     return frame;
2285 }
2286
2287
2288 static ngx_int_t
2289 ngx_http_spdy_ctl_frame_handler(ngx_http_spdy_connection_t *sc,
2290     ngx_http_spdy_out_frame_t *frame)
2291 {
2292     ngx_buf_t  *buf;
2293
2294     buf = frame->first->buf;
2295
2296     if (buf->pos != buf->last) {
2297         return NGX_AGAIN;
2298     }
2299
2300     frame->next = sc->free_ctl_frames;
2301     sc->free_ctl_frames = frame;
2302
2303     return NGX_OK;
2304 }
2305
2306
2307 static ngx_http_spdy_stream_t *
2308 ngx_http_spdy_create_stream(ngx_http_spdy_connection_t *sc, ngx_uint_t id,
2309     ngx_uint_t priority)
2310 {
2311     ngx_log_t                 *log;
2312     ngx_uint_t                 index;
2313     ngx_event_t               *rev, *wev;
2314     ngx_connection_t          *fc;
2315     ngx_http_log_ctx_t        *ctx;
2316     ngx_http_request_t        *r;
2317     ngx_http_spdy_stream_t    *stream;
2318     ngx_http_core_srv_conf_t  *cscf;
2319     ngx_http_spdy_srv_conf_t  *sscf;
2320
2321     fc = sc->free_fake_connections;
2322
2323     if (fc) {
2324         sc->free_fake_connections = fc->data;
2325
2326         rev = fc->read;
2327         wev = fc->write;
2328         log = fc->log;
2329         ctx = log->data;
2330
2331     } else {
2332         fc = ngx_palloc(sc->pool, sizeof(ngx_connection_t));
2333         if (fc == NULL) {
2334             return NULL;
2335         }
2336
2337         rev = ngx_palloc(sc->pool, sizeof(ngx_event_t));
2338         if (rev == NULL) {
2339             return NULL;
2340         }
2341
2342         wev = ngx_palloc(sc->pool, sizeof(ngx_event_t));
2343         if (wev == NULL) {
2344             return NULL;
2345         }
2346
2347         log = ngx_palloc(sc->pool, sizeof(ngx_log_t));
2348         if (log == NULL) {
```

```
            return NULL;
        }

        ctx = ngx_palloc(sc->pool, sizeof(ngx_http_log_ctx_t));
        if (ctx == NULL) {
            return NULL;
        }

        ctx->connection = fc;
        ctx->request = NULL;
    }

    ngx_memcpy(log, sc->connection->log, sizeof(ngx_log_t));

    log->data = ctx;

    ngx_memzero(rev, sizeof(ngx_event_t));

    rev->data = fc;
    rev->ready = 1;
    rev->handler = ngx_http_spdy_close_stream_handler;
    rev->log = log;

    ngx_memcpy(wev, rev, sizeof(ngx_event_t));

    wev->write = 1;

    ngx_memcpy(fc, sc->connection, sizeof(ngx_connection_t));

    fc->data = sc->http_connection;
    fc->read = rev;
    fc->write = wev;
    fc->sent = 0;
    fc->log = log;
    fc->buffered = 0;
    fc->sndlowat = 1;
    fc->tcp_nodelay = NGX_TCP_NODELAY_DISABLED;

    r = ngx_http_create_request(fc);
    if (r == NULL) {
        return NULL;
    }

    r->valid_location = 1;

    fc->data = r;
    sc->connection->requests++;

    cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);

    r->header_in = ngx_create_temp_buf(r->pool,
                                       cscf->client_header_buffer_size);
    if (r->header_in == NULL) {
        ngx_http_free_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return NULL;
    }

    r->headers_in.connection_type = NGX_HTTP_CONNECTION_CLOSE;

    stream = ngx_pcalloc(r->pool, sizeof(ngx_http_spdy_stream_t));
    if (stream == NULL) {
        ngx_http_free_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return NULL;
    }

    r->spdy_stream = stream;

    stream->id = id;
    stream->request = r;
    stream->connection = sc;

    stream->send_window = sc->init_window;
    stream->recv_window = NGX_SPDY_STREAM_WINDOW;

    stream->priority = priority;

```

```
    sscf = ngx_http_get_module_srv_conf(r, ngx_http_spdy_module);

    index = ngx_http_spdy_stream_index(sscf, id);

    stream->index = sc->streams_index[index];
    sc->streams_index[index] = stream;

    sc->processing++;

    return stream;
}


static ngx_http_spdy_stream_t *
ngx_http_spdy_get_stream_by_id(ngx_http_spdy_connection_t *sc,
    ngx_uint_t sid)
{
    ngx_http_spdy_stream_t    *stream;
    ngx_http_spdy_srv_conf_t  *sscf;

    sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
                                        ngx_http_spdy_module);

    stream = sc->streams_index[ngx_http_spdy_stream_index(sscf, sid)];

    while (stream) {
        if (stream->id == sid) {
            return stream;
        }

        stream = stream->index;
    }

    return NULL;
}


static ngx_int_t
ngx_http_spdy_parse_header(ngx_http_request_t *r)
{
    u_char                     *p, *end, ch;
    ngx_uint_t                  hash;
    ngx_http_core_srv_conf_t   *cscf;

    enum {
        sw_name_len = 0,
        sw_name,
        sw_value_len,
        sw_value
    } state;

    state = r->state;

    p = r->header_in->pos;
    end = r->header_in->last;

    switch (state) {

    case sw_name_len:

        if (end - p < NGX_SPDY_NV_NLEN_SIZE) {
            return NGX_AGAIN;
        }

        r->lowcase_index = ngx_spdy_frame_parse_uint32(p);

        if (r->lowcase_index == 0) {
            return NGX_ERROR;
        }

        /* null-terminate the previous header value */
        *p = '\0';

        p += NGX_SPDY_NV_NLEN_SIZE;

        r->invalid_header = 0;
```

```
2501
2502            state = sw_name;
2503
2504            /* fall through */
2505
2506        case sw_name:
2507
2508            if ((ngx_uint_t) (end - p) < r->lowcase_index) {
2509                break;
2510            }
2511
2512            cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
2513
2514            r->header_name_start = p;
2515            r->header_name_end = p + r->lowcase_index;
2516
2517            if (p[0] == ':') {
2518                p++;
2519            }
2520
2521            hash = 0;
2522
2523            for ( /* void */ ; p != r->header_name_end; p++) {
2524
2525                ch = *p;
2526
2527                hash = ngx_hash(hash, ch);
2528
2529                if ((ch >= 'a' && ch <= 'z')
2530                    || (ch == '-')
2531                    || (ch >= '0' && ch <= '9')
2532                    || (ch == '_' && cscf->underscores_in_headers))
2533                {
2534                    continue;
2535                }
2536
2537                switch (ch) {
2538                case '\0':
2539                case LF:
2540                case CR:
2541                case ':':
2542                    ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2543                                  "client sent invalid header name: \"%*s\"",
2544                                  r->lowcase_index, r->header_name_start);
2545
2546                    return NGX_HTTP_PARSE_INVALID_HEADER;
2547                }
2548
2549                if (ch >= 'A' && ch <= 'Z') {
2550                    return NGX_ERROR;
2551                }
2552
2553                r->invalid_header = 1;
2554            }
2555
2556            r->header_hash = hash;
2557
2558            state = sw_value_len;
2559
2560            /* fall through */
2561
2562        case sw_value_len:
2563
2564            if (end - p < NGX_SPDY_NV_VLEN_SIZE) {
2565                break;
2566            }
2567
2568            r->lowcase_index = ngx_spdy_frame_parse_uint32(p);
2569
2570            /* null-terminate header name */
2571            *p = '\0';
2572
2573            p += NGX_SPDY_NV_VLEN_SIZE;
2574
2575            state = sw_value;
2576
```

```
2577         /* fall through */
2578
2579     case sw_value:
2580
2581         if ((ngx_uint_t) (end - p) < r->lowcase_index) {
2582             break;
2583         }
2584
2585         r->header_start = p;
2586
2587         while (r->lowcase_index--) {
2588             ch = *p;
2589
2590             if (ch == '\0') {
2591
2592                 if (p == r->header_start) {
2593                     return NGX_ERROR;
2594                 }
2595
2596                 r->header_end = p;
2597                 r->header_in->pos = p + 1;
2598
2599                 r->state = sw_value;
2600
2601                 return NGX_OK;
2602             }
2603
2604             if (ch == CR || ch == LF) {
2605                 ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2606                               "client sent header \"%*s\" with "
2607                               "invalid value: \"%*s\\%c...\"",
2608                               r->header_name_end - r->header_name_start,
2609                               r->header_name_start,
2610                               p - r->header_start,
2611                               r->header_start,
2612                               ch == CR ? 'r' : 'n');
2613
2614                 return NGX_HTTP_PARSE_INVALID_HEADER;
2615             }
2616
2617             p++;
2618         }
2619
2620         r->header_end = p;
2621         r->header_in->pos = p;
2622
2623         r->state = 0;
2624
2625         return NGX_DONE;
2626     }
2627
2628     r->header_in->pos = p;
2629     r->state = state;
2630
2631     return NGX_AGAIN;
2632 }
2633
2634
2635 static ngx_int_t
2636 ngx_http_spdy_alloc_large_header_buffer(ngx_http_request_t *r)
2637 {
2638     u_char                    *old, *new, *p;
2639     size_t                     rest;
2640     ngx_buf_t                 *buf;
2641     ngx_http_spdy_stream_t    *stream;
2642     ngx_http_core_srv_conf_t  *cscf;
2643
2644     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2645                    "spdy alloc large header buffer");
2646
2647     stream = r->spdy_stream;
2648
2649     cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
2650
2651     if (stream->header_buffers
2652         == (ngx_uint_t) cscf->large_client_header_buffers.num)
```

```
2653     {
2654         ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2655                       "client sent too large request");
2656
2657         return NGX_DECLINED;
2658     }
2659
2660     rest = r->header_in->last - r->header_in->pos;
2661
2662     /*
2663      * One more byte is needed for null-termination
2664      * and another one for further progress.
2665      */
2666     if (rest > cscf->large_client_header_buffers.size - 2) {
2667         p = r->header_in->pos;
2668
2669         if (rest > NGX_MAX_ERROR_STR - 300) {
2670             rest = NGX_MAX_ERROR_STR - 300;
2671         }
2672
2673         ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2674                       "client sent too long header name or value: \"%*s...\"",
2675                       rest, p);
2676
2677         return NGX_DECLINED;
2678     }
2679
2680     buf = ngx_create_temp_buf(r->pool, cscf->large_client_header_buffers.size);
2681     if (buf == NULL) {
2682         return NGX_ERROR;
2683     }
2684
2685     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2686                    "spdy large header alloc: %p %uz",
2687                    buf->pos, buf->end - buf->last);
2688
2689     old = r->header_in->pos;
2690     new = buf->pos;
2691
2692     if (rest) {
2693         buf->last = ngx_cpymem(new, old, rest);
2694     }
2695
2696     r->header_in = buf;
2697
2698     stream->header_buffers++;
2699
2700     return NGX_OK;
2701 }
2702
2703
2704 static ngx_int_t
2705 ngx_http_spdy_handle_request_header(ngx_http_request_t *r)
2706 {
2707     ngx_uint_t                      i;
2708     ngx_table_elt_t                *h;
2709     ngx_http_core_srv_conf_t       *cscf;
2710     ngx_http_spdy_request_header_t  *sh;
2711
2712     if (r->invalid_header) {
2713         cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
2714
2715         if (cscf->ignore_invalid_headers) {
2716             ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2717                           "client sent invalid header: \"%*s\"",
2718                           r->header_end - r->header_name_start,
2719                           r->header_name_start);
2720             return NGX_OK;
2721         }
2722
2723     }
2724
2725     if (r->header_name_start[0] == ':') {
2726         r->header_name_start++;
2727
2728         for (i = 0; i < NGX_SPDY_REQUEST_HEADERS; i++) {
```

```
            sh = &ngx_http_spdy_request_headers[i];

            if (sh->hash != r->header_hash
                || sh->len != r->header_name_end - r->header_name_start
                || ngx_strncmp(sh->header, r->header_name_start, sh->len) != 0)
            {
                continue;
            }

            return sh->handler(r);
        }

        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "client sent invalid header name: \":%*s\"",
                      r->header_end - r->header_name_start,
                      r->header_name_start);

        return NGX_HTTP_PARSE_INVALID_HEADER;
    }

    h = ngx_list_push(&r->headers_in.headers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    h->hash = r->header_hash;

    h->key.len = r->header_name_end - r->header_name_start;
    h->key.data = r->header_name_start;

    h->value.len = r->header_end - r->header_start;
    h->value.data = r->header_start;

    h->lowcase_key = h->key.data;

    return NGX_OK;
}


void
ngx_http_spdy_request_headers_init(void)
{
    ngx_uint_t                      i;
    ngx_http_spdy_request_header_t  *h;

    for (i = 0; i < NGX_SPDY_REQUEST_HEADERS; i++) {
        h = &ngx_http_spdy_request_headers[i];
        h->hash = ngx_hash_key(h->header, h->len);
    }
}


static ngx_int_t
ngx_http_spdy_parse_method(ngx_http_request_t *r)
{
    size_t         k, len;
    ngx_uint_t     n;
    const u_char  *p, *m;

    /*
     * This array takes less than 256 sequential bytes,
     * and if typical CPU cache line size is 64 bytes,
     * it is prefetched for 4 load operations.
     */
    static const struct {
        u_char            len;
        const u_char      method[11];
        uint32_t          value;
    } tests[] = {
        { 3, "GET",       NGX_HTTP_GET },
        { 4, "POST",      NGX_HTTP_POST },
        { 4, "HEAD",      NGX_HTTP_HEAD },
        { 7, "OPTIONS",   NGX_HTTP_OPTIONS },
        { 8, "PROPFIND",  NGX_HTTP_PROPFIND },
        { 3, "PUT",       NGX_HTTP_PUT },
        { 5, "MKCOL",     NGX_HTTP_MKCOL },
```

```
        { 6, "DELETE",    NGX_HTTP_DELETE },
        { 4, "COPY",      NGX_HTTP_COPY },
        { 4, "MOVE",      NGX_HTTP_MOVE },
        { 9, "PROPPATCH", NGX_HTTP_PROPPATCH },
        { 4, "LOCK",      NGX_HTTP_LOCK },
        { 6, "UNLOCK",    NGX_HTTP_UNLOCK },
        { 5, "PATCH",     NGX_HTTP_PATCH },
        { 5, "TRACE",     NGX_HTTP_TRACE }
    }, *test;

    if (r->method_name.len) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "client sent duplicate :method header");

        return NGX_HTTP_PARSE_INVALID_HEADER;
    }

    len = r->header_end - r->header_start;

    r->method_name.len = len;
    r->method_name.data = r->header_start;

    test = tests;
    n = sizeof(tests) / sizeof(tests[0]);

    do {
        if (len == test->len) {
            p = r->method_name.data;
            m = test->method;
            k = len;

            do {
                if (*p++ != *m++) {
                    goto next;
                }
            } while (--k);

            r->method = test->value;
            return NGX_OK;
        }

    next:
        test++;

    } while (--n);

    p = r->method_name.data;

    do {
        if ((*p < 'A' || *p > 'Z') && *p != '_') {
            ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                          "client sent invalid method: \"%V\"",
                          &r->method_name);

            return NGX_HTTP_PARSE_INVALID_HEADER;
        }

        p++;

    } while (--len);

    return NGX_OK;
}


static ngx_int_t
ngx_http_spdy_parse_scheme(ngx_http_request_t *r)
{
    if (r->schema_start) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "client sent duplicate :schema header");

        return NGX_HTTP_PARSE_INVALID_HEADER;
    }

    r->schema_start = r->header_start;
```

```
2881      r->schema_end = r->header_end;
2882
2883      return NGX_OK;
2884  }
2885
2886
2887  static ngx_int_t
2888  ngx_http_spdy_parse_host(ngx_http_request_t *r)
2889  {
2890      ngx_table_elt_t  *h;
2891
2892      if (r->headers_in.host) {
2893          ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2894                        "client sent duplicate :host header");
2895
2896          return NGX_HTTP_PARSE_INVALID_HEADER;
2897      }
2898
2899      h = ngx_list_push(&r->headers_in.headers);
2900      if (h == NULL) {
2901          return NGX_ERROR;
2902      }
2903
2904      r->headers_in.host = h;
2905
2906      h->hash = r->header_hash;
2907
2908      h->key.len = r->header_name_end - r->header_name_start;
2909      h->key.data = r->header_name_start;
2910
2911      h->value.len = r->header_end - r->header_start;
2912      h->value.data = r->header_start;
2913
2914      h->lowcase_key = h->key.data;
2915
2916      return NGX_OK;
2917  }
2918
2919
2920  static ngx_int_t
2921  ngx_http_spdy_parse_path(ngx_http_request_t *r)
2922  {
2923      if (r->unparsed_uri.len) {
2924          ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2925                        "client sent duplicate :path header");
2926
2927          return NGX_HTTP_PARSE_INVALID_HEADER;
2928      }
2929
2930      r->uri_start = r->header_start;
2931      r->uri_end = r->header_end;
2932
2933      if (ngx_http_parse_uri(r) != NGX_OK) {
2934          ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2935                        "client sent invalid URI: \"%*s\"",
2936                        r->uri_end - r->uri_start, r->uri_start);
2937
2938          return NGX_HTTP_PARSE_INVALID_HEADER;
2939      }
2940
2941      if (ngx_http_process_request_uri(r) != NGX_OK) {
2942          /*
2943           * request has been finalized already
2944           * in ngx_http_process_request_uri()
2945           */
2946          return NGX_ABORT;
2947      }
2948
2949      return NGX_OK;
2950  }
2951
2952
2953  static ngx_int_t
2954  ngx_http_spdy_parse_version(ngx_http_request_t *r)
2955  {
2956      u_char  *p, ch;
```

```
2957
2958        if (r->http_protocol.len) {
2959            ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2960                          "client sent duplicate :version header");
2961
2962            return NGX_HTTP_PARSE_INVALID_HEADER;
2963        }
2964
2965        p = r->header_start;
2966
2967        if (r->header_end - p < 8 || !(ngx_str5cmp(p, 'H', 'T', 'T', 'P', '/'))) {
2968            goto invalid;
2969        }
2970
2971        ch = *(p + 5);
2972
2973        if (ch < '1' || ch > '9') {
2974            goto invalid;
2975        }
2976
2977        r->http_major = ch - '0';
2978
2979        for (p += 6; p != r->header_end - 2; p++) {
2980
2981            ch = *p;
2982
2983            if (ch == '.') {
2984                break;
2985            }
2986
2987            if (ch < '0' || ch > '9') {
2988                goto invalid;
2989            }
2990
2991            r->http_major = r->http_major * 10 + ch - '0';
2992        }
2993
2994        if (*p != '.') {
2995            goto invalid;
2996        }
2997
2998        ch = *(p + 1);
2999
3000        if (ch < '0' || ch > '9') {
3001            goto invalid;
3002        }
3003
3004        r->http_minor = ch - '0';
3005
3006        for (p += 2; p != r->header_end; p++) {
3007
3008            ch = *p;
3009
3010            if (ch < '0' || ch > '9') {
3011                goto invalid;
3012            }
3013
3014            r->http_minor = r->http_minor * 10 + ch - '0';
3015        }
3016
3017        r->http_protocol.len = r->header_end - r->header_start;
3018        r->http_protocol.data = r->header_start;
3019        r->http_version = r->http_major * 1000 + r->http_minor;
3020
3021        return NGX_OK;
3022
3023    invalid:
3024
3025        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
3026                      "client sent invalid http version: \"%*s\"",
3027                      r->header_end - r->header_start, r->header_start);
3028
3029        return NGX_HTTP_PARSE_INVALID_HEADER;
3030    }
3031
3032
```

```
3033 static ngx_int_t
3034 ngx_http_spdy_construct_request_line(ngx_http_request_t *r)
3035 {
3036     u_char  *p;
3037
3038     if (r->method_name.len == 0
3039         || r->unparsed_uri.len == 0
3040         || r->http_protocol.len == 0)
3041     {
3042         ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
3043         return NGX_ERROR;
3044     }
3045
3046     r->request_line.len = r->method_name.len + 1
3047                           + r->unparsed_uri.len + 1
3048                           + r->http_protocol.len;
3049
3050     p = ngx_pnalloc(r->pool, r->request_line.len + 1);
3051     if (p == NULL) {
3052         ngx_http_spdy_close_stream(r->spdy_stream,
3053                                    NGX_HTTP_INTERNAL_SERVER_ERROR);
3054         return NGX_ERROR;
3055     }
3056
3057     r->request_line.data = p;
3058
3059     p = ngx_cpymem(p, r->method_name.data, r->method_name.len);
3060
3061     *p++ = ' ';
3062
3063     p = ngx_cpymem(p, r->unparsed_uri.data, r->unparsed_uri.len);
3064
3065     *p++ = ' ';
3066
3067     ngx_memcpy(p, r->http_protocol.data, r->http_protocol.len + 1);
3068
3069     /* some modules expect the space character after method name */
3070     r->method_name.data = r->request_line.data;
3071
3072     return NGX_OK;
3073 }
3074
3075
3076 static void
3077 ngx_http_spdy_run_request(ngx_http_request_t *r)
3078 {
3079     ngx_uint_t                 i;
3080     ngx_list_part_t           *part;
3081     ngx_table_elt_t           *h;
3082     ngx_http_header_t         *hh;
3083     ngx_http_core_main_conf_t  *cmcf;
3084
3085     if (ngx_http_spdy_construct_request_line(r) != NGX_OK) {
3086         return;
3087     }
3088
3089     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3090                    "spdy http request line: \"%V\"", &r->request_line);
3091
3092     cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
3093
3094     part = &r->headers_in.headers.part;
3095     h = part->elts;
3096
3097     for (i = 0 ;; i++) {
3098
3099         if (i >= part->nelts) {
3100             if (part->next == NULL) {
3101                 break;
3102             }
3103
3104             part = part->next;
3105             h = part->elts;
3106             i = 0;
3107         }
3108
```

```
        hh = ngx_hash_find(&cmcf->headers_in_hash, h[i].hash,
                           h[i].lowcase_key, h[i].key.len);

        if (hh && hh->handler(r, &h[i], hh->offset) != NGX_OK) {
            return;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "spdy http header: \"%V: %V\"", &h[i].key, &h[i].value);
    }

    r->http_state = NGX_HTTP_PROCESS_REQUEST_STATE;

    if (ngx_http_process_request_header(r) != NGX_OK) {
        return;
    }

    if (r->headers_in.content_length_n > 0 && r->spdy_stream->in_closed) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "client prematurely closed stream");

        r->spdy_stream->skip_data = NGX_SPDY_DATA_ERROR;

        ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
        return;
    }

    ngx_http_process_request(r);
}


static ngx_int_t
ngx_http_spdy_init_request_body(ngx_http_request_t *r)
{
    ngx_buf_t                 *buf;
    ngx_temp_file_t           *tf;
    ngx_http_request_body_t   *rb;
    ngx_http_core_loc_conf_t  *clcf;

    rb = ngx_pcalloc(r->pool, sizeof(ngx_http_request_body_t));
    if (rb == NULL) {
        return NGX_ERROR;
    }

    r->request_body = rb;

    if (r->spdy_stream->in_closed) {
        return NGX_OK;
    }

    rb->rest = r->headers_in.content_length_n;

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (r->request_body_in_file_only
        || rb->rest > (off_t) clcf->client_body_buffer_size
        || rb->rest < 0)
    {
        tf = ngx_pcalloc(r->pool, sizeof(ngx_temp_file_t));
        if (tf == NULL) {
            return NGX_ERROR;
        }

        tf->file.fd = NGX_INVALID_FILE;
        tf->file.log = r->connection->log;
        tf->path = clcf->client_body_temp_path;
        tf->pool = r->pool;
        tf->warn = "a client request body is buffered to a temporary file";
        tf->log_level = r->request_body_file_log_level;
        tf->persistent = r->request_body_in_persistent_file;
        tf->clean = r->request_body_in_clean_file;

        if (r->request_body_file_group_access) {
            tf->access = 0660;
        }

```

```
3185            rb->temp_file = tf;
3186
3187            if (r->spdy_stream->in_closed
3188                && ngx_create_temp_file(&tf->file, tf->path, tf->pool,
3189                                        tf->persistent, tf->clean, tf->access)
3190                   != NGX_OK)
3191            {
3192                return NGX_ERROR;
3193            }
3194
3195            buf = ngx_calloc_buf(r->pool);
3196            if (buf == NULL) {
3197                return NGX_ERROR;
3198            }
3199
3200        } else {
3201
3202            if (rb->rest == 0) {
3203                return NGX_OK;
3204            }
3205
3206            buf = ngx_create_temp_buf(r->pool, (size_t) rb->rest);
3207            if (buf == NULL) {
3208                return NGX_ERROR;
3209            }
3210        }
3211
3212        rb->buf = buf;
3213
3214        rb->bufs = ngx_alloc_chain_link(r->pool);
3215        if (rb->bufs == NULL) {
3216            return NGX_ERROR;
3217        }
3218
3219        rb->bufs->buf = buf;
3220        rb->bufs->next = NULL;
3221
3222        rb->rest = 0;
3223
3224        return NGX_OK;
3225    }
3226
3227
3228    ngx_int_t
3229    ngx_http_spdy_read_request_body(ngx_http_request_t *r,
3230        ngx_http_client_body_handler_pt post_handler)
3231    {
3232        ngx_http_spdy_stream_t  *stream;
3233
3234        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3235                       "spdy read request body");
3236
3237        stream = r->spdy_stream;
3238
3239        switch (stream->skip_data) {
3240
3241        case NGX_SPDY_DATA_DISCARD:
3242            post_handler(r);
3243            return NGX_OK;
3244
3245        case NGX_SPDY_DATA_ERROR:
3246            if (r->headers_in.content_length_n == -1) {
3247                return NGX_HTTP_REQUEST_ENTITY_TOO_LARGE;
3248            } else {
3249                return NGX_HTTP_BAD_REQUEST;
3250            }
3251
3252        case NGX_SPDY_DATA_INTERNAL_ERROR:
3253            return NGX_HTTP_INTERNAL_SERVER_ERROR;
3254        }
3255
3256        if (!r->request_body && ngx_http_spdy_init_request_body(r) != NGX_OK) {
3257            stream->skip_data = NGX_SPDY_DATA_INTERNAL_ERROR;
3258            return NGX_HTTP_INTERNAL_SERVER_ERROR;
3259        }
3260
```

```
3261      if (stream->in_closed) {
3262          post_handler(r);
3263          return NGX_OK;
3264      }
3265
3266      r->request_body->post_handler = post_handler;
3267
3268      r->read_event_handler = ngx_http_test_reading;
3269      r->write_event_handler = ngx_http_request_empty_handler;
3270
3271      return NGX_AGAIN;
3272  }
3273
3274
3275  static ngx_int_t
3276  ngx_http_spdy_terminate_stream(ngx_http_spdy_connection_t *sc,
3277      ngx_http_spdy_stream_t *stream, ngx_uint_t status)
3278  {
3279      ngx_event_t       *rev;
3280      ngx_connection_t  *fc;
3281
3282      if (ngx_http_spdy_send_rst_stream(sc, stream->id, status,
3283                                        NGX_SPDY_HIGHEST_PRIORITY)
3284          == NGX_ERROR)
3285      {
3286          return NGX_ERROR;
3287      }
3288
3289      stream->out_closed = 1;
3290
3291      fc = stream->request->connection;
3292      fc->error = 1;
3293
3294      rev = fc->read;
3295      rev->handler(rev);
3296
3297      return NGX_OK;
3298  }
3299
3300
3301  static void
3302  ngx_http_spdy_close_stream_handler(ngx_event_t *ev)
3303  {
3304      ngx_connection_t    *fc;
3305      ngx_http_request_t  *r;
3306
3307      fc = ev->data;
3308      r = fc->data;
3309
3310      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3311                     "spdy close stream handler");
3312
3313      ngx_http_spdy_close_stream(r->spdy_stream, 0);
3314  }
3315
3316
3317  void
3318  ngx_http_spdy_close_stream(ngx_http_spdy_stream_t *stream, ngx_int_t rc)
3319  {
3320      int                           tcp_nodelay;
3321      ngx_event_t                  *ev;
3322      ngx_connection_t             *c, *fc;
3323      ngx_http_core_loc_conf_t     *clcf;
3324      ngx_http_spdy_stream_t      **index, *s;
3325      ngx_http_spdy_srv_conf_t     *sscf;
3326      ngx_http_spdy_connection_t   *sc;
3327
3328      sc = stream->connection;
3329
3330      ngx_log_debug3(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
3331                     "spdy close stream %ui, queued %ui, processing %ui",
3332                     stream->id, stream->queued, sc->processing);
3333
3334      fc = stream->request->connection;
3335
3336      if (stream->queued) {
```

```
3337        fc->write->handler = ngx_http_spdy_close_stream_handler;
3338        return;
3339    }
3340
3341    if (!stream->out_closed) {
3342        if (ngx_http_spdy_send_rst_stream(sc, stream->id,
3343                                          NGX_SPDY_INTERNAL_ERROR,
3344                                          stream->priority)
3345            != NGX_OK)
3346        {
3347            sc->connection->error = 1;
3348        }
3349
3350    } else {
3351        c = sc->connection;
3352
3353        if (c->tcp_nopush == NGX_TCP_NOPUSH_SET) {
3354            if (ngx_tcp_push(c->fd) == -1) {
3355                ngx_connection_error(c, ngx_socket_errno,
3356                                     ngx_tcp_push_n " failed");
3357                c->error = 1;
3358                tcp_nodelay = 0;
3359
3360            } else {
3361                c->tcp_nopush = NGX_TCP_NOPUSH_UNSET;
3362                tcp_nodelay = ngx_tcp_nodelay_and_tcp_nopush ? 1 : 0;
3363            }
3364
3365        } else {
3366            tcp_nodelay = 1;
3367        }
3368
3369        clcf = ngx_http_get_module_loc_conf(stream->request,
3370                                            ngx_http_core_module);
3371
3372        if (tcp_nodelay
3373            && clcf->tcp_nodelay
3374            && c->tcp_nodelay == NGX_TCP_NODELAY_UNSET)
3375        {
3376            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "tcp_nodelay");
3377
3378            if (setsockopt(c->fd, IPPROTO_TCP, TCP_NODELAY,
3379                           (const void *) &tcp_nodelay, sizeof(int))
3380                == -1)
3381            {
3382 #if (NGX_SOLARIS)
3383                /* Solaris returns EINVAL if a socket has been shut down */
3384                c->log_error = NGX_ERROR_IGNORE_EINVAL;
3385 #endif
3386
3387                ngx_connection_error(c, ngx_socket_errno,
3388                                     "setsockopt(TCP_NODELAY) failed");
3389
3390                c->log_error = NGX_ERROR_INFO;
3391                c->error = 1;
3392
3393            } else {
3394                c->tcp_nodelay = NGX_TCP_NODELAY_SET;
3395            }
3396        }
3397    }
3398
3399    if (sc->stream == stream) {
3400        sc->stream = NULL;
3401    }
3402
3403    sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
3404                                        ngx_http_spdy_module);
3405
3406    index = sc->streams_index + ngx_http_spdy_stream_index(sscf, stream->id);
3407
3408    for ( ;; ) {
3409        s = *index;
3410
3411        if (s == NULL) {
3412            break;
```

```
3413         }
3414
3415         if (s == stream) {
3416             *index = s->index;
3417             break;
3418         }
3419
3420         index = &s->index;
3421     }
3422
3423     ngx_http_free_request(stream->request, rc);
3424
3425     ev = fc->read;
3426
3427     if (ev->active || ev->disabled) {
3428         ngx_log_error(NGX_LOG_ALERT, sc->connection->log, 0,
3429                       "fake read event was activated");
3430     }
3431
3432     if (ev->timer_set) {
3433         ngx_del_timer(ev);
3434     }
3435
3436     if (ev->posted) {
3437         ngx_delete_posted_event(ev);
3438     }
3439
3440     ev = fc->write;
3441
3442     if (ev->active || ev->disabled) {
3443         ngx_log_error(NGX_LOG_ALERT, sc->connection->log, 0,
3444                       "fake write event was activated");
3445     }
3446
3447     if (ev->timer_set) {
3448         ngx_del_timer(ev);
3449     }
3450
3451     if (ev->posted) {
3452         ngx_delete_posted_event(ev);
3453     }
3454
3455     fc->data = sc->free_fake_connections;
3456     sc->free_fake_connections = fc;
3457
3458     sc->processing--;
3459
3460     if (sc->processing || sc->blocked) {
3461         return;
3462     }
3463
3464     ev = sc->connection->read;
3465
3466     ev->handler = ngx_http_spdy_handle_connection_handler;
3467     ngx_post_event(ev, &ngx_posted_events);
3468 }
3469
3470
3471 static void
3472 ngx_http_spdy_handle_connection_handler(ngx_event_t *rev)
3473 {
3474     ngx_connection_t  *c;
3475
3476     rev->handler = ngx_http_spdy_read_handler;
3477
3478     if (rev->ready) {
3479         ngx_http_spdy_read_handler(rev);
3480         return;
3481     }
3482
3483     c = rev->data;
3484
3485     ngx_http_spdy_handle_connection(c->data);
3486 }
3487
3488
```

```
3489 static void
3490 ngx_http_spdy_keepalive_handler(ngx_event_t *rev)
3491 {
3492     ngx_connection_t            *c;
3493     ngx_http_spdy_srv_conf_t    *sscf;
3494     ngx_http_spdy_connection_t  *sc;
3495
3496     c = rev->data;
3497
3498     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "spdy keepalive handler");
3499
3500     if (rev->timedout || c->close) {
3501         ngx_http_close_connection(c);
3502         return;
3503     }
3504
3505 #if (NGX_HAVE_KQUEUE)
3506
3507     if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
3508         if (rev->pending_eof) {
3509             c->log->handler = NULL;
3510             ngx_log_error(NGX_LOG_INFO, c->log, rev->kq_errno,
3511                           "kevent() reported that client %V closed "
3512                           "keepalive connection", &c->addr_text);
3513 #if (NGX_HTTP_SSL)
3514             if (c->ssl) {
3515                 c->ssl->no_send_shutdown = 1;
3516             }
3517 #endif
3518             ngx_http_close_connection(c);
3519             return;
3520         }
3521     }
3522
3523 #endif
3524
3525     c->destroyed = 0;
3526     c->idle = 0;
3527     ngx_reusable_connection(c, 0);
3528
3529     sc = c->data;
3530
3531     sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
3532                                         ngx_http_spdy_module);
3533
3534     sc->pool = ngx_create_pool(sscf->pool_size, sc->connection->log);
3535     if (sc->pool == NULL) {
3536         ngx_http_close_connection(c);
3537         return;
3538     }
3539
3540     sc->streams_index = ngx_pcalloc(sc->pool,
3541                                     ngx_http_spdy_streams_index_size(sscf)
3542                                     * sizeof(ngx_http_spdy_stream_t *));
3543     if (sc->streams_index == NULL) {
3544         ngx_http_close_connection(c);
3545         return;
3546     }
3547
3548     c->write->handler = ngx_http_spdy_write_handler;
3549
3550     rev->handler = ngx_http_spdy_read_handler;
3551     ngx_http_spdy_read_handler(rev);
3552 }
3553
3554
3555 static void
3556 ngx_http_spdy_finalize_connection(ngx_http_spdy_connection_t *sc,
3557     ngx_int_t rc)
3558 {
3559     ngx_uint_t                 i, size;
3560     ngx_event_t               *ev;
3561     ngx_connection_t          *c, *fc;
3562     ngx_http_request_t        *r;
3563     ngx_http_spdy_stream_t    *stream;
3564     ngx_http_spdy_srv_conf_t  *sscf;
```

```
3565
3566        c = sc->connection;
3567
3568        if (!sc->processing) {
3569            ngx_http_close_connection(c);
3570            return;
3571        }
3572
3573        c->error = 1;
3574        c->read->handler = ngx_http_empty_handler;
3575        c->write->handler = ngx_http_empty_handler;
3576
3577        sc->last_out = NULL;
3578
3579        sc->blocked = 1;
3580
3581        sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
3582                                            ngx_http_spdy_module);
3583
3584        size = ngx_http_spdy_streams_index_size(sscf);
3585
3586        for (i = 0; i < size; i++) {
3587            stream = sc->streams_index[i];
3588
3589            while (stream) {
3590                stream->handled = 0;
3591
3592                r = stream->request;
3593                fc = r->connection;
3594
3595                fc->error = 1;
3596
3597                if (stream->queued) {
3598                    stream->queued = 0;
3599
3600                    ev = fc->write;
3601                    ev->delayed = 0;
3602
3603                } else {
3604                    ev = fc->read;
3605                }
3606
3607                stream = stream->index;
3608
3609                ev->eof = 1;
3610                ev->handler(ev);
3611            }
3612        }
3613
3614        sc->blocked = 0;
3615
3616        if (sc->processing) {
3617            return;
3618        }
3619
3620        ngx_http_close_connection(c);
3621    }
3622
3623
3624    static ngx_int_t
3625    ngx_http_spdy_adjust_windows(ngx_http_spdy_connection_t *sc, ssize_t delta)
3626    {
3627        ngx_uint_t                 i, size;
3628        ngx_event_t               *wev;
3629        ngx_http_spdy_stream_t    *stream, *sn;
3630        ngx_http_spdy_srv_conf_t  *sscf;
3631
3632        sscf = ngx_http_get_module_srv_conf(sc->http_connection->conf_ctx,
3633                                            ngx_http_spdy_module);
3634
3635        size = ngx_http_spdy_streams_index_size(sscf);
3636
3637        for (i = 0; i < size; i++) {
3638
3639            for (stream = sc->streams_index[i]; stream; stream = sn) {
3640                sn = stream->index;
```

```
3641
3642            if (delta > 0
3643                && stream->send_window
3644                     > (ssize_t) (NGX_SPDY_MAX_WINDOW - delta))
3645            {
3646                if (ngx_http_spdy_terminate_stream(sc, stream,
3647                                                   NGX_SPDY_FLOW_CONTROL_ERROR)
3648                    == NGX_ERROR)
3649                {
3650                    return NGX_ERROR;
3651                }
3652
3653                continue;
3654            }
3655
3656            stream->send_window += delta;
3657
3658            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
3659                           "spdy:%ui adjust window:%z",
3660                           stream->id, stream->send_window);
3661
3662            if (stream->send_window > 0 && stream->exhausted) {
3663                stream->exhausted = 0;
3664
3665                wev = stream->request->connection->write;
3666
3667                if (!wev->timer_set) {
3668                    wev->delayed = 0;
3669                    wev->handler(wev);
3670                }
3671            }
3672        }
3673    }
3674
3675    return NGX_OK;
3676 }
3677
3678
3679 static void
3680 ngx_http_spdy_pool_cleanup(void *data)
3681 {
3682     ngx_http_spdy_connection_t  *sc = data;
3683
3684     if (sc->pool) {
3685         ngx_destroy_pool(sc->pool);
3686     }
3687 }
3688
3689
3690 static void *
3691 ngx_http_spdy_zalloc(void *opaque, u_int items, u_int size)
3692 {
3693     ngx_http_spdy_connection_t *sc = opaque;
3694
3695     return ngx_palloc(sc->connection->pool, items * size);
3696 }
3697
3698
3699 static void
3700 ngx_http_spdy_zfree(void *opaque, void *address)
3701 {
3702 #if 0
3703     ngx_http_spdy_connection_t *sc = opaque;
3704
3705     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
3706                    "spdy zfree: %p", address);
3707 #endif
3708 }
```

One Level Up Top Level

src/http/ngx_http_spdy_module.c - nginx-1.7.10

Global variables defined

- ngx_http_spdy_chunk_size_post
- ngx_http_spdy_commands
- ngx_http_spdy_headers_comp_bounds
- ngx_http_spdy_module
- ngx_http_spdy_module_ctx
- ngx_http_spdy_pool_size_post
- ngx_http_spdy_recv_buffer_size_post
- ngx_http_spdy_streams_index_mask_post
- ngx_http_spdy_vars

Functions defined

- ngx_http_spdy_add_variables
- ngx_http_spdy_chunk_size
- ngx_http_spdy_create_loc_conf
- ngx_http_spdy_create_main_conf
- ngx_http_spdy_create_srv_conf
- ngx_http_spdy_init_main_conf
- ngx_http_spdy_merge_loc_conf
- ngx_http_spdy_merge_srv_conf
- ngx_http_spdy_module_init
- ngx_http_spdy_pool_size
- ngx_http_spdy_recv_buffer_size
- ngx_http_spdy_request_priority_variable
- ngx_http_spdy_streams_index_mask
- ngx_http_spdy_variable

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  * Copyright (C) Valentin V. Bartenev
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
```

```
#include <ngx_http.h>
#include <ngx_http_spdy_module.h>


static ngx_int_t ngx_http_spdy_add_variables(ngx_conf_t *cf);

static ngx_int_t ngx_http_spdy_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_spdy_request_priority_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);

static ngx_int_t ngx_http_spdy_module_init(ngx_cycle_t *cycle);

static void *ngx_http_spdy_create_main_conf(ngx_conf_t *cf);
static char *ngx_http_spdy_init_main_conf(ngx_conf_t *cf, void *conf);
static void *ngx_http_spdy_create_srv_conf(ngx_conf_t *cf);
static char *ngx_http_spdy_merge_srv_conf(ngx_conf_t *cf, void *parent,
    void *child);
static void *ngx_http_spdy_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_spdy_merge_loc_conf(ngx_conf_t *cf, void *parent,
    void *child);

static char *ngx_http_spdy_recv_buffer_size(ngx_conf_t *cf, void *post,
    void *data);
static char *ngx_http_spdy_pool_size(ngx_conf_t *cf, void *post, void *data);
static char *ngx_http_spdy_streams_index_mask(ngx_conf_t *cf, void *post,
    void *data);
static char *ngx_http_spdy_chunk_size(ngx_conf_t *cf, void *post, void *data);


static ngx_conf_num_bounds_t  ngx_http_spdy_headers_comp_bounds = {
    ngx_conf_check_num_bounds, 0, 9
};

static ngx_conf_post_t  ngx_http_spdy_recv_buffer_size_post =
    { ngx_http_spdy_recv_buffer_size };
static ngx_conf_post_t  ngx_http_spdy_pool_size_post =
    { ngx_http_spdy_pool_size };
static ngx_conf_post_t  ngx_http_spdy_streams_index_mask_post =
    { ngx_http_spdy_streams_index_mask };
static ngx_conf_post_t  ngx_http_spdy_chunk_size_post =
    { ngx_http_spdy_chunk_size };


static ngx_command_t  ngx_http_spdy_commands[] = {

    { ngx_string("spdy_recv_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_spdy_main_conf_t, recv_buffer_size),
      &ngx_http_spdy_recv_buffer_size_post },

    { ngx_string("spdy_pool_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_spdy_srv_conf_t, pool_size),
      &ngx_http_spdy_pool_size_post },

    { ngx_string("spdy_max_concurrent_streams"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_spdy_srv_conf_t, concurrent_streams),
      NULL },

    { ngx_string("spdy_streams_index_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_spdy_srv_conf_t, streams_index_mask),
      &ngx_http_spdy_streams_index_mask_post },

    { ngx_string("spdy_recv_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
```

```
      ngx_conf_set_msec_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_spdy_srv_conf_t, recv_timeout),
      NULL },

    { ngx_string("spdy_keepalive_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_spdy_srv_conf_t, keepalive_timeout),
      NULL },

    { ngx_string("spdy_headers_comp"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_spdy_srv_conf_t, headers_comp),
      &ngx_http_spdy_headers_comp_bounds },

     { ngx_string("spdy_chunk_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_spdy_loc_conf_t, chunk_size),
      &ngx_http_spdy_chunk_size_post },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_spdy_module_ctx = {
    ngx_http_spdy_add_variables,           /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_spdy_create_main_conf,        /* create main configuration */
    ngx_http_spdy_init_main_conf,          /* init main configuration */

    ngx_http_spdy_create_srv_conf,         /* create server configuration */
    ngx_http_spdy_merge_srv_conf,          /* merge server configuration */

    ngx_http_spdy_create_loc_conf,         /* create location configuration */
    ngx_http_spdy_merge_loc_conf           /* merge location configuration */
};


ngx_module_t  ngx_http_spdy_module = {
    NGX_MODULE_V1,
    &ngx_http_spdy_module_ctx,             /* module context */
    ngx_http_spdy_commands,                /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    ngx_http_spdy_module_init,             /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_variable_t  ngx_http_spdy_vars[] = {

    { ngx_string("spdy"), NULL,
      ngx_http_spdy_variable, 0, 0, 0 },

    { ngx_string("spdy_request_priority"), NULL,
      ngx_http_spdy_request_priority_variable, 0, 0, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static ngx_int_t
ngx_http_spdy_add_variables(ngx_conf_t *cf)
{
```

```
    ngx_http_variable_t  *var, *v;

    for (v = ngx_http_spdy_vars; v->name.len; v++) {
        var = ngx_http_add_variable(cf, &v->name, v->flags);
        if (var == NULL) {
            return NGX_ERROR;
        }

        var->get_handler = v->get_handler;
        var->data = v->data;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_spdy_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    if (r->spdy_stream) {
        v->len = sizeof("3.1") - 1;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = (u_char *) "3.1";

        return NGX_OK;
    }

    *v = ngx_http_variable_null_value;

    return NGX_OK;
}


static ngx_int_t
ngx_http_spdy_request_priority_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    if (r->spdy_stream) {
        v->len = 1;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;

        v->data = ngx_pnalloc(r->pool, 1);
        if (v->data == NULL) {
            return NGX_ERROR;
        }

        v->data[0] = '0' + (u_char) r->spdy_stream->priority;

        return NGX_OK;
    }

    *v = ngx_http_variable_null_value;

    return NGX_OK;
}


static ngx_int_t
ngx_http_spdy_module_init(ngx_cycle_t *cycle)
{
    ngx_http_spdy_request_headers_init();

    return NGX_OK;
}


static void *
ngx_http_spdy_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_spdy_main_conf_t  *smcf;

```

```
    smcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_spdy_main_conf_t));
    if (smcf == NULL) {
        return NULL;
    }

    smcf->recv_buffer_size = NGX_CONF_UNSET_SIZE;

    return smcf;
}


static char *
ngx_http_spdy_init_main_conf(ngx_conf_t *cf, void *conf)
{
    ngx_http_spdy_main_conf_t *smcf = conf;

    ngx_conf_init_size_value(smcf->recv_buffer_size, 256 * 1024);

    return NGX_CONF_OK;
}


static void *
ngx_http_spdy_create_srv_conf(ngx_conf_t *cf)
{
    ngx_http_spdy_srv_conf_t  *sscf;

    sscf = ngx_pcalloc(cf->pool, sizeof(ngx_http_spdy_srv_conf_t));
    if (sscf == NULL) {
        return NULL;
    }

    sscf->pool_size = NGX_CONF_UNSET_SIZE;

    sscf->concurrent_streams = NGX_CONF_UNSET_UINT;
    sscf->streams_index_mask = NGX_CONF_UNSET_UINT;

    sscf->recv_timeout = NGX_CONF_UNSET_MSEC;
    sscf->keepalive_timeout = NGX_CONF_UNSET_MSEC;

    sscf->headers_comp = NGX_CONF_UNSET;

    return sscf;
}


static char *
ngx_http_spdy_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_spdy_srv_conf_t *prev = parent;
    ngx_http_spdy_srv_conf_t *conf = child;

    ngx_conf_merge_size_value(conf->pool_size, prev->pool_size, 4096);

    ngx_conf_merge_uint_value(conf->concurrent_streams,
                              prev->concurrent_streams, 100);

    ngx_conf_merge_uint_value(conf->streams_index_mask,
                              prev->streams_index_mask, 32 - 1);

    ngx_conf_merge_msec_value(conf->recv_timeout,
                              prev->recv_timeout, 30000);
    ngx_conf_merge_msec_value(conf->keepalive_timeout,
                              prev->keepalive_timeout, 180000);

    ngx_conf_merge_value(conf->headers_comp, prev->headers_comp, 0);

    return NGX_CONF_OK;
}


static void *
ngx_http_spdy_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_spdy_loc_conf_t  *slcf;

```

```
    slcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_spdy_loc_conf_t));
    if (slcf == NULL) {
        return NULL;
    }

    slcf->chunk_size = NGX_CONF_UNSET_SIZE;

    return slcf;
}


static char *
ngx_http_spdy_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_spdy_loc_conf_t *prev = parent;
    ngx_http_spdy_loc_conf_t *conf = child;

    ngx_conf_merge_size_value(conf->chunk_size, prev->chunk_size, 8 * 1024);

    return NGX_CONF_OK;
}


static char *
ngx_http_spdy_recv_buffer_size(ngx_conf_t *cf, void *post, void *data)
{
    size_t *sp = data;

    if (*sp <= 2 * NGX_SPDY_STATE_BUFFER_SIZE) {
        return "value is too small";
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_spdy_pool_size(ngx_conf_t *cf, void *post, void *data)
{
    size_t *sp = data;

    if (*sp < NGX_MIN_POOL_SIZE) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "the pool size must be no less than %uz",
                           NGX_MIN_POOL_SIZE);
        return NGX_CONF_ERROR;
    }

    if (*sp % NGX_POOL_ALIGNMENT) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "the pool size must be a multiple of %uz",
                           NGX_POOL_ALIGNMENT);
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_spdy_streams_index_mask(ngx_conf_t *cf, void *post, void *data)
{
    ngx_uint_t *np = data;

    ngx_uint_t  mask;

    mask = *np - 1;

    if (*np == 0 || (*np & mask)) {
        return "must be a power of two";
    }

    *np = mask;

    return NGX_CONF_OK;
}
```

```
390
391
392 static char *
393 ngx_http_spdy_chunk_size(ngx_conf_t *cf, void *post, void *data)
394 {
395     size_t *sp = data;
396
397     if (*sp == 0) {
398         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
399                            "the spdy chunk size cannot be zero");
400         return NGX_CONF_ERROR;
401     }
402
403     if (*sp > NGX_SPDY_MAX_FRAME_SIZE) {
404         *sp = NGX_SPDY_MAX_FRAME_SIZE;
405     }
406
407     return NGX_CONF_OK;
408 }
```

One Level Up Top Level

src/http/ngx_http_config.h - nginx-1.7.10

Data types defined

- ngx_http_conf_ctx_t
- ngx_http_module_t

Macros defined

- NGX_HTTP_LIF_CONF
- NGX_HTTP_LMT_CONF
- NGX_HTTP_LOC_CONF
- NGX_HTTP_LOC_CONF_OFFSET
- NGX_HTTP_MAIN_CONF
- NGX_HTTP_MAIN_CONF_OFFSET
- NGX_HTTP_MODULE
- NGX_HTTP_SIF_CONF
- NGX_HTTP_SRV_CONF
- NGX_HTTP_SRV_CONF_OFFSET
- NGX_HTTP_UPS_CONF
- _NGX_HTTP_CONFIG_H_INCLUDED_
- ngx_http_conf_get_module_loc_conf
- ngx_http_conf_get_module_main_conf
- ngx_http_conf_get_module_srv_conf
- ngx_http_cycle_get_module_main_conf
- ngx_http_get_module_loc_conf
- ngx_http_get_module_main_conf
- ngx_http_get_module_srv_conf

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_HTTP_CONFIG_H_INCLUDED_
9  #define _NGX_HTTP_CONFIG_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
```

```
15
16
17  typedef struct {
18      void        **main_conf;
19      void        **srv_conf;
20      void        **loc_conf;
21  } ngx_http_conf_ctx_t;
22
23
24  typedef struct {
25      ngx_int_t   (*preconfiguration)(ngx_conf_t *cf);
26      ngx_int_t   (*postconfiguration)(ngx_conf_t *cf);
27
28      void       *(*create_main_conf)(ngx_conf_t *cf);
29      char       *(*init_main_conf)(ngx_conf_t *cf, void *conf);
30
31      void       *(*create_srv_conf)(ngx_conf_t *cf);
32      char       *(*merge_srv_conf)(ngx_conf_t *cf, void *prev, void *conf);
33
34      void       *(*create_loc_conf)(ngx_conf_t *cf);
35      char       *(*merge_loc_conf)(ngx_conf_t *cf, void *prev, void *conf);
36  } ngx_http_module_t;
37
38
39  #define NGX_HTTP_MODULE           0x50545448   /* "HTTP" */
40
41  #define NGX_HTTP_MAIN_CONF        0x02000000
42  #define NGX_HTTP_SRV_CONF         0x04000000
43  #define NGX_HTTP_LOC_CONF         0x08000000
44  #define NGX_HTTP_UPS_CONF         0x10000000
45  #define NGX_HTTP_SIF_CONF         0x20000000
46  #define NGX_HTTP_LIF_CONF         0x40000000
47  #define NGX_HTTP_LMT_CONF         0x80000000
48
49
50  #define NGX_HTTP_MAIN_CONF_OFFSET  offsetof(ngx_http_conf_ctx_t, main_conf)
51  #define NGX_HTTP_SRV_CONF_OFFSET   offsetof(ngx_http_conf_ctx_t, srv_conf)
52  #define NGX_HTTP_LOC_CONF_OFFSET   offsetof(ngx_http_conf_ctx_t, loc_conf)
53
54
55  #define ngx_http_get_module_main_conf(r, module)                             \
56      (r)->main_conf[module.ctx_index]
57  #define ngx_http_get_module_srv_conf(r, module)  (r)->srv_conf[module.ctx_index]
58  #define ngx_http_get_module_loc_conf(r, module)  (r)->loc_conf[module.ctx_index]
59
60
61  #define ngx_http_conf_get_module_main_conf(cf, module)                       \
62      ((ngx_http_conf_ctx_t *) cf->ctx)->main_conf[module.ctx_index]
63  #define ngx_http_conf_get_module_srv_conf(cf, module)                        \
64      ((ngx_http_conf_ctx_t *) cf->ctx)->srv_conf[module.ctx_index]
65  #define ngx_http_conf_get_module_loc_conf(cf, module)                        \
66      ((ngx_http_conf_ctx_t *) cf->ctx)->loc_conf[module.ctx_index]
67
68  #define ngx_http_cycle_get_module_main_conf(cycle, module)                   \
69      (cycle->conf_ctx[ngx_http_module.index] ?                                \
70          ((ngx_http_conf_ctx_t *) cycle->conf_ctx[ngx_http_module.index])     \
71              ->main_conf[module.ctx_index]:                                   \
72          NULL)
73
74
75  #endif /* _NGX_HTTP_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_spdy_module.h - nginx-1.7.10

Data types defined

- ngx_http_spdy_loc_conf_t
- ngx_http_spdy_main_conf_t
- ngx_http_spdy_srv_conf_t

Macros defined

- _NGX_HTTP_SPDY_MODULE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  * Copyright (C) Valentin V. Bartenev
5  */
6
7
8  #ifndef _NGX_HTTP_SPDY_MODULE_H_INCLUDED_
9  #define _NGX_HTTP_SPDY_MODULE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
17  typedef struct {
18      size_t                          recv_buffer_size;
19      u_char                         *recv_buffer;
20  } ngx_http_spdy_main_conf_t;
21
22
23  typedef struct {
24      size_t                          pool_size;
25      ngx_uint_t                      concurrent_streams;
26      ngx_uint_t                      streams_index_mask;
27      ngx_msec_t                      recv_timeout;
28      ngx_msec_t                      keepalive_timeout;
29      ngx_int_t                       headers_comp;
30  } ngx_http_spdy_srv_conf_t;
31
32
33  typedef struct {
34      size_t                          chunk_size;
35  } ngx_http_spdy_loc_conf_t;
36
37
38  extern ngx_module_t  ngx_http_spdy_module;
39
40
41  #endif /* _NGX_HTTP_SPDY_MODULE_H_INCLUDED_ */
```

src/http/ngx_http_spdy.h - nginx-1.7.10

Data types defined

- ngx_http_spdy_connection_s
- ngx_http_spdy_connection_t
- ngx_http_spdy_handler_pt
- ngx_http_spdy_out_frame_s
- ngx_http_spdy_out_frame_t
- ngx_http_spdy_stream_s

Functions defined

- ngx_http_spdy_queue_blocked_frame
- ngx_http_spdy_queue_frame

Macros defined

- NGX_SPDY_CTL_BIT
- NGX_SPDY_DATA_DISCARD
- NGX_SPDY_DATA_ERROR
- NGX_SPDY_DATA_INTERNAL_ERROR
- NGX_SPDY_DELTA_SIZE
- NGX_SPDY_FLAG_CLEAR_SETTINGS
- NGX_SPDY_FLAG_FIN
- NGX_SPDY_FLAG_UNIDIRECTIONAL
- NGX_SPDY_FRAME_HEADER_SIZE
- NGX_SPDY_GOAWAY
- NGX_SPDY_GOAWAY_SIZE
- NGX_SPDY_HEADERS
- NGX_SPDY_HIGHEST_PRIORITY
- NGX_SPDY_LOWEST_PRIORITY
- NGX_SPDY_MAX_FRAME_SIZE
- NGX_SPDY_NPN_ADVERTISE
- NGX_SPDY_NPN_NEGOTIATED
- NGX_SPDY_NV_NLEN_SIZE
- NGX_SPDY_NV_NUM_SIZE

- NGX_SPDY_NV_VLEN_SIZE
- NGX_SPDY_PING
- NGX_SPDY_PING_SIZE
- NGX_SPDY_RST_STREAM
- NGX_SPDY_RST_STREAM_SIZE
- NGX_SPDY_SETTINGS
- NGX_SPDY_SETTINGS_FID_SIZE
- NGX_SPDY_SETTINGS_NUM_SIZE
- NGX_SPDY_SETTINGS_PAIR_SIZE
- NGX_SPDY_SETTINGS_VAL_SIZE
- NGX_SPDY_SID_SIZE
- NGX_SPDY_STATE_BUFFER_SIZE
- NGX_SPDY_SYN_REPLY
- NGX_SPDY_SYN_REPLY_SIZE
- NGX_SPDY_SYN_STREAM
- NGX_SPDY_SYN_STREAM_SIZE
- NGX_SPDY_VERSION
- NGX_SPDY_WINDOW_UPDATE
- NGX_SPDY_WINDOW_UPDATE_SIZE
- _NGX_HTTP_SPDY_H_INCLUDED_
- ngx_spdy_ctl_frame_head
- ngx_spdy_frame_aligned_write_uint16
- ngx_spdy_frame_aligned_write_uint32
- ngx_spdy_frame_write_flags_and_id
- ngx_spdy_frame_write_flags_and_len
- ngx_spdy_frame_write_head
- ngx_spdy_frame_write_sid
- ngx_spdy_frame_write_uint16
- ngx_spdy_frame_write_uint16
- ngx_spdy_frame_write_uint32
- ngx_spdy_frame_write_uint32
- ngx_spdy_frame_write_window

Source code

```c
/*
 * Copyright (C) Nginx, Inc.
 * Copyright (C) Valentin V. Bartenev
 */


#ifndef _NGX_HTTP_SPDY_H_INCLUDED_
#define _NGX_HTTP_SPDY_H_INCLUDED_


#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_http.h>

#include <zlib.h>


#define NGX_SPDY_VERSION              3

#define NGX_SPDY_NPN_ADVERTISE        "\x08spdy/3.1"
#define NGX_SPDY_NPN_NEGOTIATED       "spdy/3.1"

#define NGX_SPDY_STATE_BUFFER_SIZE    16

#define NGX_SPDY_CTL_BIT              1

#define NGX_SPDY_SYN_STREAM           1
#define NGX_SPDY_SYN_REPLY            2
#define NGX_SPDY_RST_STREAM           3
#define NGX_SPDY_SETTINGS             4
#define NGX_SPDY_PING                 6
#define NGX_SPDY_GOAWAY               7
#define NGX_SPDY_HEADERS              8
#define NGX_SPDY_WINDOW_UPDATE        9

#define NGX_SPDY_FRAME_HEADER_SIZE    8

#define NGX_SPDY_SID_SIZE             4
#define NGX_SPDY_DELTA_SIZE           4

#define NGX_SPDY_SYN_STREAM_SIZE      10
#define NGX_SPDY_SYN_REPLY_SIZE       4
#define NGX_SPDY_RST_STREAM_SIZE      8
#define NGX_SPDY_PING_SIZE            4
#define NGX_SPDY_GOAWAY_SIZE          8
#define NGX_SPDY_WINDOW_UPDATE_SIZE   8
#define NGX_SPDY_NV_NUM_SIZE          4
#define NGX_SPDY_NV_NLEN_SIZE         4
#define NGX_SPDY_NV_VLEN_SIZE         4
#define NGX_SPDY_SETTINGS_NUM_SIZE    4
#define NGX_SPDY_SETTINGS_FID_SIZE    4
#define NGX_SPDY_SETTINGS_VAL_SIZE    4

#define NGX_SPDY_SETTINGS_PAIR_SIZE                                           \
    (NGX_SPDY_SETTINGS_FID_SIZE + NGX_SPDY_SETTINGS_VAL_SIZE)

#define NGX_SPDY_HIGHEST_PRIORITY     0
#define NGX_SPDY_LOWEST_PRIORITY      7

#define NGX_SPDY_FLAG_FIN             0x01
#define NGX_SPDY_FLAG_UNIDIRECTIONAL  0x02
#define NGX_SPDY_FLAG_CLEAR_SETTINGS  0x01

#define NGX_SPDY_MAX_FRAME_SIZE       ((1 << 24) - 1)

#define NGX_SPDY_DATA_DISCARD         1
#define NGX_SPDY_DATA_ERROR           2
#define NGX_SPDY_DATA_INTERNAL_ERROR  3


typedef struct ngx_http_spdy_connection_s   ngx_http_spdy_connection_t;
typedef struct ngx_http_spdy_out_frame_s    ngx_http_spdy_out_frame_t;


typedef u_char *(*ngx_http_spdy_handler_pt) (ngx_http_spdy_connection_t *sc,
    u_char *pos, u_char *end);
```

```
77
78  struct ngx_http_spdy_connection_s {
79      ngx_connection_t               *connection;
80      ngx_http_connection_t          *http_connection;
81
82      ngx_uint_t                      processing;
83
84      size_t                          send_window;
85      size_t                          recv_window;
86      size_t                          init_window;
87
88      ngx_queue_t                     waiting;
89
90      u_char                          buffer[NGX_SPDY_STATE_BUFFER_SIZE];
91      size_t                          buffer_used;
92      ngx_http_spdy_handler_pt        handler;
93
94      z_stream                        zstream_in;
95      z_stream                        zstream_out;
96
97      ngx_pool_t                     *pool;
98
99      ngx_http_spdy_out_frame_t      *free_ctl_frames;
100      ngx_connection_t               *free_fake_connections;
101
102     ngx_http_spdy_stream_t        **streams_index;
103
104     ngx_http_spdy_out_frame_t      *last_out;
105
106     ngx_queue_t                     posted;
107
108     ngx_http_spdy_stream_t         *stream;
109
110     ngx_uint_t                      entries;
111     size_t                          length;
112     u_char                          flags;
113
114     ngx_uint_t                      last_sid;
115
116     unsigned                        blocked:1;
117     unsigned                        incomplete:1;
118 };
119
120
121 struct ngx_http_spdy_stream_s {
122     ngx_uint_t                      id;
123     ngx_http_request_t             *request;
124     ngx_http_spdy_connection_t     *connection;
125     ngx_http_spdy_stream_t          *index;
126
127     ngx_uint_t                      header_buffers;
128     ngx_uint_t                      queued;
129
130     /*
131      * A change to SETTINGS_INITIAL_WINDOW_SIZE could cause the
132      * send_window to become negative, hence it's signed.
133      */
134     ssize_t                         send_window;
135     size_t                          recv_window;
136
137     ngx_http_spdy_out_frame_t      *free_frames;
138     ngx_chain_t                    *free_data_headers;
139     ngx_chain_t                    *free_bufs;
140
141     ngx_queue_t                     queue;
142
143     unsigned                        priority:3;
144     unsigned                        handled:1;
145     unsigned                        blocked:1;
146     unsigned                        exhausted:1;
147     unsigned                        in_closed:1;
148     unsigned                        out_closed:1;
149     unsigned                        skip_data:2;
150 };
151
152
```

```
struct ngx_http_spdy_out_frame_s {
    ngx_http_spdy_out_frame_t       *next;
    ngx_chain_t                     *first;
    ngx_chain_t                     *last;
    ngx_int_t                      (*handler)(ngx_http_spdy_connection_t *sc,
                                         ngx_http_spdy_out_frame_t *frame);

    ngx_http_spdy_stream_t          *stream;
    size_t                           length;

    ngx_uint_t                       priority;
    unsigned                         blocked:1;
    unsigned                         fin:1;
};


static ngx_inline void
ngx_http_spdy_queue_frame(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_out_frame_t *frame)
{
    ngx_http_spdy_out_frame_t  **out;

    for (out = &sc->last_out; *out; out = &(*out)->next)
    {
        /*
         * NB: higher values represent lower priorities.
         */
        if (frame->priority >= (*out)->priority) {
            break;
        }
    }

    frame->next = *out;
    *out = frame;
}


static ngx_inline void
ngx_http_spdy_queue_blocked_frame(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_out_frame_t *frame)
{
    ngx_http_spdy_out_frame_t  **out;

    for (out = &sc->last_out; *out; out = &(*out)->next)
    {
        if ((*out)->blocked) {
            break;
        }
    }

    frame->next = *out;
    *out = frame;
}


void ngx_http_spdy_init(ngx_event_t *rev);
void ngx_http_spdy_request_headers_init(void);

ngx_int_t ngx_http_spdy_read_request_body(ngx_http_request_t *r,
    ngx_http_client_body_handler_pt post_handler);

void ngx_http_spdy_close_stream(ngx_http_spdy_stream_t *stream, ngx_int_t rc);

ngx_int_t ngx_http_spdy_send_output_queue(ngx_http_spdy_connection_t *sc);


#define ngx_spdy_frame_aligned_write_uint16(p, s)                             \
    (*(uint16_t *) (p) = htons((uint16_t) (s)), (p) + sizeof(uint16_t))

#define ngx_spdy_frame_aligned_write_uint32(p, s)                             \
    (*(uint32_t *) (p) = htonl((uint32_t) (s)), (p) + sizeof(uint32_t))

#if (NGX_HAVE_NONALIGNED)

#define ngx_spdy_frame_write_uint16  ngx_spdy_frame_aligned_write_uint16
#define ngx_spdy_frame_write_uint32  ngx_spdy_frame_aligned_write_uint32
```

```
229
230 #else
231
232 #define ngx_spdy_frame_write_uint16(p, s)                                    \
233     ((p)[0] = (u_char) ((s) >> 8),                                           \
234      (p)[1] = (u_char)  (s),                                                 \
235      (p) + sizeof(uint16_t))
236
237 #define ngx_spdy_frame_write_uint32(p, s)                                    \
238     ((p)[0] = (u_char) ((s) >> 24),                                          \
239      (p)[1] = (u_char) ((s) >> 16),                                          \
240      (p)[2] = (u_char) ((s) >> 8),                                           \
241      (p)[3] = (u_char)  (s),                                                 \
242      (p) + sizeof(uint32_t))
243
244 #endif
245
246
247 #define ngx_spdy_ctl_frame_head(t)                                           \
248     ((uint32_t) NGX_SPDY_CTL_BIT << 31 | NGX_SPDY_VERSION << 16 | (t))
249
250 #define ngx_spdy_frame_write_head(p, t)                                      \
251     ngx_spdy_frame_aligned_write_uint32(p, ngx_spdy_ctl_frame_head(t))
252
253 #define ngx_spdy_frame_write_flags_and_len(p, f, l)                          \
254     ngx_spdy_frame_aligned_write_uint32(p, (f) << 24 | (l))
255 #define ngx_spdy_frame_write_flags_and_id(p, f, i)                           \
256     ngx_spdy_frame_aligned_write_uint32(p, (f) << 24 | (i))
257
258 #define ngx_spdy_frame_write_sid     ngx_spdy_frame_aligned_write_uint32
259 #define ngx_spdy_frame_write_window  ngx_spdy_frame_aligned_write_uint32
260
261 #endif /* _NGX_HTTP_SPDY_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_request.c - nginx-1.7.10

Global variables defined

- ngx_http_client_errors
- ngx_http_headers_in

Functions defined

- ngx_http_alloc_large_header_buffer
- ngx_http_block_reading
- ngx_http_close_connection
- ngx_http_close_request
- ngx_http_create_request
- ngx_http_empty_handler
- ngx_http_finalize_connection
- ngx_http_finalize_request
- ngx_http_find_virtual_server
- ngx_http_free_request
- ngx_http_init_connection
- ngx_http_keepalive_handler
- ngx_http_lingering_close_handler
- ngx_http_log_error
- ngx_http_log_error_handler
- ngx_http_log_request
- ngx_http_post_action
- ngx_http_post_request
- ngx_http_process_connection
- ngx_http_process_header_line
- ngx_http_process_host
- ngx_http_process_multi_header_lines
- ngx_http_process_request
- ngx_http_process_request_header
- ngx_http_process_request_headers
- ngx_http_process_request_line

- ngx_http_process_request_uri
- ngx_http_process_unique_header_line
- ngx_http_process_user_agent
- ngx_http_read_request_header
- ngx_http_request_empty_handler
- ngx_http_request_finalizer
- ngx_http_request_handler
- ngx_http_run_posted_requests
- ngx_http_send_special
- ngx_http_set_keepalive
- ngx_http_set_lingering_close
- ngx_http_set_virtual_server
- ngx_http_set_write_handler
- ngx_http_ssl_handshake
- ngx_http_ssl_handshake_handler
- ngx_http_ssl_servername
- ngx_http_terminate_handler
- ngx_http_terminate_request
- ngx_http_test_reading
- ngx_http_validate_host
- ngx_http_wait_request_handler
- ngx_http_writer

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static void ngx_http_wait_request_handler(ngx_event_t *ev);
14  static void ngx_http_process_request_line(ngx_event_t *rev);
15  static void ngx_http_process_request_headers(ngx_event_t *rev);
16  static ssize_t ngx_http_read_request_header(ngx_http_request_t *r);
17  static ngx_int_t ngx_http_alloc_large_header_buffer(ngx_http_request_t *r,
18      ngx_uint_t request_line);
19
20  static ngx_int_t ngx_http_process_header_line(ngx_http_request_t *r,
21      ngx_table_elt_t *h, ngx_uint_t offset);
22  static ngx_int_t ngx_http_process_unique_header_line(ngx_http_request_t *r,
```

```
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_process_multi_header_lines(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_process_host(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_process_connection(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_process_user_agent(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);

static ngx_int_t ngx_http_validate_host(ngx_str_t *host, ngx_pool_t *pool,
    ngx_uint_t alloc);
static ngx_int_t ngx_http_set_virtual_server(ngx_http_request_t *r,
    ngx_str_t *host);
static ngx_int_t ngx_http_find_virtual_server(ngx_connection_t *c,
    ngx_http_virtual_names_t *virtual_names, ngx_str_t *host,
    ngx_http_request_t *r, ngx_http_core_srv_conf_t **cscfp);

static void ngx_http_request_handler(ngx_event_t *ev);
static void ngx_http_terminate_request(ngx_http_request_t *r, ngx_int_t rc);
static void ngx_http_terminate_handler(ngx_http_request_t *r);
static void ngx_http_finalize_connection(ngx_http_request_t *r);
static ngx_int_t ngx_http_set_write_handler(ngx_http_request_t *r);
static void ngx_http_writer(ngx_http_request_t *r);
static void ngx_http_request_finalizer(ngx_http_request_t *r);

static void ngx_http_set_keepalive(ngx_http_request_t *r);
static void ngx_http_keepalive_handler(ngx_event_t *ev);
static void ngx_http_set_lingering_close(ngx_http_request_t *r);
static void ngx_http_lingering_close_handler(ngx_event_t *ev);
static ngx_int_t ngx_http_post_action(ngx_http_request_t *r);
static void ngx_http_close_request(ngx_http_request_t *r, ngx_int_t error);
static void ngx_http_log_request(ngx_http_request_t *r);

static u_char *ngx_http_log_error(ngx_log_t *log, u_char *buf, size_t len);
static u_char *ngx_http_log_error_handler(ngx_http_request_t *r,
    ngx_http_request_t *sr, u_char *buf, size_t len);

#if (NGX_HTTP_SSL)
static void ngx_http_ssl_handshake(ngx_event_t *rev);
static void ngx_http_ssl_handshake_handler(ngx_connection_t *c);
#endif


static char *ngx_http_client_errors[] = {

    /* NGX_HTTP_PARSE_INVALID_METHOD */
    "client sent invalid method",

    /* NGX_HTTP_PARSE_INVALID_REQUEST */
    "client sent invalid request",

    /* NGX_HTTP_PARSE_INVALID_09_METHOD */
    "client sent invalid method in HTTP/0.9 request"
};


ngx_http_header_t  ngx_http_headers_in[] = {
    { ngx_string("Host"), offsetof(ngx_http_headers_in_t, host),
                 ngx_http_process_host },

    { ngx_string("Connection"), offsetof(ngx_http_headers_in_t, connection),
                 ngx_http_process_connection },

    { ngx_string("If-Modified-Since"),
                 offsetof(ngx_http_headers_in_t, if_modified_since),
                 ngx_http_process_unique_header_line },

    { ngx_string("If-Unmodified-Since"),
                 offsetof(ngx_http_headers_in_t, if_unmodified_since),
                 ngx_http_process_unique_header_line },

    { ngx_string("If-Match"),
                 offsetof(ngx_http_headers_in_t, if_match),
                 ngx_http_process_unique_header_line },

```

```
    { ngx_string("If-None-Match"),
                 offsetof(ngx_http_headers_in_t, if_none_match),
                 ngx_http_process_unique_header_line },

    { ngx_string("User-Agent"), offsetof(ngx_http_headers_in_t, user_agent),
                 ngx_http_process_user_agent },

    { ngx_string("Referer"), offsetof(ngx_http_headers_in_t, referer),
                 ngx_http_process_header_line },

    { ngx_string("Content-Length"),
                 offsetof(ngx_http_headers_in_t, content_length),
                 ngx_http_process_unique_header_line },

    { ngx_string("Content-Type"),
                 offsetof(ngx_http_headers_in_t, content_type),
                 ngx_http_process_header_line },

    { ngx_string("Range"), offsetof(ngx_http_headers_in_t, range),
                 ngx_http_process_header_line },

    { ngx_string("If-Range"),
                 offsetof(ngx_http_headers_in_t, if_range),
                 ngx_http_process_unique_header_line },

    { ngx_string("Transfer-Encoding"),
                 offsetof(ngx_http_headers_in_t, transfer_encoding),
                 ngx_http_process_header_line },

    { ngx_string("Expect"),
                 offsetof(ngx_http_headers_in_t, expect),
                 ngx_http_process_unique_header_line },

    { ngx_string("Upgrade"),
                 offsetof(ngx_http_headers_in_t, upgrade),
                 ngx_http_process_header_line },

#if (NGX_HTTP_GZIP)
    { ngx_string("Accept-Encoding"),
                 offsetof(ngx_http_headers_in_t, accept_encoding),
                 ngx_http_process_header_line },

    { ngx_string("Via"), offsetof(ngx_http_headers_in_t, via),
                 ngx_http_process_header_line },
#endif

    { ngx_string("Authorization"),
                 offsetof(ngx_http_headers_in_t, authorization),
                 ngx_http_process_unique_header_line },

    { ngx_string("Keep-Alive"), offsetof(ngx_http_headers_in_t, keep_alive),
                 ngx_http_process_header_line },

#if (NGX_HTTP_X_FORWARDED_FOR)
    { ngx_string("X-Forwarded-For"),
                 offsetof(ngx_http_headers_in_t, x_forwarded_for),
                 ngx_http_process_multi_header_lines },
#endif

#if (NGX_HTTP_REALIP)
    { ngx_string("X-Real-IP"),
                 offsetof(ngx_http_headers_in_t, x_real_ip),
                 ngx_http_process_header_line },
#endif

#if (NGX_HTTP_HEADERS)
    { ngx_string("Accept"), offsetof(ngx_http_headers_in_t, accept),
                 ngx_http_process_header_line },

    { ngx_string("Accept-Language"),
                 offsetof(ngx_http_headers_in_t, accept_language),
                 ngx_http_process_header_line },
#endif

#if (NGX_HTTP_DAV)
    { ngx_string("Depth"), offsetof(ngx_http_headers_in_t, depth),
```

```
                ngx_http_process_header_line },

    { ngx_string("Destination"), offsetof(ngx_http_headers_in_t, destination),
                ngx_http_process_header_line },

    { ngx_string("Overwrite"), offsetof(ngx_http_headers_in_t, overwrite),
                ngx_http_process_header_line },

    { ngx_string("Date"), offsetof(ngx_http_headers_in_t, date),
                ngx_http_process_header_line },
#endif

    { ngx_string("Cookie"), offsetof(ngx_http_headers_in_t, cookies),
                ngx_http_process_multi_header_lines },

    { ngx_null_string, 0, NULL }
};


void
ngx_http_init_connection(ngx_connection_t *c)
{
    ngx_uint_t              i;
    ngx_event_t            *rev;
    struct sockaddr_in     *sin;
    ngx_http_port_t        *port;
    ngx_http_in_addr_t     *addr;
    ngx_http_log_ctx_t     *ctx;
    ngx_http_connection_t  *hc;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6    *sin6;
    ngx_http_in6_addr_t    *addr6;
#endif

    hc = ngx_pcalloc(c->pool, sizeof(ngx_http_connection_t));
    if (hc == NULL) {
        ngx_http_close_connection(c);
        return;
    }

    c->data = hc;

    /* find the server configuration for the address:port */

    port = c->listening->servers;

    if (port->naddrs > 1) {

        /*
         * there are several addresses on this port and one of them
         * is an "*:port" wildcard so getsockname() in ngx_http_server_addr()
         * is required to determine a server address
         */

        if (ngx_connection_local_sockaddr(c, NULL, 0) != NGX_OK) {
            ngx_http_close_connection(c);
            return;
        }

        switch (c->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            sin6 = (struct sockaddr_in6 *) c->local_sockaddr;

            addr6 = port->addrs;

            /* the last address is "*" */

            for (i = 0; i < port->naddrs - 1; i++) {
                if (ngx_memcmp(&addr6[i].addr6, &sin6->sin6_addr, 16) == 0) {
                    break;
                }
            }

            hc->addr_conf = &addr6[i].conf;
```

```
            break;
#endif

        default: /* AF_INET */
            sin = (struct sockaddr_in *) c->local_sockaddr;

            addr = port->addrs;

            /* the last address is "*" */

            for (i = 0; i < port->naddrs - 1; i++) {
                if (addr[i].addr == sin->sin_addr.s_addr) {
                    break;
                }
            }

            hc->addr_conf = &addr[i].conf;

            break;
        }

    } else {

        switch (c->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            addr6 = port->addrs;
            hc->addr_conf = &addr6[0].conf;
            break;
#endif

        default: /* AF_INET */
            addr = port->addrs;
            hc->addr_conf = &addr[0].conf;
            break;
        }
    }

    /* the default server configuration for the address:port */
    hc->conf_ctx = hc->addr_conf->default_server->ctx;

    ctx = ngx_palloc(c->pool, sizeof(ngx_http_log_ctx_t));
    if (ctx == NULL) {
        ngx_http_close_connection(c);
        return;
    }

    ctx->connection = c;
    ctx->request = NULL;
    ctx->current_request = NULL;

    c->log->connection = c->number;
    c->log->handler = ngx_http_log_error;
    c->log->data = ctx;
    c->log->action = "waiting for request";

    c->log_error = NGX_ERROR_INFO;

    rev = c->read;
    rev->handler = ngx_http_wait_request_handler;
    c->write->handler = ngx_http_empty_handler;

#if (NGX_HTTP_SPDY)
    if (hc->addr_conf->spdy) {
        rev->handler = ngx_http_spdy_init;
    }
#endif

#if (NGX_HTTP_SSL)
    {
    ngx_http_ssl_srv_conf_t  *sscf;

    sscf = ngx_http_get_module_srv_conf(hc->conf_ctx, ngx_http_ssl_module);

```

```
    if (sscf->enable || hc->addr_conf->ssl) {

        c->log->action = "SSL handshaking";

        if (hc->addr_conf->ssl && sscf->ssl.ctx == NULL) {
            ngx_log_error(NGX_LOG_ERR, c->log, 0,
                          "no \"ssl_certificate\" is defined "
                          "in server listening on SSL port");
            ngx_http_close_connection(c);
            return;
        }

        hc->ssl = 1;

        rev->handler = ngx_http_ssl_handshake;
    }
    }
#endif

    if (hc->addr_conf->proxy_protocol) {
        hc->proxy_protocol = 1;
        c->log->action = "reading PROXY protocol";
    }

    if (rev->ready) {
        /* the deferred accept(), rtsig, aio, iocp */

        if (ngx_use_accept_mutex) {
            ngx_post_event(rev, &ngx_posted_events);
            return;
        }

        rev->handler(rev);
        return;
    }

    ngx_add_timer(rev, c->listening->post_accept_timeout);
    ngx_reusable_connection(c, 1);

    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
        ngx_http_close_connection(c);
        return;
    }
}


static void
ngx_http_wait_request_handler(ngx_event_t *rev)
{
    u_char                    *p;
    size_t                     size;
    ssize_t                    n;
    ngx_buf_t                 *b;
    ngx_connection_t          *c;
    ngx_http_connection_t     *hc;
    ngx_http_core_srv_conf_t  *cscf;

    c = rev->data;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "http wait request handler");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        ngx_http_close_connection(c);
        return;
    }

    if (c->close) {
        ngx_http_close_connection(c);
        return;
    }

    hc = c->data;
    cscf = ngx_http_get_module_srv_conf(hc->conf_ctx, ngx_http_core_module);

    size = cscf->client_header_buffer_size;
```

```
403
404     b = c->buffer;
405
406     if (b == NULL) {
407         b = ngx_create_temp_buf(c->pool, size);
408         if (b == NULL) {
409             ngx_http_close_connection(c);
410             return;
411         }
412
413         c->buffer = b;
414
415     } else if (b->start == NULL) {
416
417         b->start = ngx_palloc(c->pool, size);
418         if (b->start == NULL) {
419             ngx_http_close_connection(c);
420             return;
421         }
422
423         b->pos = b->start;
424         b->last = b->start;
425         b->end = b->last + size;
426     }
427
428     n = c->recv(c, b->last, size);
429
430     if (n == NGX_AGAIN) {
431
432         if (!rev->timer_set) {
433             ngx_add_timer(rev, c->listening->post_accept_timeout);
434             ngx_reusable_connection(c, 1);
435         }
436
437         if (ngx_handle_read_event(rev, 0) != NGX_OK) {
438             ngx_http_close_connection(c);
439             return;
440         }
441
442         /*
443          * We are trying to not hold c->buffer's memory for an idle connection.
444          */
445
446         if (ngx_pfree(c->pool, b->start) == NGX_OK) {
447             b->start = NULL;
448         }
449
450         return;
451     }
452
453     if (n == NGX_ERROR) {
454         ngx_http_close_connection(c);
455         return;
456     }
457
458     if (n == 0) {
459         ngx_log_error(NGX_LOG_INFO, c->log, 0,
460                       "client closed connection");
461         ngx_http_close_connection(c);
462         return;
463     }
464
465     b->last += n;
466
467     if (hc->proxy_protocol) {
468         hc->proxy_protocol = 0;
469
470         p = ngx_proxy_protocol_parse(c, b->pos, b->last);
471
472         if (p == NULL) {
473             ngx_http_close_connection(c);
474             return;
475         }
476
477         b->pos = p;
478
```

```
        if (b->pos == b->last) {
            c->log->action = "waiting for request";
            b->pos = b->start;
            b->last = b->start;
            ngx_post_event(rev, &ngx_posted_events);
            return;
        }
    }

    c->log->action = "reading client request line";

    ngx_reusable_connection(c, 0);

    c->data = ngx_http_create_request(c);
    if (c->data == NULL) {
        ngx_http_close_connection(c);
        return;
    }

    rev->handler = ngx_http_process_request_line;
    ngx_http_process_request_line(rev);
}


ngx_http_request_t *
ngx_http_create_request(ngx_connection_t *c)
{
    ngx_pool_t                 *pool;
    ngx_time_t                 *tp;
    ngx_http_request_t         *r;
    ngx_http_log_ctx_t         *ctx;
    ngx_http_connection_t      *hc;
    ngx_http_core_srv_conf_t   *cscf;
    ngx_http_core_loc_conf_t   *clcf;
    ngx_http_core_main_conf_t  *cmcf;

    c->requests++;

    hc = c->data;

    cscf = ngx_http_get_module_srv_conf(hc->conf_ctx, ngx_http_core_module);

    pool = ngx_create_pool(cscf->request_pool_size, c->log);
    if (pool == NULL) {
        return NULL;
    }

    r = ngx_pcalloc(pool, sizeof(ngx_http_request_t));
    if (r == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    r->pool = pool;

    r->http_connection = hc;
    r->signature = NGX_HTTP_MODULE;
    r->connection = c;

    r->main_conf = hc->conf_ctx->main_conf;
    r->srv_conf = hc->conf_ctx->srv_conf;
    r->loc_conf = hc->conf_ctx->loc_conf;

    r->read_event_handler = ngx_http_block_reading;

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    ngx_http_set_connection_log(r->connection, clcf->error_log);

    r->header_in = hc->nbusy ? hc->busy[0] : c->buffer;

    if (ngx_list_init(&r->headers_out.headers, r->pool, 20,
                      sizeof(ngx_table_elt_t))
        != NGX_OK)
    {
        ngx_destroy_pool(r->pool);
```

```
        return NULL;
    }

    r->ctx = ngx_pcalloc(r->pool, sizeof(void *) * ngx_http_max_module);
    if (r->ctx == NULL) {
        ngx_destroy_pool(r->pool);
        return NULL;
    }

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    r->variables = ngx_pcalloc(r->pool, cmcf->variables.nelts
                                        * sizeof(ngx_http_variable_value_t));
    if (r->variables == NULL) {
        ngx_destroy_pool(r->pool);
        return NULL;
    }

#if (NGX_HTTP_SSL)
    if (c->ssl) {
        r->main_filter_need_in_memory = 1;
    }
#endif

    r->main = r;
    r->count = 1;

    tp = ngx_timeofday();
    r->start_sec = tp->sec;
    r->start_msec = tp->msec;

    r->method = NGX_HTTP_UNKNOWN;
    r->http_version = NGX_HTTP_VERSION_10;

    r->headers_in.content_length_n = -1;
    r->headers_in.keep_alive_n = -1;
    r->headers_out.content_length_n = -1;
    r->headers_out.last_modified_time = -1;

    r->uri_changes = NGX_HTTP_MAX_URI_CHANGES + 1;
    r->subrequests = NGX_HTTP_MAX_SUBREQUESTS + 1;

    r->http_state = NGX_HTTP_READING_REQUEST_STATE;

    ctx = c->log->data;
    ctx->request = r;
    ctx->current_request = r;
    r->log_handler = ngx_http_log_error_handler;

#if (NGX_STAT_STUB)
    (void) ngx_atomic_fetch_add(ngx_stat_reading, 1);
    r->stat_reading = 1;
    (void) ngx_atomic_fetch_add(ngx_stat_requests, 1);
#endif

    return r;
}


#if (NGX_HTTP_SSL)

static void
ngx_http_ssl_handshake(ngx_event_t *rev)
{
    u_char                   *p, buf[NGX_PROXY_PROTOCOL_MAX_HEADER + 1];
    size_t                    size;
    ssize_t                   n;
    ngx_err_t                 err;
    ngx_int_t                 rc;
    ngx_connection_t         *c;
    ngx_http_connection_t    *hc;
    ngx_http_ssl_srv_conf_t  *sscf;

    c = rev->data;
    hc = c->data;

```

```
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, rev->log, 0,
                   "http check ssl handshake");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        ngx_http_close_connection(c);
        return;
    }

    if (c->close) {
        ngx_http_close_connection(c);
        return;
    }

    size = hc->proxy_protocol ? sizeof(buf) : 1;

    n = recv(c->fd, (char *) buf, size, MSG_PEEK);

    err = ngx_socket_errno;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, rev->log, 0, "http recv(): %d", n);

    if (n == -1) {
        if (err == NGX_EAGAIN) {

            if (!rev->timer_set) {
                ngx_add_timer(rev, c->listening->post_accept_timeout);
                ngx_reusable_connection(c, 1);
            }

            if (ngx_handle_read_event(rev, 0) != NGX_OK) {
                ngx_http_close_connection(c);
            }

            return;
        }

        ngx_connection_error(c, err, "recv() failed");
        ngx_http_close_connection(c);

        return;
    }

    if (hc->proxy_protocol) {
        hc->proxy_protocol = 0;

        p = ngx_proxy_protocol_parse(c, buf, buf + n);

        if (p == NULL) {
            ngx_http_close_connection(c);
            return;
        }

        size = p - buf;

        if (c->recv(c, buf, size) != (ssize_t) size) {
            ngx_http_close_connection(c);
            return;
        }

        c->log->action = "SSL handshaking";

        if (n == (ssize_t) size) {
            ngx_post_event(rev, &ngx_posted_events);
            return;
        }

        n = 1;
        buf[0] = *p;
    }

    if (n == 1) {
        if (buf[0] & 0x80 /* SSLv2 */ || buf[0] == 0x16 /* SSLv3/TLSv1 */) {
            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, rev->log, 0,
                           "https ssl handshake: 0x%02Xd", buf[0]);

```

```
707             sscf = ngx_http_get_module_srv_conf(hc->conf_ctx,
708                                                 ngx_http_ssl_module);
709
710             if (ngx_ssl_create_connection(&sscf->ssl, c, NGX_SSL_BUFFER)
711                 != NGX_OK)
712             {
713                 ngx_http_close_connection(c);
714                 return;
715             }
716
717             rc = ngx_ssl_handshake(c);
718
719             if (rc == NGX_AGAIN) {
720
721                 if (!rev->timer_set) {
722                     ngx_add_timer(rev, c->listening->post_accept_timeout);
723                 }
724
725                 ngx_reusable_connection(c, 0);
726
727                 c->ssl->handler = ngx_http_ssl_handshake_handler;
728                 return;
729             }
730
731             ngx_http_ssl_handshake_handler(c);
732
733             return;
734         }
735
736         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, rev->log, 0, "plain http");
737
738         c->log->action = "waiting for request";
739
740         rev->handler = ngx_http_wait_request_handler;
741         ngx_http_wait_request_handler(rev);
742
743         return;
744     }
745
746     ngx_log_error(NGX_LOG_INFO, c->log, 0, "client closed connection");
747     ngx_http_close_connection(c);
748 }
749
750
751 static void
752 ngx_http_ssl_handshake_handler(ngx_connection_t *c)
753 {
754     if (c->ssl->handshaked) {
755
756         /*
757          * The majority of browsers do not send the "close notify" alert.
758          * Among them are MSIE, old Mozilla, Netscape 4, Konqueror,
759          * and Links.  And what is more, MSIE ignores the server's alert.
760          *
761          * Opera and recent Mozilla send the alert.
762          */
763
764         c->ssl->no_wait_shutdown = 1;
765
766 #if (NGX_HTTP_SPDY                                                      \
767      && (defined TLSEXT_TYPE_application_layer_protocol_negotiation     \
768          || defined TLSEXT_TYPE_next_proto_neg))
769         {
770         unsigned int             len;
771         const unsigned char     *data;
772         static const ngx_str_t   spdy = ngx_string(NGX_SPDY_NPN_NEGOTIATED);
773
774 #ifdef TLSEXT_TYPE_application_layer_protocol_negotiation
775         SSL_get0_alpn_selected(c->ssl->connection, &data, &len);
776
777 #ifdef TLSEXT_TYPE_next_proto_neg
778         if (len == 0) {
779             SSL_get0_next_proto_negotiated(c->ssl->connection, &data, &len);
780         }
781 #endif
782
```

```
#else /* TLSEXT_TYPE_next_proto_neg */
        SSL_get0_next_proto_negotiated(c->ssl->connection, &data, &len);
#endif

        if (len == spdy.len && ngx_strncmp(data, spdy.data, spdy.len) == 0) {
            ngx_http_spdy_init(c->read);
            return;
        }
        }
#endif

        c->log->action = "waiting for request";

        c->read->handler = ngx_http_wait_request_handler;
        /* STUB: epoll edge */ c->write->handler = ngx_http_empty_handler;

        ngx_reusable_connection(c, 1);

        ngx_http_wait_request_handler(c->read);

        return;
    }

    if (c->read->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
    }

    ngx_http_close_connection(c);
}

#ifdef SSL_CTRL_SET_TLSEXT_HOSTNAME

int
ngx_http_ssl_servername(ngx_ssl_conn_t *ssl_conn, int *ad, void *arg)
{
    ngx_str_t                  host;
    const char                *servername;
    ngx_connection_t          *c;
    ngx_http_connection_t     *hc;
    ngx_http_ssl_srv_conf_t   *sscf;
    ngx_http_core_loc_conf_t  *clcf;
    ngx_http_core_srv_conf_t  *cscf;

    servername = SSL_get_servername(ssl_conn, TLSEXT_NAMETYPE_host_name);

    if (servername == NULL) {
        return SSL_TLSEXT_ERR_NOACK;
    }

    c = ngx_ssl_get_connection(ssl_conn);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "SSL server name: \"%s\"", servername);

    host.len = ngx_strlen(servername);

    if (host.len == 0) {
        return SSL_TLSEXT_ERR_NOACK;
    }

    host.data = (u_char *) servername;

    if (ngx_http_validate_host(&host, c->pool, 1) != NGX_OK) {
        return SSL_TLSEXT_ERR_NOACK;
    }

    hc = c->data;

    if (ngx_http_find_virtual_server(c, hc->addr_conf->virtual_names, &host,
                                     NULL, &cscf)
        != NGX_OK)
    {
        return SSL_TLSEXT_ERR_NOACK;
    }

    hc->ssl_servername = ngx_palloc(c->pool, sizeof(ngx_str_t));
```

```
    if (hc->ssl_servername == NULL) {
        return SSL_TLSEXT_ERR_NOACK;
    }

    *hc->ssl_servername = host;

    hc->conf_ctx = cscf->ctx;

    clcf = ngx_http_get_module_loc_conf(hc->conf_ctx, ngx_http_core_module);

    ngx_http_set_connection_log(c, clcf->error_log);

    sscf = ngx_http_get_module_srv_conf(hc->conf_ctx, ngx_http_ssl_module);

    if (sscf->ssl.ctx) {
        SSL_set_SSL_CTX(ssl_conn, sscf->ssl.ctx);

        /*
         * SSL_set_SSL_CTX() only changes certs as of 1.0.0d
         * adjust other things we care about
         */

        SSL_set_verify(ssl_conn, SSL_CTX_get_verify_mode(sscf->ssl.ctx),
                       SSL_CTX_get_verify_callback(sscf->ssl.ctx));

        SSL_set_verify_depth(ssl_conn, SSL_CTX_get_verify_depth(sscf->ssl.ctx));

#ifdef SSL_CTRL_CLEAR_OPTIONS
        /* only in 0.9.8m+ */
        SSL_clear_options(ssl_conn, SSL_get_options(ssl_conn) &
                                    ~SSL_CTX_get_options(sscf->ssl.ctx));
#endif

        SSL_set_options(ssl_conn, SSL_CTX_get_options(sscf->ssl.ctx));
    }

    return SSL_TLSEXT_ERR_OK;
}

#endif

#endif


static void
ngx_http_process_request_line(ngx_event_t *rev)
{
    ssize_t              n;
    ngx_int_t            rc, rv;
    ngx_str_t            host;
    ngx_connection_t    *c;
    ngx_http_request_t  *r;

    c = rev->data;
    r = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, rev->log, 0,
                   "http process request line");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        c->timedout = 1;
        ngx_http_close_request(r, NGX_HTTP_REQUEST_TIME_OUT);
        return;
    }

    rc = NGX_AGAIN;

    for ( ;; ) {

        if (rc == NGX_AGAIN) {
            n = ngx_http_read_request_header(r);

            if (n == NGX_AGAIN || n == NGX_ERROR) {
                return;
            }
```

```
        }

        rc = ngx_http_parse_request_line(r, r->header_in);

        if (rc == NGX_OK) {

            /* the request line has been parsed successfully */

            r->request_line.len = r->request_end - r->request_start;
            r->request_line.data = r->request_start;
            r->request_length = r->header_in->pos - r->request_start;

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                           "http request line: \"%V\"", &r->request_line);

            r->method_name.len = r->method_end - r->request_start + 1;
            r->method_name.data = r->request_line.data;

            if (r->http_protocol.data) {
                r->http_protocol.len = r->request_end - r->http_protocol.data;
            }

            if (ngx_http_process_request_uri(r) != NGX_OK) {
                return;
            }

            if (r->host_start && r->host_end) {

                host.len = r->host_end - r->host_start;
                host.data = r->host_start;

                rc = ngx_http_validate_host(&host, r->pool, 0);

                if (rc == NGX_DECLINED) {
                    ngx_log_error(NGX_LOG_INFO, c->log, 0,
                                  "client sent invalid host in request line");
                    ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
                    return;
                }

                if (rc == NGX_ERROR) {
                    ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
                    return;
                }

                if (ngx_http_set_virtual_server(r, &host) == NGX_ERROR) {
                    return;
                }

                r->headers_in.server = host;
            }

            if (r->http_version < NGX_HTTP_VERSION_10) {

                if (r->headers_in.server.len == 0
                    && ngx_http_set_virtual_server(r, &r->headers_in.server)
                       == NGX_ERROR)
                {
                    return;
                }

                ngx_http_process_request(r);
                return;
            }


             if (ngx_list_init(&r->headers_in.headers, r->pool, 20,
                               sizeof(ngx_table_elt_t))
                 != NGX_OK)
             {
                 ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
                 return;
             }

             c->log->action = "reading client request headers";

```

```
            rev->handler = ngx_http_process_request_headers;
            ngx_http_process_request_headers(rev);

            return;
        }

        if (rc != NGX_AGAIN) {

            /* there was error while a request line parsing */

            ngx_log_error(NGX_LOG_INFO, c->log, 0,
                          ngx_http_client_errors[rc - NGX_HTTP_CLIENT_ERROR]);
            ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
            return;
        }

        /* NGX_AGAIN: a request line parsing is still incomplete */

        if (r->header_in->pos == r->header_in->end) {

            rv = ngx_http_alloc_large_header_buffer(r, 1);

            if (rv == NGX_ERROR) {
                ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
                return;
            }

            if (rv == NGX_DECLINED) {
                r->request_line.len = r->header_in->end - r->request_start;
                r->request_line.data = r->request_start;

                ngx_log_error(NGX_LOG_INFO, c->log, 0,
                              "client sent too long URI");
                ngx_http_finalize_request(r, NGX_HTTP_REQUEST_URI_TOO_LARGE);
                return;
            }
        }
    }
}


ngx_int_t
ngx_http_process_request_uri(ngx_http_request_t *r)
{
    ngx_http_core_srv_conf_t  *cscf;

    if (r->args_start) {
        r->uri.len = r->args_start - 1 - r->uri_start;
    } else {
        r->uri.len = r->uri_end - r->uri_start;
    }

    if (r->complex_uri || r->quoted_uri) {

        r->uri.data = ngx_pnalloc(r->pool, r->uri.len + 1);
        if (r->uri.data == NULL) {
            ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return NGX_ERROR;
        }

        cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);

        if (ngx_http_parse_complex_uri(r, cscf->merge_slashes) != NGX_OK) {
            r->uri.len = 0;

            ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                          "client sent invalid request");
            ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
            return NGX_ERROR;
        }

    } else {
        r->uri.data = r->uri_start;
    }

    r->unparsed_uri.len = r->uri_end - r->uri_start;
```

```
    r->unparsed_uri.data = r->uri_start;

    r->valid_unparsed_uri = r->space_in_uri ? 0 : 1;

    if (r->uri_ext) {
        if (r->args_start) {
            r->exten.len = r->args_start - 1 - r->uri_ext;
        } else {
            r->exten.len = r->uri_end - r->uri_ext;
        }

        r->exten.data = r->uri_ext;
    }

    if (r->args_start && r->uri_end > r->args_start) {
        r->args.len = r->uri_end - r->args_start;
        r->args.data = r->args_start;
    }

#if (NGX_WIN32)
    {
    u_char  *p, *last;

    p = r->uri.data;
    last = r->uri.data + r->uri.len;

    while (p < last) {

        if (*p++ == ':') {

            /*
             * this check covers "::$data", "::$index_allocation" and
             * ":$i30:$index_allocation"
             */

            if (p < last && *p == '$') {
                ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                              "client sent unsafe win32 URI");
                ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
                return NGX_ERROR;
            }
        }
    }

    p = r->uri.data + r->uri.len - 1;

    while (p > r->uri.data) {

        if (*p == ' ') {
            p--;
            continue;
        }

        if (*p == '.') {
            p--;
            continue;
        }

        break;
    }

    if (p != r->uri.data + r->uri.len - 1) {
        r->uri.len = p + 1 - r->uri.data;
        ngx_http_set_exten(r);
    }

    }
#endif

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http uri: \"%V\"", &r->uri);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http args: \"%V\"", &r->args);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
```

```
                   "http exten: \"%V\"", &r->exten);

    return NGX_OK;
}


static void
ngx_http_process_request_headers(ngx_event_t *rev)
{
    u_char                     *p;
    size_t                      len;
    ssize_t                     n;
    ngx_int_t                   rc, rv;
    ngx_table_elt_t            *h;
    ngx_connection_t           *c;
    ngx_http_header_t          *hh;
    ngx_http_request_t         *r;
    ngx_http_core_srv_conf_t   *cscf;
    ngx_http_core_main_conf_t  *cmcf;

    c = rev->data;
    r = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, rev->log, 0,
                   "http process request header line");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        c->timedout = 1;
        ngx_http_close_request(r, NGX_HTTP_REQUEST_TIME_OUT);
        return;
    }

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    rc = NGX_AGAIN;

    for ( ;; ) {

        if (rc == NGX_AGAIN) {

            if (r->header_in->pos == r->header_in->end) {

                rv = ngx_http_alloc_large_header_buffer(r, 0);

                if (rv == NGX_ERROR) {
                    ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
                    return;
                }

                if (rv == NGX_DECLINED) {
                    p = r->header_name_start;

                    r->lingering_close = 1;

                    if (p == NULL) {
                        ngx_log_error(NGX_LOG_INFO, c->log, 0,
                                      "client sent too large request");
                        ngx_http_finalize_request(r,
                                                  NGX_HTTP_REQUEST_HEADER_TOO_LARGE);
                        return;
                    }

                    len = r->header_in->end - p;

                    if (len > NGX_MAX_ERROR_STR - 300) {
                        len = NGX_MAX_ERROR_STR - 300;
                    }

                    ngx_log_error(NGX_LOG_INFO, c->log, 0,
                                  "client sent too long header line: \"%*s...\"",
                                  len, r->header_name_start);

                    ngx_http_finalize_request(r,
                                              NGX_HTTP_REQUEST_HEADER_TOO_LARGE);
                    return;
```

```
            }
        }

        n = ngx_http_read_request_header(r);

        if (n == NGX_AGAIN || n == NGX_ERROR) {
            return;
        }
    }

    /* the host header could change the server configuration context */
    cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);

    rc = ngx_http_parse_header_line(r, r->header_in,
                                    cscf->underscores_in_headers);

    if (rc == NGX_OK) {

        r->request_length += r->header_in->pos - r->header_name_start;

        if (r->invalid_header && cscf->ignore_invalid_headers) {

            /* there was error while a header line parsing */

            ngx_log_error(NGX_LOG_INFO, c->log, 0,
                          "client sent invalid header line: \"%*s\"",
                          r->header_end - r->header_name_start,
                          r->header_name_start);
            continue;
        }

        /* a header line has been parsed successfully */

        h = ngx_list_push(&r->headers_in.headers);
        if (h == NULL) {
            ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        h->hash = r->header_hash;

        h->key.len = r->header_name_end - r->header_name_start;
        h->key.data = r->header_name_start;
        h->key.data[h->key.len] = '\0';

        h->value.len = r->header_end - r->header_start;
        h->value.data = r->header_start;
        h->value.data[h->value.len] = '\0';

        h->lowcase_key = ngx_pnalloc(r->pool, h->key.len);
        if (h->lowcase_key == NULL) {
            ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        if (h->key.len == r->lowcase_index) {
            ngx_memcpy(h->lowcase_key, r->lowcase_header, h->key.len);

        } else {
            ngx_strlow(h->lowcase_key, h->key.data, h->key.len);
        }

        hh = ngx_hash_find(&cmcf->headers_in_hash, h->hash,
                           h->lowcase_key, h->key.len);

        if (hh && hh->handler(r, h, hh->offset) != NGX_OK) {
            return;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http header: \"%V: %V\"",
                       &h->key, &h->value);

        continue;
    }
```

```
        if (rc == NGX_HTTP_PARSE_HEADER_DONE) {

            /* a whole header has been parsed successfully */

            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http header done");

            r->request_length += r->header_in->pos - r->header_name_start;

            r->http_state = NGX_HTTP_PROCESS_REQUEST_STATE;

            rc = ngx_http_process_request_header(r);

            if (rc != NGX_OK) {
                return;
            }

            ngx_http_process_request(r);

            return;
        }

        if (rc == NGX_AGAIN) {

            /* a header line parsing is still not complete */

            continue;
        }

        /* rc == NGX_HTTP_PARSE_INVALID_HEADER: "\r" is not followed by "\n" */

        ngx_log_error(NGX_LOG_INFO, c->log, 0,
                      "client sent invalid header line: \"%*s\\r...\"",
                      r->header_end - r->header_name_start,
                      r->header_name_start);
        ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
        return;
    }
}


static ssize_t
ngx_http_read_request_header(ngx_http_request_t *r)
{
    ssize_t                    n;
    ngx_event_t               *rev;
    ngx_connection_t          *c;
    ngx_http_core_srv_conf_t  *cscf;

    c = r->connection;
    rev = c->read;

    n = r->header_in->last - r->header_in->pos;

    if (n > 0) {
        return n;
    }

    if (rev->ready) {
        n = c->recv(c, r->header_in->last,
                    r->header_in->end - r->header_in->last);
    } else {
        n = NGX_AGAIN;
    }

    if (n == NGX_AGAIN) {
        if (!rev->timer_set) {
            cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
            ngx_add_timer(rev, cscf->client_header_timeout);
        }

        if (ngx_handle_read_event(rev, 0) != NGX_OK) {
            ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return NGX_ERROR;
        }
```

```
1391        return NGX_AGAIN;
1392    }
1393
1394    if (n == 0) {
1395        ngx_log_error(NGX_LOG_INFO, c->log, 0,
1396                      "client prematurely closed connection");
1397    }
1398
1399    if (n == 0 || n == NGX_ERROR) {
1400        c->error = 1;
1401        c->log->action = "reading client request headers";
1402
1403        ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
1404        return NGX_ERROR;
1405    }
1406
1407    r->header_in->last += n;
1408
1409    return n;
1410 }
1411
1412
1413 static ngx_int_t
1414 ngx_http_alloc_large_header_buffer(ngx_http_request_t *r,
1415     ngx_uint_t request_line)
1416 {
1417     u_char                    *old, *new;
1418     ngx_buf_t                 *b;
1419     ngx_http_connection_t     *hc;
1420     ngx_http_core_srv_conf_t  *cscf;
1421
1422     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1423                    "http alloc large header buffer");
1424
1425     if (request_line && r->state == 0) {
1426
1427         /* the client fills up the buffer with "\r\n" */
1428
1429         r->header_in->pos = r->header_in->start;
1430         r->header_in->last = r->header_in->start;
1431
1432         return NGX_OK;
1433     }
1434
1435     old = request_line ? r->request_start : r->header_name_start;
1436
1437     cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
1438
1439     if (r->state != 0
1440         && (size_t) (r->header_in->pos - old)
1441                                     >= cscf->large_client_header_buffers.size)
1442     {
1443         return NGX_DECLINED;
1444     }
1445
1446     hc = r->http_connection;
1447
1448     if (hc->nfree) {
1449         b = hc->free[--hc->nfree];
1450
1451         ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1452                        "http large header free: %p %uz",
1453                        b->pos, b->end - b->last);
1454
1455     } else if (hc->nbusy < cscf->large_client_header_buffers.num) {
1456
1457         if (hc->busy == NULL) {
1458             hc->busy = ngx_palloc(r->connection->pool,
1459                   cscf->large_client_header_buffers.num * sizeof(ngx_buf_t *));
1460             if (hc->busy == NULL) {
1461                 return NGX_ERROR;
1462             }
1463         }
1464
1465         b = ngx_create_temp_buf(r->connection->pool,
1466                                 cscf->large_client_header_buffers.size);
```

```
1467            if (b == NULL) {
1468                return NGX_ERROR;
1469            }
1470
1471            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1472                           "http large header alloc: %p %uz",
1473                           b->pos, b->end - b->last);
1474
1475        } else {
1476            return NGX_DECLINED;
1477        }
1478
1479        hc->busy[hc->nbusy++] = b;
1480
1481        if (r->state == 0) {
1482            /*
1483             * r->state == 0 means that a header line was parsed successfully
1484             * and we do not need to copy incomplete header line and
1485             * to relocate the parser header pointers
1486             */
1487
1488            r->header_in = b;
1489
1490            return NGX_OK;
1491        }
1492
1493        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1494                       "http large header copy: %d", r->header_in->pos - old);
1495
1496        new = b->start;
1497
1498        ngx_memcpy(new, old, r->header_in->pos - old);
1499
1500        b->pos = new + (r->header_in->pos - old);
1501        b->last = new + (r->header_in->pos - old);
1502
1503        if (request_line) {
1504            r->request_start = new;
1505
1506            if (r->request_end) {
1507                r->request_end = new + (r->request_end - old);
1508            }
1509
1510            r->method_end = new + (r->method_end - old);
1511
1512            r->uri_start = new + (r->uri_start - old);
1513            r->uri_end = new + (r->uri_end - old);
1514
1515            if (r->schema_start) {
1516                r->schema_start = new + (r->schema_start - old);
1517                r->schema_end = new + (r->schema_end - old);
1518            }
1519
1520            if (r->host_start) {
1521                r->host_start = new + (r->host_start - old);
1522                if (r->host_end) {
1523                    r->host_end = new + (r->host_end - old);
1524                }
1525            }
1526
1527            if (r->port_start) {
1528                r->port_start = new + (r->port_start - old);
1529                r->port_end = new + (r->port_end - old);
1530            }
1531
1532            if (r->uri_ext) {
1533                r->uri_ext = new + (r->uri_ext - old);
1534            }
1535
1536            if (r->args_start) {
1537                r->args_start = new + (r->args_start - old);
1538            }
1539
1540            if (r->http_protocol.data) {
1541                r->http_protocol.data = new + (r->http_protocol.data - old);
1542            }
```

```
    } else {
        r->header_name_start = new;
        r->header_name_end = new + (r->header_name_end - old);
        r->header_start = new + (r->header_start - old);
        r->header_end = new + (r->header_end - old);
    }

    r->header_in = b;

    return NGX_OK;
}


static ngx_int_t
ngx_http_process_header_line(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    ngx_table_elt_t  **ph;

    ph = (ngx_table_elt_t **) ((char *) &r->headers_in + offset);

    if (*ph == NULL) {
        *ph = h;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_process_unique_header_line(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    ngx_table_elt_t  **ph;

    ph = (ngx_table_elt_t **) ((char *) &r->headers_in + offset);

    if (*ph == NULL) {
        *ph = h;
        return NGX_OK;
    }

    ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                  "client sent duplicate header line: \"%V: %V\", "
                  "previous value: \"%V: %V\"",
                  &h->key, &h->value, &(*ph)->key, &(*ph)->value);

    ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_process_host(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    ngx_int_t  rc;
    ngx_str_t  host;

    if (r->headers_in.host == NULL) {
        r->headers_in.host = h;
    }

    host = h->value;

    rc = ngx_http_validate_host(&host, r->pool, 0);

    if (rc == NGX_DECLINED) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "client sent invalid host header");
        ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
        return NGX_ERROR;
    }

```

```
1619      if (rc == NGX_ERROR) {
1620          ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
1621          return NGX_ERROR;
1622      }
1623
1624      if (r->headers_in.server.len) {
1625          return NGX_OK;
1626      }
1627
1628      if (ngx_http_set_virtual_server(r, &host) == NGX_ERROR) {
1629          return NGX_ERROR;
1630      }
1631
1632      r->headers_in.server = host;
1633
1634      return NGX_OK;
1635  }
1636
1637
1638  static ngx_int_t
1639  ngx_http_process_connection(ngx_http_request_t *r, ngx_table_elt_t *h,
1640      ngx_uint_t offset)
1641  {
1642      if (ngx_strcasestrn(h->value.data, "close", 5 - 1)) {
1643          r->headers_in.connection_type = NGX_HTTP_CONNECTION_CLOSE;
1644
1645      } else if (ngx_strcasestrn(h->value.data, "keep-alive", 10 - 1)) {
1646          r->headers_in.connection_type = NGX_HTTP_CONNECTION_KEEP_ALIVE;
1647      }
1648
1649      return NGX_OK;
1650  }
1651
1652
1653  static ngx_int_t
1654  ngx_http_process_user_agent(ngx_http_request_t *r, ngx_table_elt_t *h,
1655      ngx_uint_t offset)
1656  {
1657      u_char  *user_agent, *msie;
1658
1659      if (r->headers_in.user_agent) {
1660          return NGX_OK;
1661      }
1662
1663      r->headers_in.user_agent = h;
1664
1665      /* check some widespread browsers while the header is in CPU cache */
1666
1667      user_agent = h->value.data;
1668
1669      msie = ngx_strstrn(user_agent, "MSIE ", 5 - 1);
1670
1671      if (msie && msie + 7 < user_agent + h->value.len) {
1672
1673          r->headers_in.msie = 1;
1674
1675          if (msie[6] == '.') {
1676
1677              switch (msie[5]) {
1678              case '4':
1679              case '5':
1680                  r->headers_in.msie6 = 1;
1681                  break;
1682              case '6':
1683                  if (ngx_strstrn(msie + 8, "SV1", 3 - 1) == NULL) {
1684                      r->headers_in.msie6 = 1;
1685                  }
1686                  break;
1687              }
1688          }
1689
1690  #if 0
1691          /* MSIE ignores the SSL "close notify" alert */
1692          if (c->ssl) {
1693              c->ssl->no_send_shutdown = 1;
1694          }
```

```
#endif
    }

    if (ngx_strstrn(user_agent, "Opera", 5 - 1)) {
        r->headers_in.opera = 1;
        r->headers_in.msie = 0;
        r->headers_in.msie6 = 0;
    }

    if (!r->headers_in.msie && !r->headers_in.opera) {

        if (ngx_strstrn(user_agent, "Gecko/", 6 - 1)) {
            r->headers_in.gecko = 1;

        } else if (ngx_strstrn(user_agent, "Chrome/", 7 - 1)) {
            r->headers_in.chrome = 1;

        } else if (ngx_strstrn(user_agent, "Safari/", 7 - 1)
                   && ngx_strstrn(user_agent, "Mac OS X", 8 - 1))
        {
            r->headers_in.safari = 1;

        } else if (ngx_strstrn(user_agent, "Konqueror", 9 - 1)) {
            r->headers_in.konqueror = 1;
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_process_multi_header_lines(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    ngx_array_t       *headers;
    ngx_table_elt_t  **ph;

    headers = (ngx_array_t *) ((char *) &r->headers_in + offset);

    if (headers->elts == NULL) {
        if (ngx_array_init(headers, r->pool, 1, sizeof(ngx_table_elt_t *))
            != NGX_OK)
        {
            ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return NGX_ERROR;
        }
    }

    ph = ngx_array_push(headers);
    if (ph == NULL) {
        ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return NGX_ERROR;
    }

    *ph = h;
    return NGX_OK;
}


ngx_int_t
ngx_http_process_request_header(ngx_http_request_t *r)
{
    if (r->headers_in.server.len == 0
        && ngx_http_set_virtual_server(r, &r->headers_in.server)
           == NGX_ERROR)
    {
        return NGX_ERROR;
    }

    if (r->headers_in.host == NULL && r->http_version > NGX_HTTP_VERSION_10) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "client sent HTTP/1.1 request without \"Host\" header");
        ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
        return NGX_ERROR;
    }
```

```
1771
1772        if (r->headers_in.content_length) {
1773            r->headers_in.content_length_n =
1774                                ngx_atoof(r->headers_in.content_length->value.data,
1775                                          r->headers_in.content_length->value.len);
1776
1777            if (r->headers_in.content_length_n == NGX_ERROR) {
1778                ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
1779                              "client sent invalid \"Content-Length\" header");
1780                ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
1781                return NGX_ERROR;
1782            }
1783        }
1784
1785        if (r->method & NGX_HTTP_TRACE) {
1786            ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
1787                          "client sent TRACE method");
1788            ngx_http_finalize_request(r, NGX_HTTP_NOT_ALLOWED);
1789            return NGX_ERROR;
1790        }
1791
1792        if (r->headers_in.transfer_encoding) {
1793            if (r->headers_in.transfer_encoding->value.len == 7
1794                && ngx_strncasecmp(r->headers_in.transfer_encoding->value.data,
1795                                   (u_char *) "chunked", 7) == 0)
1796            {
1797                r->headers_in.content_length = NULL;
1798                r->headers_in.content_length_n = -1;
1799                r->headers_in.chunked = 1;
1800
1801            } else if (r->headers_in.transfer_encoding->value.len != 8
1802                || ngx_strncasecmp(r->headers_in.transfer_encoding->value.data,
1803                                   (u_char *) "identity", 8) != 0)
1804            {
1805                ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
1806                              "client sent unknown \"Transfer-Encoding\": \"%V\"",
1807                              &r->headers_in.transfer_encoding->value);
1808                ngx_http_finalize_request(r, NGX_HTTP_NOT_IMPLEMENTED);
1809                return NGX_ERROR;
1810            }
1811        }
1812
1813        if (r->headers_in.connection_type == NGX_HTTP_CONNECTION_KEEP_ALIVE) {
1814            if (r->headers_in.keep_alive) {
1815                r->headers_in.keep_alive_n =
1816                                ngx_atotm(r->headers_in.keep_alive->value.data,
1817                                          r->headers_in.keep_alive->value.len);
1818            }
1819        }
1820
1821        return NGX_OK;
1822    }
1823
1824
1825    void
1826    ngx_http_process_request(ngx_http_request_t *r)
1827    {
1828        ngx_connection_t  *c;
1829
1830        c = r->connection;
1831
1832    #if (NGX_HTTP_SSL)
1833
1834        if (r->http_connection->ssl) {
1835            long                      rc;
1836            X509                     *cert;
1837            ngx_http_ssl_srv_conf_t  *sscf;
1838
1839            if (c->ssl == NULL) {
1840                ngx_log_error(NGX_LOG_INFO, c->log, 0,
1841                              "client sent plain HTTP request to HTTPS port");
1842                ngx_http_finalize_request(r, NGX_HTTP_TO_HTTPS);
1843                return;
1844            }
1845
1846            sscf = ngx_http_get_module_srv_conf(r, ngx_http_ssl_module);
```

```
1847
1848        if (sscf->verify) {
1849            rc = SSL_get_verify_result(c->ssl->connection);
1850
1851            if (rc != X509_V_OK
1852                && (sscf->verify != 3 || !ngx_ssl_verify_error_optional(rc)))
1853            {
1854                ngx_log_error(NGX_LOG_INFO, c->log, 0,
1855                              "client SSL certificate verify error: (%l:%s)",
1856                              rc, X509_verify_cert_error_string(rc));
1857
1858                ngx_ssl_remove_cached_session(sscf->ssl.ctx,
1859                                      (SSL_get0_session(c->ssl->connection)));
1860
1861                ngx_http_finalize_request(r, NGX_HTTPS_CERT_ERROR);
1862                return;
1863            }
1864
1865            if (sscf->verify == 1) {
1866                cert = SSL_get_peer_certificate(c->ssl->connection);
1867
1868                if (cert == NULL) {
1869                    ngx_log_error(NGX_LOG_INFO, c->log, 0,
1870                                  "client sent no required SSL certificate");
1871
1872                    ngx_ssl_remove_cached_session(sscf->ssl.ctx,
1873                                      (SSL_get0_session(c->ssl->connection)));
1874
1875                    ngx_http_finalize_request(r, NGX_HTTPS_NO_CERT);
1876                    return;
1877                }
1878
1879                X509_free(cert);
1880            }
1881        }
1882    }
1883
1884 #endif
1885
1886    if (c->read->timer_set) {
1887        ngx_del_timer(c->read);
1888    }
1889
1890 #if (NGX_STAT_STUB)
1891    (void) ngx_atomic_fetch_add(ngx_stat_reading, -1);
1892    r->stat_reading = 0;
1893    (void) ngx_atomic_fetch_add(ngx_stat_writing, 1);
1894    r->stat_writing = 1;
1895 #endif
1896
1897    c->read->handler = ngx_http_request_handler;
1898    c->write->handler = ngx_http_request_handler;
1899    r->read_event_handler = ngx_http_block_reading;
1900
1901    ngx_http_handler(r);
1902
1903    ngx_http_run_posted_requests(c);
1904 }
1905
1906
1907 static ngx_int_t
1908 ngx_http_validate_host(ngx_str_t *host, ngx_pool_t *pool, ngx_uint_t alloc)
1909 {
1910    u_char  *h, ch;
1911    size_t   i, dot_pos, host_len;
1912
1913    enum {
1914        sw_usual = 0,
1915        sw_literal,
1916        sw_rest
1917    } state;
1918
1919    dot_pos = host->len;
1920    host_len = host->len;
1921
1922    h = host->data;
```

```
1923
1924        state = sw_usual;
1925
1926        for (i = 0; i < host->len; i++) {
1927            ch = h[i];
1928
1929            switch (ch) {
1930
1931            case '.':
1932                if (dot_pos == i - 1) {
1933                    return NGX_DECLINED;
1934                }
1935                dot_pos = i;
1936                break;
1937
1938            case ':':
1939                if (state == sw_usual) {
1940                    host_len = i;
1941                    state = sw_rest;
1942                }
1943                break;
1944
1945            case '[':
1946                if (i == 0) {
1947                    state = sw_literal;
1948                }
1949                break;
1950
1951            case ']':
1952                if (state == sw_literal) {
1953                    host_len = i + 1;
1954                    state = sw_rest;
1955                }
1956                break;
1957
1958            case '\0':
1959                return NGX_DECLINED;
1960
1961            default:
1962
1963                if (ngx_path_separator(ch)) {
1964                    return NGX_DECLINED;
1965                }
1966
1967                if (ch >= 'A' && ch <= 'Z') {
1968                    alloc = 1;
1969                }
1970
1971                break;
1972            }
1973        }
1974
1975        if (dot_pos == host_len - 1) {
1976            host_len--;
1977        }
1978
1979        if (host_len == 0) {
1980            return NGX_DECLINED;
1981        }
1982
1983        if (alloc) {
1984            host->data = ngx_pnalloc(pool, host_len);
1985            if (host->data == NULL) {
1986                return NGX_ERROR;
1987            }
1988
1989            ngx_strlow(host->data, h, host_len);
1990        }
1991
1992        host->len = host_len;
1993
1994        return NGX_OK;
1995    }
1996
1997
1998    static ngx_int_t
```

```
1999 ngx_http_set_virtual_server(ngx_http_request_t *r, ngx_str_t *host)
2000 {
2001     ngx_int_t                  rc;
2002     ngx_http_connection_t     *hc;
2003     ngx_http_core_loc_conf_t  *clcf;
2004     ngx_http_core_srv_conf_t  *cscf;
2005 
2006 #if (NGX_SUPPRESS_WARN)
2007     cscf = NULL;
2008 #endif
2009 
2010     hc = r->http_connection;
2011 
2012 #if (NGX_HTTP_SSL && defined SSL_CTRL_SET_TLSEXT_HOSTNAME)
2013 
2014     if (hc->ssl_servername) {
2015         if (hc->ssl_servername->len == host->len
2016             && ngx_strncmp(hc->ssl_servername->data,
2017                            host->data, host->len) == 0)
2018         {
2019 #if (NGX_PCRE)
2020             if (hc->ssl_servername_regex
2021                 && ngx_http_regex_exec(r, hc->ssl_servername_regex,
2022                                        hc->ssl_servername) != NGX_OK)
2023             {
2024                 ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
2025                 return NGX_ERROR;
2026             }
2027 #endif
2028             return NGX_OK;
2029         }
2030     }
2031 
2032 #endif
2033 
2034     rc = ngx_http_find_virtual_server(r->connection,
2035                                       hc->addr_conf->virtual_names,
2036                                       host, r, &cscf);
2037 
2038     if (rc == NGX_ERROR) {
2039         ngx_http_close_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
2040         return NGX_ERROR;
2041     }
2042 
2043 #if (NGX_HTTP_SSL && defined SSL_CTRL_SET_TLSEXT_HOSTNAME)
2044 
2045     if (hc->ssl_servername) {
2046         ngx_http_ssl_srv_conf_t  *sscf;
2047 
2048         if (rc == NGX_DECLINED) {
2049             cscf = hc->addr_conf->default_server;
2050             rc = NGX_OK;
2051         }
2052 
2053         sscf = ngx_http_get_module_srv_conf(cscf->ctx, ngx_http_ssl_module);
2054 
2055         if (sscf->verify) {
2056             ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
2057                           "client attempted to request the server name "
2058                           "different from that one was negotiated");
2059             ngx_http_finalize_request(r, NGX_HTTP_BAD_REQUEST);
2060             return NGX_ERROR;
2061         }
2062     }
2063 
2064 #endif
2065 
2066     if (rc == NGX_DECLINED) {
2067         return NGX_OK;
2068     }
2069 
2070     r->srv_conf = cscf->ctx->srv_conf;
2071     r->loc_conf = cscf->ctx->loc_conf;
2072 
2073     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2074 
```

```
2075      ngx_http_set_connection_log(r->connection, clcf->error_log);
2076
2077      return NGX_OK;
2078  }
2079
2080
2081  static ngx_int_t
2082  ngx_http_find_virtual_server(ngx_connection_t *c,
2083      ngx_http_virtual_names_t *virtual_names, ngx_str_t *host,
2084      ngx_http_request_t *r, ngx_http_core_srv_conf_t **cscfp)
2085  {
2086      ngx_http_core_srv_conf_t  *cscf;
2087
2088      if (virtual_names == NULL) {
2089          return NGX_DECLINED;
2090      }
2091
2092      cscf = ngx_hash_find_combined(&virtual_names->names,
2093                                    ngx_hash_key(host->data, host->len),
2094                                    host->data, host->len);
2095
2096      if (cscf) {
2097          *cscfp = cscf;
2098          return NGX_OK;
2099      }
2100
2101  #if (NGX_PCRE)
2102
2103      if (host->len && virtual_names->nregex) {
2104          ngx_int_t                n;
2105          ngx_uint_t               i;
2106          ngx_http_server_name_t  *sn;
2107
2108          sn = virtual_names->regex;
2109
2110  #if (NGX_HTTP_SSL && defined SSL_CTRL_SET_TLSEXT_HOSTNAME)
2111
2112          if (r == NULL) {
2113              ngx_http_connection_t  *hc;
2114
2115              for (i = 0; i < virtual_names->nregex; i++) {
2116
2117                  n = ngx_regex_exec(sn[i].regex->regex, host, NULL, 0);
2118
2119                  if (n == NGX_REGEX_NO_MATCHED) {
2120                      continue;
2121                  }
2122
2123                  if (n >= 0) {
2124                      hc = c->data;
2125                      hc->ssl_servername_regex = sn[i].regex;
2126
2127                      *cscfp = sn[i].server;
2128                      return NGX_OK;
2129                  }
2130
2131                  ngx_log_error(NGX_LOG_ALERT, c->log, 0,
2132                                ngx_regex_exec_n " failed: %i "
2133                                "on \"%V\" using \"%V\"",
2134                                n, host, &sn[i].regex->name);
2135
2136                  return NGX_ERROR;
2137              }
2138
2139              return NGX_DECLINED;
2140          }
2141
2142  #endif /* NGX_HTTP_SSL && defined SSL_CTRL_SET_TLSEXT_HOSTNAME */
2143
2144          for (i = 0; i < virtual_names->nregex; i++) {
2145
2146              n = ngx_http_regex_exec(r, sn[i].regex, host);
2147
2148              if (n == NGX_DECLINED) {
2149                  continue;
2150              }
```

```
            if (n == NGX_OK) {
                *cscfp = sn[i].server;
                return NGX_OK;
            }

            return NGX_ERROR;
        }
    }

#endif /* NGX_PCRE */

    return NGX_DECLINED;
}


static void
ngx_http_request_handler(ngx_event_t *ev)
{
    ngx_connection_t    *c;
    ngx_http_request_t  *r;

    c = ev->data;
    r = c->data;

    ngx_http_set_log_request(c->log, r);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http run request: \"%V?%V\"", &r->uri, &r->args);

    if (ev->write) {
        r->write_event_handler(r);

    } else {
        r->read_event_handler(r);
    }

    ngx_http_run_posted_requests(c);
}


void
ngx_http_run_posted_requests(ngx_connection_t *c)
{
    ngx_http_request_t         *r;
    ngx_http_posted_request_t  *pr;

    for ( ;; ) {

        if (c->destroyed) {
            return;
        }

        r = c->data;
        pr = r->main->posted_requests;

        if (pr == NULL) {
            return;
        }

        r->main->posted_requests = pr->next;

        r = pr->request;

        ngx_http_set_log_request(c->log, r);

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
                       "http posted request: \"%V?%V\"", &r->uri, &r->args);

        r->write_event_handler(r);
    }
}


ngx_int_t
ngx_http_post_request(ngx_http_request_t *r, ngx_http_posted_request_t *pr)
```

```
{
    ngx_http_posted_request_t  **p;

    if (pr == NULL) {
        pr = ngx_palloc(r->pool, sizeof(ngx_http_posted_request_t));
        if (pr == NULL) {
            return NGX_ERROR;
        }
    }

    pr->request = r;
    pr->next = NULL;

    for (p = &r->main->posted_requests; *p; p = &(*p)->next) { /* void */ }

    *p = pr;

    return NGX_OK;
}


void
ngx_http_finalize_request(ngx_http_request_t *r, ngx_int_t rc)
{
    ngx_connection_t          *c;
    ngx_http_request_t        *pr;
    ngx_http_core_loc_conf_t  *clcf;

    c = r->connection;

    ngx_log_debug5(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http finalize request: %d, \"%V?%V\" a:%d, c:%d",
                   rc, &r->uri, &r->args, r == c->data, r->main->count);

    if (rc == NGX_DONE) {
        ngx_http_finalize_connection(r);
        return;
    }

    if (rc == NGX_OK && r->filter_finalize) {
        c->error = 1;
    }

    if (rc == NGX_DECLINED) {
        r->content_handler = NULL;
        r->write_event_handler = ngx_http_core_run_phases;
        ngx_http_core_run_phases(r);
        return;
    }

    if (r != r->main && r->post_subrequest) {
        rc = r->post_subrequest->handler(r, r->post_subrequest->data, rc);
    }

    if (rc == NGX_ERROR
        || rc == NGX_HTTP_REQUEST_TIME_OUT
        || rc == NGX_HTTP_CLIENT_CLOSED_REQUEST
        || c->error)
    {
        if (ngx_http_post_action(r) == NGX_OK) {
            return;
        }

        if (r->main->blocked) {
            r->write_event_handler = ngx_http_request_finalizer;
        }

        ngx_http_terminate_request(r, rc);
        return;
    }

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE
        || rc == NGX_HTTP_CREATED
        || rc == NGX_HTTP_NO_CONTENT)
    {
        if (rc == NGX_HTTP_CLOSE) {
```

```
2303            ngx_http_terminate_request(r, rc);
2304            return;
2305        }
2306
2307        if (r == r->main) {
2308            if (c->read->timer_set) {
2309                ngx_del_timer(c->read);
2310            }
2311
2312            if (c->write->timer_set) {
2313                ngx_del_timer(c->write);
2314            }
2315        }
2316
2317        c->read->handler = ngx_http_request_handler;
2318        c->write->handler = ngx_http_request_handler;
2319
2320        ngx_http_finalize_request(r, ngx_http_special_response_handler(r, rc));
2321        return;
2322    }
2323
2324    if (r != r->main) {
2325
2326        if (r->buffered || r->postponed) {
2327
2328            if (ngx_http_set_write_handler(r) != NGX_OK) {
2329                ngx_http_terminate_request(r, 0);
2330            }
2331
2332            return;
2333        }
2334
2335        pr = r->parent;
2336
2337        if (r == c->data) {
2338
2339            r->main->count--;
2340            r->main->subrequests++;
2341
2342            if (!r->logged) {
2343
2344                clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2345
2346                if (clcf->log_subrequest) {
2347                    ngx_http_log_request(r);
2348                }
2349
2350                r->logged = 1;
2351
2352            } else {
2353                ngx_log_error(NGX_LOG_ALERT, c->log, 0,
2354                              "subrequest: \"%V?%V\" logged again",
2355                              &r->uri, &r->args);
2356            }
2357
2358            r->done = 1;
2359
2360            if (pr->postponed && pr->postponed->request == r) {
2361                pr->postponed = pr->postponed->next;
2362            }
2363
2364            c->data = pr;
2365
2366        } else {
2367
2368            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
2369                           "http finalize non-active request: \"%V?%V\"",
2370                           &r->uri, &r->args);
2371
2372            r->write_event_handler = ngx_http_request_finalizer;
2373
2374            if (r->waited) {
2375                r->done = 1;
2376            }
2377        }
2378
```

```
2379         if (ngx_http_post_request(pr, NULL) != NGX_OK) {
2380             r->main->count++;
2381             ngx_http_terminate_request(r, 0);
2382             return;
2383         }
2384 
2385         ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
2386                        "http wake parent request: \"%V?%V\"",
2387                        &pr->uri, &pr->args);
2388 
2389         return;
2390     }
2391 
2392     if (r->buffered || c->buffered || r->postponed || r->blocked) {
2393 
2394         if (ngx_http_set_write_handler(r) != NGX_OK) {
2395             ngx_http_terminate_request(r, 0);
2396         }
2397 
2398         return;
2399     }
2400 
2401     if (r != c->data) {
2402         ngx_log_error(NGX_LOG_ALERT, c->log, 0,
2403                       "http finalize non-active request: \"%V?%V\"",
2404                       &r->uri, &r->args);
2405         return;
2406     }
2407 
2408     r->done = 1;
2409     r->write_event_handler = ngx_http_request_empty_handler;
2410 
2411     if (!r->post_action) {
2412         r->request_complete = 1;
2413     }
2414 
2415     if (ngx_http_post_action(r) == NGX_OK) {
2416         return;
2417     }
2418 
2419     if (c->read->timer_set) {
2420         ngx_del_timer(c->read);
2421     }
2422 
2423     if (c->write->timer_set) {
2424         c->write->delayed = 0;
2425         ngx_del_timer(c->write);
2426     }
2427 
2428     if (c->read->eof) {
2429         ngx_http_close_request(r, 0);
2430         return;
2431     }
2432 
2433     ngx_http_finalize_connection(r);
2434 }
2435 
2436 
2437 static void
2438 ngx_http_terminate_request(ngx_http_request_t *r, ngx_int_t rc)
2439 {
2440     ngx_http_cleanup_t    *cln;
2441     ngx_http_request_t    *mr;
2442     ngx_http_ephemeral_t  *e;
2443 
2444     mr = r->main;
2445 
2446     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2447                    "http terminate request count:%d", mr->count);
2448 
2449     if (rc > 0 && (mr->headers_out.status == 0 || mr->connection->sent == 0)) {
2450         mr->headers_out.status = rc;
2451     }
2452 
2453     cln = mr->cleanup;
2454     mr->cleanup = NULL;
```

```
2455
2456    while (cln) {
2457        if (cln->handler) {
2458            cln->handler(cln->data);
2459        }
2460
2461        cln = cln->next;
2462    }
2463
2464    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2465                   "http terminate cleanup count:%d blk:%d",
2466                   mr->count, mr->blocked);
2467
2468    if (mr->write_event_handler) {
2469
2470        if (mr->blocked) {
2471            return;
2472        }
2473
2474        e = ngx_http_ephemeral(mr);
2475        mr->posted_requests = NULL;
2476        mr->write_event_handler = ngx_http_terminate_handler;
2477        (void) ngx_http_post_request(mr, &e->terminal_posted_request);
2478        return;
2479    }
2480
2481    ngx_http_close_request(mr, rc);
2482 }
2483
2484
2485 static void
2486 ngx_http_terminate_handler(ngx_http_request_t *r)
2487 {
2488    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2489                   "http terminate handler count:%d", r->count);
2490
2491    r->count = 1;
2492
2493    ngx_http_close_request(r, 0);
2494 }
2495
2496
2497 static void
2498 ngx_http_finalize_connection(ngx_http_request_t *r)
2499 {
2500    ngx_http_core_loc_conf_t  *clcf;
2501
2502 #if (NGX_HTTP_SPDY)
2503    if (r->spdy_stream) {
2504        ngx_http_close_request(r, 0);
2505        return;
2506    }
2507 #endif
2508
2509    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2510
2511    if (r->main->count != 1) {
2512
2513        if (r->discard_body) {
2514            r->read_event_handler = ngx_http_discarded_request_body_handler;
2515            ngx_add_timer(r->connection->read, clcf->lingering_timeout);
2516
2517            if (r->lingering_time == 0) {
2518                r->lingering_time = ngx_time()
2519                                      + (time_t) (clcf->lingering_time / 1000);
2520            }
2521        }
2522
2523        ngx_http_close_request(r, 0);
2524        return;
2525    }
2526
2527    if (!ngx_terminate
2528         && !ngx_exiting
2529         && r->keepalive
2530         && clcf->keepalive_timeout > 0)
```

```
2531      {
2532          ngx_http_set_keepalive(r);
2533          return;
2534      }
2535
2536      if (clcf->lingering_close == NGX_HTTP_LINGERING_ALWAYS
2537          || (clcf->lingering_close == NGX_HTTP_LINGERING_ON
2538              && (r->lingering_close
2539                  || r->header_in->pos < r->header_in->last
2540                  || r->connection->read->ready)))
2541      {
2542          ngx_http_set_lingering_close(r);
2543          return;
2544      }
2545
2546      ngx_http_close_request(r, 0);
2547  }
2548
2549
2550  static ngx_int_t
2551  ngx_http_set_write_handler(ngx_http_request_t *r)
2552  {
2553      ngx_event_t               *wev;
2554      ngx_http_core_loc_conf_t  *clcf;
2555
2556      r->http_state = NGX_HTTP_WRITING_REQUEST_STATE;
2557
2558      r->read_event_handler = r->discard_body ?
2559                                  ngx_http_discarded_request_body_handler:
2560                                  ngx_http_test_reading;
2561      r->write_event_handler = ngx_http_writer;
2562
2563  #if (NGX_HTTP_SPDY)
2564      if (r->spdy_stream) {
2565          return NGX_OK;
2566      }
2567  #endif
2568
2569      wev = r->connection->write;
2570
2571      if (wev->ready && wev->delayed) {
2572          return NGX_OK;
2573      }
2574
2575      clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2576      if (!wev->delayed) {
2577          ngx_add_timer(wev, clcf->send_timeout);
2578      }
2579
2580      if (ngx_handle_write_event(wev, clcf->send_lowat) != NGX_OK) {
2581          ngx_http_close_request(r, 0);
2582          return NGX_ERROR;
2583      }
2584
2585      return NGX_OK;
2586  }
2587
2588
2589  static void
2590  ngx_http_writer(ngx_http_request_t *r)
2591  {
2592      int                        rc;
2593      ngx_event_t               *wev;
2594      ngx_connection_t          *c;
2595      ngx_http_core_loc_conf_t  *clcf;
2596
2597      c = r->connection;
2598      wev = c->write;
2599
2600      ngx_log_debug2(NGX_LOG_DEBUG_HTTP, wev->log, 0,
2601                     "http writer handler: \"%V?%V\"", &r->uri, &r->args);
2602
2603      clcf = ngx_http_get_module_loc_conf(r->main, ngx_http_core_module);
2604
2605      if (wev->timedout) {
2606          if (!wev->delayed) {
```

```
            ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT,
                          "client timed out");
            c->timedout = 1;

            ngx_http_finalize_request(r, NGX_HTTP_REQUEST_TIME_OUT);
            return;
        }

        wev->timedout = 0;
        wev->delayed = 0;

        if (!wev->ready) {
            ngx_add_timer(wev, clcf->send_timeout);

            if (ngx_handle_write_event(wev, clcf->send_lowat) != NGX_OK) {
                ngx_http_close_request(r, 0);
            }

            return;
        }

    }

    if (wev->delayed || r->aio) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, wev->log, 0,
                       "http writer delayed");

        if (ngx_handle_write_event(wev, clcf->send_lowat) != NGX_OK) {
            ngx_http_close_request(r, 0);
        }

        return;
    }

    rc = ngx_http_output_filter(r, NULL);

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http writer output filter: %d, \"%V?%V\"",
                   rc, &r->uri, &r->args);

    if (rc == NGX_ERROR) {
        ngx_http_finalize_request(r, rc);
        return;
    }

    if (r->buffered || r->postponed || (r == r->main && c->buffered)) {

#if (NGX_HTTP_SPDY)
        if (r->spdy_stream) {
            return;
        }
#endif

        if (!wev->delayed) {
            ngx_add_timer(wev, clcf->send_timeout);
        }

        if (ngx_handle_write_event(wev, clcf->send_lowat) != NGX_OK) {
            ngx_http_close_request(r, 0);
        }

        return;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, wev->log, 0,
                   "http writer done: \"%V?%V\"", &r->uri, &r->args);

    r->write_event_handler = ngx_http_request_empty_handler;

    ngx_http_finalize_request(r, rc);
}


static void
ngx_http_request_finalizer(ngx_http_request_t *r)
{
```

```
2683      ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2684                     "http finalizer done: \"%V?%V\"", &r->uri, &r->args);
2685
2686      ngx_http_finalize_request(r, 0);
2687  }
2688
2689
2690  void
2691  ngx_http_block_reading(ngx_http_request_t *r)
2692  {
2693      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2694                     "http reading blocked");
2695
2696      /* aio does not call this handler */
2697
2698      if ((ngx_event_flags & NGX_USE_LEVEL_EVENT)
2699          && r->connection->read->active)
2700      {
2701          if (ngx_del_event(r->connection->read, NGX_READ_EVENT, 0) != NGX_OK) {
2702              ngx_http_close_request(r, 0);
2703          }
2704      }
2705  }
2706
2707
2708  void
2709  ngx_http_test_reading(ngx_http_request_t *r)
2710  {
2711      int                n;
2712      char               buf[1];
2713      ngx_err_t          err;
2714      ngx_event_t       *rev;
2715      ngx_connection_t  *c;
2716
2717      c = r->connection;
2718      rev = c->read;
2719
2720      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "http test reading");
2721
2722  #if (NGX_HTTP_SPDY)
2723
2724      if (r->spdy_stream) {
2725          if (c->error) {
2726              err = 0;
2727              goto closed;
2728          }
2729
2730          return;
2731      }
2732
2733  #endif
2734
2735  #if (NGX_HAVE_KQUEUE)
2736
2737      if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
2738
2739          if (!rev->pending_eof) {
2740              return;
2741          }
2742
2743          rev->eof = 1;
2744          c->error = 1;
2745          err = rev->kq_errno;
2746
2747          goto closed;
2748      }
2749
2750  #endif
2751
2752  #if (NGX_HAVE_EPOLLRDHUP)
2753
2754      if ((ngx_event_flags & NGX_USE_EPOLL_EVENT) && rev->pending_eof) {
2755          socklen_t  len;
2756
2757          rev->eof = 1;
2758          c->error = 1;
```

```
2759
2760            err = 0;
2761            len = sizeof(ngx_err_t);
2762
2763            /*
2764             * BSDs and Linux return 0 and set a pending error in err
2765             * Solaris returns -1 and sets errno
2766             */
2767
2768            if (getsockopt(c->fd, SOL_SOCKET, SO_ERROR, (void *) &err, &len)
2769                == -1)
2770            {
2771                err = ngx_socket_errno;
2772            }
2773
2774            goto closed;
2775        }
2776
2777    #endif
2778
2779        n = recv(c->fd, buf, 1, MSG_PEEK);
2780
2781        if (n == 0) {
2782            rev->eof = 1;
2783            c->error = 1;
2784            err = 0;
2785
2786            goto closed;
2787
2788        } else if (n == -1) {
2789            err = ngx_socket_errno;
2790
2791            if (err != NGX_EAGAIN) {
2792                rev->eof = 1;
2793                c->error = 1;
2794
2795                goto closed;
2796            }
2797        }
2798
2799        /* aio does not call this handler */
2800
2801        if ((ngx_event_flags & NGX_USE_LEVEL_EVENT) && rev->active) {
2802
2803            if (ngx_del_event(rev, NGX_READ_EVENT, 0) != NGX_OK) {
2804                ngx_http_close_request(r, 0);
2805            }
2806        }
2807
2808        return;
2809
2810    closed:
2811
2812        if (err) {
2813            rev->error = 1;
2814        }
2815
2816        ngx_log_error(NGX_LOG_INFO, c->log, err,
2817                      "client prematurely closed connection");
2818
2819        ngx_http_finalize_request(r, NGX_HTTP_CLIENT_CLOSED_REQUEST);
2820    }
2821
2822
2823    static void
2824    ngx_http_set_keepalive(ngx_http_request_t *r)
2825    {
2826        int                        tcp_nodelay;
2827        ngx_int_t                  i;
2828        ngx_buf_t                 *b, *f;
2829        ngx_event_t               *rev, *wev;
2830        ngx_connection_t          *c;
2831        ngx_http_connection_t     *hc;
2832        ngx_http_core_srv_conf_t  *cscf;
2833        ngx_http_core_loc_conf_t  *clcf;
2834
```

```
    c = r->connection;
    rev = c->read;

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "set http keepalive handler");

    if (r->discard_body) {
        r->write_event_handler = ngx_http_request_empty_handler;
        r->lingering_time = ngx_time() + (time_t) (clcf->lingering_time / 1000);
        ngx_add_timer(rev, clcf->lingering_timeout);
        return;
    }

    c->log->action = "closing request";

    hc = r->http_connection;
    b = r->header_in;

    if (b->pos < b->last) {

        /* the pipelined request */

        if (b != c->buffer) {

            /*
             * If the large header buffers were allocated while the previous
             * request processing then we do not use c->buffer for
             * the pipelined request (see ngx_http_create_request()).
             *
             * Now we would move the large header buffers to the free list.
             */

            cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);

            if (hc->free == NULL) {
                hc->free = ngx_palloc(c->pool,
                  cscf->large_client_header_buffers.num * sizeof(ngx_buf_t *));

                if (hc->free == NULL) {
                    ngx_http_close_request(r, 0);
                    return;
                }
            }

            for (i = 0; i < hc->nbusy - 1; i++) {
                f = hc->busy[i];
                hc->free[hc->nfree++] = f;
                f->pos = f->start;
                f->last = f->start;
            }

            hc->busy[0] = b;
            hc->nbusy = 1;
        }
    }

    /* guard against recursive call from ngx_http_finalize_connection() */
    r->keepalive = 0;

    ngx_http_free_request(r, 0);

    c->data = hc;

    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
        ngx_http_close_connection(c);
        return;
    }

    wev = c->write;
    wev->handler = ngx_http_empty_handler;

    if (b->pos < b->last) {

        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "pipelined request");

```

```
2911            c->log->action = "reading client pipelined request line";
2912
2913            r = ngx_http_create_request(c);
2914            if (r == NULL) {
2915                ngx_http_close_connection(c);
2916                return;
2917            }
2918
2919            r->pipeline = 1;
2920
2921            c->data = r;
2922
2923            c->sent = 0;
2924            c->destroyed = 0;
2925
2926            if (rev->timer_set) {
2927                ngx_del_timer(rev);
2928            }
2929
2930            rev->handler = ngx_http_process_request_line;
2931            ngx_post_event(rev, &ngx_posted_events);
2932            return;
2933        }
2934
2935        /*
2936         * To keep a memory footprint as small as possible for an idle keepalive
2937         * connection we try to free c->buffer's memory if it was allocated outside
2938         * the c->pool.  The large header buffers are always allocated outside the
2939         * c->pool and are freed too.
2940         */
2941
2942        b = c->buffer;
2943
2944        if (ngx_pfree(c->pool, b->start) == NGX_OK) {
2945
2946            /*
2947             * the special note for ngx_http_keepalive_handler() that
2948             * c->buffer's memory was freed
2949             */
2950
2951            b->pos = NULL;
2952
2953        } else {
2954            b->pos = b->start;
2955            b->last = b->start;
2956        }
2957
2958        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0, "hc free: %p %d",
2959                       hc->free, hc->nfree);
2960
2961        if (hc->free) {
2962            for (i = 0; i < hc->nfree; i++) {
2963                ngx_pfree(c->pool, hc->free[i]->start);
2964                hc->free[i] = NULL;
2965            }
2966
2967            hc->nfree = 0;
2968        }
2969
2970        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0, "hc busy: %p %d",
2971                       hc->busy, hc->nbusy);
2972
2973        if (hc->busy) {
2974            for (i = 0; i < hc->nbusy; i++) {
2975                ngx_pfree(c->pool, hc->busy[i]->start);
2976                hc->busy[i] = NULL;
2977            }
2978
2979            hc->nbusy = 0;
2980        }
2981
2982    #if (NGX_HTTP_SSL)
2983        if (c->ssl) {
2984            ngx_ssl_free_buffer(c);
2985        }
2986    #endif
```

```
2987
2988        rev->handler = ngx_http_keepalive_handler;
2989
2990        if (wev->active && (ngx_event_flags & NGX_USE_LEVEL_EVENT)) {
2991            if (ngx_del_event(wev, NGX_WRITE_EVENT, 0) != NGX_OK) {
2992                ngx_http_close_connection(c);
2993                return;
2994            }
2995        }
2996
2997        c->log->action = "keepalive";
2998
2999        if (c->tcp_nopush == NGX_TCP_NOPUSH_SET) {
3000            if (ngx_tcp_push(c->fd) == -1) {
3001                ngx_connection_error(c, ngx_socket_errno, ngx_tcp_push_n " failed");
3002                ngx_http_close_connection(c);
3003                return;
3004            }
3005
3006            c->tcp_nopush = NGX_TCP_NOPUSH_UNSET;
3007            tcp_nodelay = ngx_tcp_nodelay_and_tcp_nopush ? 1 : 0;
3008
3009        } else {
3010            tcp_nodelay = 1;
3011        }
3012
3013        if (tcp_nodelay
3014            && clcf->tcp_nodelay
3015            && c->tcp_nodelay == NGX_TCP_NODELAY_UNSET)
3016        {
3017            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "tcp_nodelay");
3018
3019            if (setsockopt(c->fd, IPPROTO_TCP, TCP_NODELAY,
3020                           (const void *) &tcp_nodelay, sizeof(int))
3021                == -1)
3022            {
3023    #if (NGX_SOLARIS)
3024                /* Solaris returns EINVAL if a socket has been shut down */
3025                c->log_error = NGX_ERROR_IGNORE_EINVAL;
3026    #endif
3027
3028                ngx_connection_error(c, ngx_socket_errno,
3029                                     "setsockopt(TCP_NODELAY) failed");
3030
3031                c->log_error = NGX_ERROR_INFO;
3032                ngx_http_close_connection(c);
3033                return;
3034            }
3035
3036            c->tcp_nodelay = NGX_TCP_NODELAY_SET;
3037        }
3038
3039    #if 0
3040        /* if ngx_http_request_t was freed then we need some other place */
3041        r->http_state = NGX_HTTP_KEEPALIVE_STATE;
3042    #endif
3043
3044        c->idle = 1;
3045        ngx_reusable_connection(c, 1);
3046
3047        ngx_add_timer(rev, clcf->keepalive_timeout);
3048
3049        if (rev->ready) {
3050            ngx_post_event(rev, &ngx_posted_events);
3051        }
3052    }
3053
3054
3055    static void
3056    ngx_http_keepalive_handler(ngx_event_t *rev)
3057    {
3058        size_t             size;
3059        ssize_t            n;
3060        ngx_buf_t         *b;
3061        ngx_connection_t  *c;
3062
```

```
3063     c = rev->data;
3064
3065     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "http keepalive handler");
3066
3067     if (rev->timedout || c->close) {
3068         ngx_http_close_connection(c);
3069         return;
3070     }
3071
3072 #if (NGX_HAVE_KQUEUE)
3073
3074     if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
3075         if (rev->pending_eof) {
3076             c->log->handler = NULL;
3077             ngx_log_error(NGX_LOG_INFO, c->log, rev->kq_errno,
3078                           "kevent() reported that client %V closed "
3079                           "keepalive connection", &c->addr_text);
3080 #if (NGX_HTTP_SSL)
3081             if (c->ssl) {
3082                 c->ssl->no_send_shutdown = 1;
3083             }
3084 #endif
3085             ngx_http_close_connection(c);
3086             return;
3087         }
3088     }
3089
3090 #endif
3091
3092     b = c->buffer;
3093     size = b->end - b->start;
3094
3095     if (b->pos == NULL) {
3096
3097         /*
3098          * The c->buffer's memory was freed by ngx_http_set_keepalive().
3099          * However, the c->buffer->start and c->buffer->end were not changed
3100          * to keep the buffer size.
3101          */
3102
3103         b->pos = ngx_palloc(c->pool, size);
3104         if (b->pos == NULL) {
3105             ngx_http_close_connection(c);
3106             return;
3107         }
3108
3109         b->start = b->pos;
3110         b->last = b->pos;
3111         b->end = b->pos + size;
3112     }
3113
3114     /*
3115      * MSIE closes a keepalive connection with RST flag
3116      * so we ignore ECONNRESET here.
3117      */
3118
3119     c->log_error = NGX_ERROR_IGNORE_ECONNRESET;
3120     ngx_set_socket_errno(0);
3121
3122     n = c->recv(c, b->last, size);
3123     c->log_error = NGX_ERROR_INFO;
3124
3125     if (n == NGX_AGAIN) {
3126         if (ngx_handle_read_event(rev, 0) != NGX_OK) {
3127             ngx_http_close_connection(c);
3128             return;
3129         }
3130
3131         /*
3132          * Like ngx_http_set_keepalive() we are trying to not hold
3133          * c->buffer's memory for a keepalive connection.
3134          */
3135
3136         if (ngx_pfree(c->pool, b->start) == NGX_OK) {
3137
3138             /*
```

```
3139             * the special note that c->buffer's memory was freed
3140             */
3141
3142            b->pos = NULL;
3143        }
3144
3145        return;
3146    }
3147
3148    if (n == NGX_ERROR) {
3149        ngx_http_close_connection(c);
3150        return;
3151    }
3152
3153    c->log->handler = NULL;
3154
3155    if (n == 0) {
3156        ngx_log_error(NGX_LOG_INFO, c->log, ngx_socket_errno,
3157                      "client %V closed keepalive connection", &c->addr_text);
3158        ngx_http_close_connection(c);
3159        return;
3160    }
3161
3162    b->last += n;
3163
3164    c->log->handler = ngx_http_log_error;
3165    c->log->action = "reading client request line";
3166
3167    c->idle = 0;
3168    ngx_reusable_connection(c, 0);
3169
3170    c->data = ngx_http_create_request(c);
3171    if (c->data == NULL) {
3172        ngx_http_close_connection(c);
3173        return;
3174    }
3175
3176    c->sent = 0;
3177    c->destroyed = 0;
3178
3179    ngx_del_timer(rev);
3180
3181    rev->handler = ngx_http_process_request_line;
3182    ngx_http_process_request_line(rev);
3183}
3184
3185
3186static void
3187ngx_http_set_lingering_close(ngx_http_request_t *r)
3188{
3189    ngx_event_t               *rev, *wev;
3190    ngx_connection_t          *c;
3191    ngx_http_core_loc_conf_t  *clcf;
3192
3193    c = r->connection;
3194
3195    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
3196
3197    rev = c->read;
3198    rev->handler = ngx_http_lingering_close_handler;
3199
3200    r->lingering_time = ngx_time() + (time_t) (clcf->lingering_time / 1000);
3201    ngx_add_timer(rev, clcf->lingering_timeout);
3202
3203    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
3204        ngx_http_close_request(r, 0);
3205        return;
3206    }
3207
3208    wev = c->write;
3209    wev->handler = ngx_http_empty_handler;
3210
3211    if (wev->active && (ngx_event_flags & NGX_USE_LEVEL_EVENT)) {
3212        if (ngx_del_event(wev, NGX_WRITE_EVENT, 0) != NGX_OK) {
3213            ngx_http_close_request(r, 0);
3214            return;
```

```
3215        }
3216    }
3217
3218    if (ngx_shutdown_socket(c->fd, NGX_WRITE_SHUTDOWN) == -1) {
3219        ngx_connection_error(c, ngx_socket_errno,
3220                             ngx_shutdown_socket_n " failed");
3221        ngx_http_close_request(r, 0);
3222        return;
3223    }
3224
3225    if (rev->ready) {
3226        ngx_http_lingering_close_handler(rev);
3227    }
3228}
3229
3230
3231static void
3232ngx_http_lingering_close_handler(ngx_event_t *rev)
3233{
3234    ssize_t                    n;
3235    ngx_msec_t                 timer;
3236    ngx_connection_t          *c;
3237    ngx_http_request_t        *r;
3238    ngx_http_core_loc_conf_t  *clcf;
3239    u_char                     buffer[NGX_HTTP_LINGERING_BUFFER_SIZE];
3240
3241    c = rev->data;
3242    r = c->data;
3243
3244    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3245                   "http lingering close handler");
3246
3247    if (rev->timedout) {
3248        ngx_http_close_request(r, 0);
3249        return;
3250    }
3251
3252    timer = (ngx_msec_t) r->lingering_time - (ngx_msec_t) ngx_time();
3253    if ((ngx_msec_int_t) timer <= 0) {
3254        ngx_http_close_request(r, 0);
3255        return;
3256    }
3257
3258    do {
3259        n = c->recv(c, buffer, NGX_HTTP_LINGERING_BUFFER_SIZE);
3260
3261        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0, "lingering read: %d", n);
3262
3263        if (n == NGX_ERROR || n == 0) {
3264            ngx_http_close_request(r, 0);
3265            return;
3266        }
3267
3268    } while (rev->ready);
3269
3270    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
3271        ngx_http_close_request(r, 0);
3272        return;
3273    }
3274
3275    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
3276
3277    timer *= 1000;
3278
3279    if (timer > clcf->lingering_timeout) {
3280        timer = clcf->lingering_timeout;
3281    }
3282
3283    ngx_add_timer(rev, timer);
3284}
3285
3286
3287void
3288ngx_http_empty_handler(ngx_event_t *wev)
3289{
3290    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, wev->log, 0, "http empty handler");
```

```
3291
3292      return;
3293  }
3294
3295
3296  void
3297  ngx_http_request_empty_handler(ngx_http_request_t *r)
3298  {
3299      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3300                     "http request empty handler");
3301
3302      return;
3303  }
3304
3305
3306  ngx_int_t
3307  ngx_http_send_special(ngx_http_request_t *r, ngx_uint_t flags)
3308  {
3309      ngx_buf_t    *b;
3310      ngx_chain_t   out;
3311
3312      b = ngx_calloc_buf(r->pool);
3313      if (b == NULL) {
3314          return NGX_ERROR;
3315      }
3316
3317      if (flags & NGX_HTTP_LAST) {
3318
3319          if (r == r->main && !r->post_action) {
3320              b->last_buf = 1;
3321
3322          } else {
3323              b->sync = 1;
3324              b->last_in_chain = 1;
3325          }
3326      }
3327
3328      if (flags & NGX_HTTP_FLUSH) {
3329          b->flush = 1;
3330      }
3331
3332      out.buf = b;
3333      out.next = NULL;
3334
3335      return ngx_http_output_filter(r, &out);
3336  }
3337
3338
3339  static ngx_int_t
3340  ngx_http_post_action(ngx_http_request_t *r)
3341  {
3342      ngx_http_core_loc_conf_t  *clcf;
3343
3344      clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
3345
3346      if (clcf->post_action.data == NULL) {
3347          return NGX_DECLINED;
3348      }
3349
3350      if (r->post_action && r->uri_changes == 0) {
3351          return NGX_DECLINED;
3352      }
3353
3354      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3355                     "post action: \"%V\"", &clcf->post_action);
3356
3357      r->main->count--;
3358
3359      r->http_version = NGX_HTTP_VERSION_9;
3360      r->header_only = 1;
3361      r->post_action = 1;
3362
3363      r->read_event_handler = ngx_http_block_reading;
3364
3365      if (clcf->post_action.data[0] == '/') {
3366          ngx_http_internal_redirect(r, &clcf->post_action, NULL);
```

```
3367
3368        } else {
3369            ngx_http_named_location(r, &clcf->post_action);
3370        }
3371
3372        return NGX_OK;
3373    }
3374
3375
3376    static void
3377    ngx_http_close_request(ngx_http_request_t *r, ngx_int_t rc)
3378    {
3379        ngx_connection_t  *c;
3380
3381        r = r->main;
3382        c = r->connection;
3383
3384        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
3385                       "http request count:%d blk:%d", r->count, r->blocked);
3386
3387        if (r->count == 0) {
3388            ngx_log_error(NGX_LOG_ALERT, c->log, 0, "http request count is zero");
3389        }
3390
3391        r->count--;
3392
3393        if (r->count || r->blocked) {
3394            return;
3395        }
3396
3397    #if (NGX_HTTP_SPDY)
3398        if (r->spdy_stream) {
3399            ngx_http_spdy_close_stream(r->spdy_stream, rc);
3400            return;
3401        }
3402    #endif
3403
3404        ngx_http_free_request(r, rc);
3405        ngx_http_close_connection(c);
3406    }
3407
3408
3409    void
3410    ngx_http_free_request(ngx_http_request_t *r, ngx_int_t rc)
3411    {
3412        ngx_log_t                  *log;
3413        ngx_pool_t                 *pool;
3414        struct linger               linger;
3415        ngx_http_cleanup_t         *cln;
3416        ngx_http_log_ctx_t         *ctx;
3417        ngx_http_core_loc_conf_t   *clcf;
3418
3419        log = r->connection->log;
3420
3421        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, log, 0, "http close request");
3422
3423        if (r->pool == NULL) {
3424            ngx_log_error(NGX_LOG_ALERT, log, 0, "http request already closed");
3425            return;
3426        }
3427
3428        cln = r->cleanup;
3429        r->cleanup = NULL;
3430
3431        while (cln) {
3432            if (cln->handler) {
3433                cln->handler(cln->data);
3434            }
3435
3436            cln = cln->next;
3437        }
3438
3439    #if (NGX_STAT_STUB)
3440
3441        if (r->stat_reading) {
3442            (void) ngx_atomic_fetch_add(ngx_stat_reading, -1);
```

```
3443     }
3444
3445     if (r->stat_writing) {
3446         (void) ngx_atomic_fetch_add(ngx_stat_writing, -1);
3447     }
3448
3449 #endif
3450
3451     if (rc > 0 && (r->headers_out.status == 0 || r->connection->sent == 0)) {
3452         r->headers_out.status = rc;
3453     }
3454
3455     log->action = "logging request";
3456
3457     ngx_http_log_request(r);
3458
3459     log->action = "closing request";
3460
3461     if (r->connection->timedout) {
3462         clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
3463
3464         if (clcf->reset_timedout_connection) {
3465             linger.l_onoff = 1;
3466             linger.l_linger = 0;
3467
3468             if (setsockopt(r->connection->fd, SOL_SOCKET, SO_LINGER,
3469                            (const void *) &linger, sizeof(struct linger)) == -1)
3470             {
3471                 ngx_log_error(NGX_LOG_ALERT, log, ngx_socket_errno,
3472                               "setsockopt(SO_LINGER) failed");
3473             }
3474         }
3475     }
3476
3477     /* the various request strings were allocated from r->pool */
3478     ctx = log->data;
3479     ctx->request = NULL;
3480
3481     r->request_line.len = 0;
3482
3483     r->connection->destroyed = 1;
3484
3485     /*
3486      * Setting r->pool to NULL will increase probability to catch double close
3487      * of request since the request object is allocated from its own pool.
3488      */
3489
3490     pool = r->pool;
3491     r->pool = NULL;
3492
3493     ngx_destroy_pool(pool);
3494 }
3495
3496
3497 static void
3498 ngx_http_log_request(ngx_http_request_t *r)
3499 {
3500     ngx_uint_t                  i, n;
3501     ngx_http_handler_pt        *log_handler;
3502     ngx_http_core_main_conf_t  *cmcf;
3503
3504     cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
3505
3506     log_handler = cmcf->phases[NGX_HTTP_LOG_PHASE].handlers.elts;
3507     n = cmcf->phases[NGX_HTTP_LOG_PHASE].handlers.nelts;
3508
3509     for (i = 0; i < n; i++) {
3510         log_handler[i](r);
3511     }
3512 }
3513
3514
3515 void
3516 ngx_http_close_connection(ngx_connection_t *c)
3517 {
3518     ngx_pool_t  *pool;
```

```
3519
3520        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
3521                       "close http connection: %d", c->fd);
3522
3523    #if (NGX_HTTP_SSL)
3524
3525        if (c->ssl) {
3526            if (ngx_ssl_shutdown(c) == NGX_AGAIN) {
3527                c->ssl->handler = ngx_http_close_connection;
3528                return;
3529            }
3530        }
3531
3532    #endif
3533
3534    #if (NGX_STAT_STUB)
3535        (void) ngx_atomic_fetch_add(ngx_stat_active, -1);
3536    #endif
3537
3538        c->destroyed = 1;
3539
3540        pool = c->pool;
3541
3542        ngx_close_connection(c);
3543
3544        ngx_destroy_pool(pool);
3545    }
3546
3547
3548    static u_char *
3549    ngx_http_log_error(ngx_log_t *log, u_char *buf, size_t len)
3550    {
3551        u_char              *p;
3552        ngx_http_request_t  *r;
3553        ngx_http_log_ctx_t  *ctx;
3554
3555        if (log->action) {
3556            p = ngx_snprintf(buf, len, " while %s", log->action);
3557            len -= p - buf;
3558            buf = p;
3559        }
3560
3561        ctx = log->data;
3562
3563        p = ngx_snprintf(buf, len, ", client: %V", &ctx->connection->addr_text);
3564        len -= p - buf;
3565
3566        r = ctx->request;
3567
3568        if (r) {
3569            return r->log_handler(r, ctx->current_request, p, len);
3570
3571        } else {
3572            p = ngx_snprintf(p, len, ", server: %V",
3573                             &ctx->connection->listening->addr_text);
3574        }
3575
3576        return p;
3577    }
3578
3579
3580    static u_char *
3581    ngx_http_log_error_handler(ngx_http_request_t *r, ngx_http_request_t *sr,
3582        u_char *buf, size_t len)
3583    {
3584        char                      *uri_separator;
3585        u_char                    *p;
3586        ngx_http_upstream_t       *u;
3587        ngx_http_core_srv_conf_t  *cscf;
3588
3589        cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
3590
3591        p = ngx_snprintf(buf, len, ", server: %V", &cscf->server_name);
3592        len -= p - buf;
3593        buf = p;
3594
```

```
3595     if (r->request_line.data == NULL && r->request_start) {
3596         for (p = r->request_start; p < r->header_in->last; p++) {
3597             if (*p == CR || *p == LF) {
3598                 break;
3599             }
3600         }
3601 
3602         r->request_line.len = p - r->request_start;
3603         r->request_line.data = r->request_start;
3604     }
3605 
3606     if (r->request_line.len) {
3607         p = ngx_snprintf(buf, len, ", request: \"%V\"", &r->request_line);
3608         len -= p - buf;
3609         buf = p;
3610     }
3611 
3612     if (r != sr) {
3613         p = ngx_snprintf(buf, len, ", subrequest: \"%V\"", &sr->uri);
3614         len -= p - buf;
3615         buf = p;
3616     }
3617 
3618     u = sr->upstream;
3619 
3620     if (u && u->peer.name) {
3621 
3622         uri_separator = "";
3623 
3624 #if (NGX_HAVE_UNIX_DOMAIN)
3625         if (u->peer.sockaddr && u->peer.sockaddr->sa_family == AF_UNIX) {
3626             uri_separator = ":";
3627         }
3628 #endif
3629 
3630         p = ngx_snprintf(buf, len, ", upstream: \"%V%V%s%V\"",
3631                          &u->schema, u->peer.name,
3632                          uri_separator, &u->uri);
3633         len -= p - buf;
3634         buf = p;
3635     }
3636 
3637     if (r->headers_in.host) {
3638         p = ngx_snprintf(buf, len, ", host: \"%V\"",
3639                          &r->headers_in.host->value);
3640         len -= p - buf;
3641         buf = p;
3642     }
3643 
3644     if (r->headers_in.referer) {
3645         p = ngx_snprintf(buf, len, ", referrer: \"%V\"",
3646                          &r->headers_in.referer->value);
3647         buf = p;
3648     }
3649 
3650     return buf;
3651 }
```

One Level Up Top Level

src/http/modules/ngx_http_ssl_module.h - nginx-1.7.10

Data types defined

- ngx_http_ssl_srv_conf_t

Macros defined

- _NGX_HTTP_SSL_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_HTTP_SSL_H_INCLUDED_
9  #define _NGX_HTTP_SSL_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
17  typedef struct {
18      ngx_flag_t                     enable;
19
20      ngx_ssl_t                      ssl;
21
22      ngx_flag_t                     prefer_server_ciphers;
23
24      ngx_uint_t                     protocols;
25
26      ngx_uint_t                     verify;
27      ngx_uint_t                     verify_depth;
28
29      size_t                         buffer_size;
30
31      ssize_t                        builtin_session_cache;
32
33      time_t                         session_timeout;
34
35      ngx_str_t                      certificate;
36      ngx_str_t                      certificate_key;
37      ngx_str_t                      dhparam;
38      ngx_str_t                      ecdh_curve;
39      ngx_str_t                      client_certificate;
40      ngx_str_t                      trusted_certificate;
41      ngx_str_t                      crl;
42
43      ngx_str_t                      ciphers;
44
45      ngx_array_t                   *passwords;
46
47      ngx_shm_zone_t                *shm_zone;
48
49      ngx_flag_t                     session_tickets;
50      ngx_array_t                   *session_ticket_keys;
51
52      ngx_flag_t                     stapling;
53      ngx_flag_t                     stapling_verify;
54      ngx_str_t                      stapling_file;
55      ngx_str_t                      stapling_responder;
56
57      u_char                        *file;
```

```
    ngx_uint_t                      line;
} ngx_http_ssl_srv_conf_t;


extern ngx_module_t  ngx_http_ssl_module;


#endif /* _NGX_HTTP_SSL_H_INCLUDED_ */
```

One Level Up Top Level

src/event/ngx_event_openssl.h - nginx-1.7.10

Data types defined

- ngx_ssl_connection_t
- ngx_ssl_sess_id_s
- ngx_ssl_sess_id_t
- ngx_ssl_session_cache_t
- ngx_ssl_session_ticket_key_t
- ngx_ssl_t

Macros defined

- NGX_SSL_BUFFER
- NGX_SSL_BUFSIZE
- NGX_SSL_CLIENT
- NGX_SSL_DFLT_BUILTIN_SCACHE
- NGX_SSL_MAX_SESSION_SIZE
- NGX_SSL_NAME
- NGX_SSL_NONE_SCACHE
- NGX_SSL_NO_BUILTIN_SCACHE
- NGX_SSL_NO_SCACHE
- NGX_SSL_SSLv2
- NGX_SSL_SSLv3
- NGX_SSL_TLSv1
- NGX_SSL_TLSv1_1
- NGX_SSL_TLSv1_2
- _NGX_EVENT_OPENSSL_H_INCLUDED_
- ngx_ssl_conn_t
- ngx_ssl_free_session
- ngx_ssl_get_connection
- ngx_ssl_get_server_conf
- ngx_ssl_get_session
- ngx_ssl_session_t
- ngx_ssl_verify_error_optional

Source code

```

/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#ifndef _NGX_EVENT_OPENSSL_H_INCLUDED_
#define _NGX_EVENT_OPENSSL_H_INCLUDED_


 #include <ngx_config.h>
 #include <ngx_core.h>

 #include <openssl/ssl.h>
 #include <openssl/err.h>
 #include <openssl/bn.h>
 #include <openssl/conf.h>
 #include <openssl/crypto.h>
 #include <openssl/dh.h>
 #ifndef OPENSSL_NO_ENGINE
 #include <openssl/engine.h>
 #endif
 #include <openssl/evp.h>
 #ifndef OPENSSL_NO_OCSP
 #include <openssl/ocsp.h>
 #endif
 #include <openssl/rand.h>
 #include <openssl/rsa.h>
 #include <openssl/x509.h>
 #include <openssl/x509v3.h>

 #define NGX_SSL_NAME     "OpenSSL"


 #define ngx_ssl_session_t       SSL_SESSION
 #define ngx_ssl_conn_t          SSL


 typedef struct {
     SSL_CTX                    *ctx;
     ngx_log_t                  *log;
     size_t                      buffer_size;
 } ngx_ssl_t;


 typedef struct {
     ngx_ssl_conn_t             *connection;

     ngx_int_t                   last;
     ngx_buf_t                  *buf;
     size_t                      buffer_size;

     ngx_connection_handler_pt   handler;

     ngx_event_handler_pt        saved_read_handler;
     ngx_event_handler_pt        saved_write_handler;

     unsigned                    handshaked:1;
     unsigned                    renegotiation:1;
     unsigned                    buffer:1;
     unsigned                    no_wait_shutdown:1;
     unsigned                    no_send_shutdown:1;
     unsigned                    handshake_buffer_set:1;
 } ngx_ssl_connection_t;


 #define NGX_SSL_NO_SCACHE            -2
 #define NGX_SSL_NONE_SCACHE          -3
 #define NGX_SSL_NO_BUILTIN_SCACHE    -4
 #define NGX_SSL_DFLT_BUILTIN_SCACHE  -5


```

```c
#define NGX_SSL_MAX_SESSION_SIZE  4096

typedef struct ngx_ssl_sess_id_s  ngx_ssl_sess_id_t;

struct ngx_ssl_sess_id_s {
    ngx_rbtree_node_t           node;
    u_char                     *id;
    size_t                      len;
    u_char                     *session;
    ngx_queue_t                 queue;
    time_t                      expire;
#if (NGX_PTR_SIZE == 8)
    void                       *stub;
    u_char                      sess_id[32];
#endif
};


typedef struct {
    ngx_rbtree_t                session_rbtree;
    ngx_rbtree_node_t           sentinel;
    ngx_queue_t                 expire_queue;
} ngx_ssl_session_cache_t;


#ifdef SSL_CTRL_SET_TLSEXT_TICKET_KEY_CB

 typedef struct {
     u_char                     name[16];
     u_char                     aes_key[16];
     u_char                     hmac_key[16];
 } ngx_ssl_session_ticket_key_t;

 #endif


 #define NGX_SSL_SSLv2    0x0002
 #define NGX_SSL_SSLv3    0x0004
 #define NGX_SSL_TLSv1    0x0008
 #define NGX_SSL_TLSv1_1  0x0010
 #define NGX_SSL_TLSv1_2  0x0020


 #define NGX_SSL_BUFFER   1
 #define NGX_SSL_CLIENT   2

 #define NGX_SSL_BUFSIZE  16384


 ngx_int_t ngx_ssl_init(ngx_log_t *log);
 ngx_int_t ngx_ssl_create(ngx_ssl_t *ssl, ngx_uint_t protocols, void *data);
 ngx_int_t ngx_ssl_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl,
     ngx_str_t *cert, ngx_str_t *key, ngx_array_t *passwords);
 ngx_int_t ngx_ssl_client_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl,
     ngx_str_t *cert, ngx_int_t depth);
 ngx_int_t ngx_ssl_trusted_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl,
     ngx_str_t *cert, ngx_int_t depth);
 ngx_int_t ngx_ssl_crl(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *crl);
 ngx_int_t ngx_ssl_stapling(ngx_conf_t *cf, ngx_ssl_t *ssl,
     ngx_str_t *file, ngx_str_t *responder, ngx_uint_t verify);
 ngx_int_t ngx_ssl_stapling_resolver(ngx_conf_t *cf, ngx_ssl_t *ssl,
     ngx_resolver_t *resolver, ngx_msec_t resolver_timeout);
 RSA *ngx_ssl_rsa512_key_callback(ngx_ssl_conn_t *ssl_conn, int is_export,
     int key_length);
 ngx_array_t *ngx_ssl_read_password_file(ngx_conf_t *cf, ngx_str_t *file);
 ngx_int_t ngx_ssl_dhparam(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *file);
 ngx_int_t ngx_ssl_ecdh_curve(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *name);
 ngx_int_t ngx_ssl_session_cache(ngx_ssl_t *ssl, ngx_str_t *sess_ctx,
     ssize_t builtin_session_cache, ngx_shm_zone_t *shm_zone, time_t timeout);
 ngx_int_t ngx_ssl_session_ticket_keys(ngx_conf_t *cf, ngx_ssl_t *ssl,
     ngx_array_t *paths);
 ngx_int_t ngx_ssl_session_cache_init(ngx_shm_zone_t *shm_zone, void *data);
 ngx_int_t ngx_ssl_create_connection(ngx_ssl_t *ssl, ngx_connection_t *c,
     ngx_uint_t flags);

 void ngx_ssl_remove_cached_session(SSL_CTX *ssl, ngx_ssl_session_t *sess);
```

```
ngx_int_t ngx_ssl_set_session(ngx_connection_t *c, ngx_ssl_session_t *session);
#define ngx_ssl_get_session(c)      SSL_get1_session(c->ssl->connection)
#define ngx_ssl_free_session        SSL_SESSION_free
#define ngx_ssl_get_connection(ssl_conn)                                     \
    SSL_get_ex_data(ssl_conn, ngx_ssl_connection_index)
#define ngx_ssl_get_server_conf(ssl_ctx)                                     \
    SSL_CTX_get_ex_data(ssl_ctx, ngx_ssl_server_conf_index)

#define ngx_ssl_verify_error_optional(n)                                     \
    (n == X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT                             \
     || n == X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN                            \
     || n == X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY                    \
     || n == X509_V_ERR_CERT_UNTRUSTED                                       \
     || n == X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE)

ngx_int_t ngx_ssl_check_host(ngx_connection_t *c, ngx_str_t *name);


ngx_int_t ngx_ssl_get_protocol(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_cipher_name(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_session_id(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_session_reused(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_server_name(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_raw_certificate(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_certificate(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_subject_dn(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_issuer_dn(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_serial_number(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_fingerprint(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);
ngx_int_t ngx_ssl_get_client_verify(ngx_connection_t *c, ngx_pool_t *pool,
    ngx_str_t *s);


ngx_int_t ngx_ssl_handshake(ngx_connection_t *c);
ssize_t ngx_ssl_recv(ngx_connection_t *c, u_char *buf, size_t size);
ssize_t ngx_ssl_write(ngx_connection_t *c, u_char *data, size_t size);
ssize_t ngx_ssl_recv_chain(ngx_connection_t *c, ngx_chain_t *cl, off_t limit);
ngx_chain_t *ngx_ssl_send_chain(ngx_connection_t *c, ngx_chain_t *in,
    off_t limit);
void ngx_ssl_free_buffer(ngx_connection_t *c);
ngx_int_t ngx_ssl_shutdown(ngx_connection_t *c);
void ngx_cdecl ngx_ssl_error(ngx_uint_t level, ngx_log_t *log, ngx_err_t err,
    char *fmt, ...);
void ngx_ssl_cleanup_ctx(void *data);


extern int  ngx_ssl_connection_index;
extern int  ngx_ssl_server_conf_index;
extern int  ngx_ssl_session_cache_index;
extern int  ngx_ssl_session_ticket_keys_index;
extern int  ngx_ssl_certificate_index;
extern int  ngx_ssl_stapling_index;


#endif /* _NGX_EVENT_OPENSSL_H_INCLUDED_ */
```

One Level Up Top Level

src/event/ngx_event_openssl.c - nginx-1.7.10

Global variables defined

- ngx_openssl_commands
- ngx_openssl_module
- ngx_openssl_module_ctx
- ngx_ssl_certificate_index
- ngx_ssl_connection_index
- ngx_ssl_server_conf_index
- ngx_ssl_session_cache_index
- ngx_ssl_session_ticket_keys_index
- ngx_ssl_stapling_index

Data types defined

- ngx_openssl_conf_t

Functions defined

- ngx_openssl_create_conf
- ngx_openssl_engine
- ngx_openssl_exit
- ngx_ssl_certificate
- ngx_ssl_check_host
- ngx_ssl_check_name
- ngx_ssl_cleanup_ctx
- ngx_ssl_clear_error
- ngx_ssl_client_certificate
- ngx_ssl_connection_error
- ngx_ssl_create
- ngx_ssl_create_connection
- ngx_ssl_crl
- ngx_ssl_dhparam
- ngx_ssl_ecdh_curve
- ngx_ssl_error
- ngx_ssl_expire_sessions

- ngx_ssl_free_buffer
- ngx_ssl_get_cached_session
- ngx_ssl_get_certificate
- ngx_ssl_get_cipher_name
- ngx_ssl_get_client_verify
- ngx_ssl_get_fingerprint
- ngx_ssl_get_issuer_dn
- ngx_ssl_get_protocol
- ngx_ssl_get_raw_certificate
- ngx_ssl_get_serial_number
- ngx_ssl_get_server_name
- ngx_ssl_get_session_id
- ngx_ssl_get_session_reused
- ngx_ssl_get_subject_dn
- ngx_ssl_handle_recv
- ngx_ssl_handshake
- ngx_ssl_handshake_handler
- ngx_ssl_info_callback
- ngx_ssl_init
- ngx_ssl_new_session
- ngx_ssl_password_callback
- ngx_ssl_passwords_cleanup
- ngx_ssl_read_handler
- ngx_ssl_read_password_file
- ngx_ssl_recv
- ngx_ssl_recv_chain
- ngx_ssl_remove_cached_session
- ngx_ssl_remove_session
- ngx_ssl_rsa512_key_callback
- ngx_ssl_send_chain
- ngx_ssl_session_cache
- ngx_ssl_session_cache_init
- ngx_ssl_session_id_context

- ngx_ssl_session_rbtree_insert_value
- ngx_ssl_session_ticket_key_callback
- ngx_ssl_session_ticket_keys
- ngx_ssl_session_ticket_keys
- ngx_ssl_set_session
- ngx_ssl_shutdown
- ngx_ssl_shutdown_handler
- ngx_ssl_trusted_certificate
- ngx_ssl_verify_callback
- ngx_ssl_write
- ngx_ssl_write_handler

Macros defined

- NGX_SSL_PASSWORD_BUFFER_SIZE
- ngx_ssl_session_ticket_md
- ngx_ssl_session_ticket_md

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #define NGX_SSL_PASSWORD_BUFFER_SIZE  4096
14
15
16  typedef struct {
17      ngx_uint_t  engine;   /* unsigned  engine:1; */
18  } ngx_openssl_conf_t;
19
20
21  static int ngx_ssl_password_callback(char *buf, int size, int rwflag,
22      void *userdata);
23  static int ngx_ssl_verify_callback(int ok, X509_STORE_CTX *x509_store);
24  static void ngx_ssl_info_callback(const ngx_ssl_conn_t *ssl_conn, int where,
25      int ret);
26  static void ngx_ssl_passwords_cleanup(void *data);
27  static void ngx_ssl_handshake_handler(ngx_event_t *ev);
28  static ngx_int_t ngx_ssl_handle_recv(ngx_connection_t *c, int n);
29  static void ngx_ssl_write_handler(ngx_event_t *wev);
30  static void ngx_ssl_read_handler(ngx_event_t *rev);
31  static void ngx_ssl_shutdown_handler(ngx_event_t *ev);
32  static void ngx_ssl_connection_error(ngx_connection_t *c, int sslerr,
33      ngx_err_t err, char *text);
34  static void ngx_ssl_clear_error(ngx_log_t *log);
35
36  static ngx_int_t ngx_ssl_session_id_context(ngx_ssl_t *ssl,
37      ngx_str_t *sess_ctx);
```

```
ngx_int_t ngx_ssl_session_cache_init(ngx_shm_zone_t *shm_zone, void *data);
static int ngx_ssl_new_session(ngx_ssl_conn_t *ssl_conn,
    ngx_ssl_session_t *sess);
static ngx_ssl_session_t *ngx_ssl_get_cached_session(ngx_ssl_conn_t *ssl_conn,
    u_char *id, int len, int *copy);
static void ngx_ssl_remove_session(SSL_CTX *ssl, ngx_ssl_session_t *sess);
static void ngx_ssl_expire_sessions(ngx_ssl_session_cache_t *cache,
    ngx_slab_pool_t *shpool, ngx_uint_t n);
static void ngx_ssl_session_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);

#ifdef SSL_CTRL_SET_TLSEXT_TICKET_KEY_CB
static int ngx_ssl_session_ticket_key_callback(ngx_ssl_conn_t *ssl_conn,
    unsigned char *name, unsigned char *iv, EVP_CIPHER_CTX *ectx,
    HMAC_CTX *hctx, int enc);
#endif

#if (OPENSSL_VERSION_NUMBER < 0x10002002L || defined LIBRESSL_VERSION_NUMBER)
static ngx_int_t ngx_ssl_check_name(ngx_str_t *name, ASN1_STRING *str);
#endif

static void *ngx_openssl_create_conf(ngx_cycle_t *cycle);
static char *ngx_openssl_engine(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static void ngx_openssl_exit(ngx_cycle_t *cycle);


static ngx_command_t  ngx_openssl_commands[] = {

    { ngx_string("ssl_engine"),
      NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
      ngx_openssl_engine,
      0,
      0,
      NULL },

      ngx_null_command
};


static ngx_core_module_t  ngx_openssl_module_ctx = {
    ngx_string("openssl"),
    ngx_openssl_create_conf,
    NULL
};


ngx_module_t  ngx_openssl_module = {
    NGX_MODULE_V1,
    &ngx_openssl_module_ctx,               /* module context */
    ngx_openssl_commands,                  /* module directives */
    NGX_CORE_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    ngx_openssl_exit,                      /* exit master */
    NGX_MODULE_V1_PADDING
};


 int  ngx_ssl_connection_index;
 int  ngx_ssl_server_conf_index;
 int  ngx_ssl_session_cache_index;
 int  ngx_ssl_session_ticket_keys_index;
 int  ngx_ssl_certificate_index;
 int  ngx_ssl_stapling_index;


 ngx_int_t
 ngx_ssl_init(ngx_log_t *log)
 {
 #ifndef OPENSSL_IS_BORINGSSL
     OPENSSL_config(NULL);
 #endif
```

```
114
115     SSL_library_init();
116     SSL_load_error_strings();
117
118     OpenSSL_add_all_algorithms();
119
120 #if OPENSSL_VERSION_NUMBER >= 0x0090800fL
121 #ifndef SSL_OP_NO_COMPRESSION
122     {
123     /*
124      * Disable gzip compression in OpenSSL prior to 1.0.0 version,
125      * this saves about 522K per connection.
126      */
127     int                  n;
128     STACK_OF(SSL_COMP)  *ssl_comp_methods;
129
130     ssl_comp_methods = SSL_COMP_get_compression_methods();
131     n = sk_SSL_COMP_num(ssl_comp_methods);
132
133     while (n--) {
134         (void) sk_SSL_COMP_pop(ssl_comp_methods);
135     }
136     }
137 #endif
138 #endif
139
140     ngx_ssl_connection_index = SSL_get_ex_new_index(0, NULL, NULL, NULL, NULL);
141
142     if (ngx_ssl_connection_index == -1) {
143         ngx_ssl_error(NGX_LOG_ALERT, log, 0, "SSL_get_ex_new_index() failed");
144         return NGX_ERROR;
145     }
146
147     ngx_ssl_server_conf_index = SSL_CTX_get_ex_new_index(0, NULL, NULL, NULL,
148                                                          NULL);
149     if (ngx_ssl_server_conf_index == -1) {
150         ngx_ssl_error(NGX_LOG_ALERT, log, 0,
151                       "SSL_CTX_get_ex_new_index() failed");
152         return NGX_ERROR;
153     }
154
155     ngx_ssl_session_cache_index = SSL_CTX_get_ex_new_index(0, NULL, NULL, NULL,
156                                                            NULL);
157     if (ngx_ssl_session_cache_index == -1) {
158         ngx_ssl_error(NGX_LOG_ALERT, log, 0,
159                       "SSL_CTX_get_ex_new_index() failed");
160         return NGX_ERROR;
161     }
162
163     ngx_ssl_session_ticket_keys_index = SSL_CTX_get_ex_new_index(0, NULL, NULL,
164                                                                  NULL, NULL);
165     if (ngx_ssl_session_ticket_keys_index == -1) {
166         ngx_ssl_error(NGX_LOG_ALERT, log, 0,
167                       "SSL_CTX_get_ex_new_index() failed");
168         return NGX_ERROR;
169     }
170
171     ngx_ssl_certificate_index = SSL_CTX_get_ex_new_index(0, NULL, NULL, NULL,
172                                                          NULL);
173     if (ngx_ssl_certificate_index == -1) {
174         ngx_ssl_error(NGX_LOG_ALERT, log, 0,
175                       "SSL_CTX_get_ex_new_index() failed");
176         return NGX_ERROR;
177     }
178
179     ngx_ssl_stapling_index = SSL_CTX_get_ex_new_index(0, NULL, NULL, NULL,
180                                                       NULL);
181     if (ngx_ssl_stapling_index == -1) {
182         ngx_ssl_error(NGX_LOG_ALERT, log, 0,
183                       "SSL_CTX_get_ex_new_index() failed");
184         return NGX_ERROR;
185     }
186
187     return NGX_OK;
188 }
189
```

```
190
191 ngx_int_t
192 ngx_ssl_create(ngx_ssl_t *ssl, ngx_uint_t protocols, void *data)
193 {
194     ssl->ctx = SSL_CTX_new(SSLv23_method());
195
196     if (ssl->ctx == NULL) {
197         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0, "SSL_CTX_new() failed");
198         return NGX_ERROR;
199     }
200
201     if (SSL_CTX_set_ex_data(ssl->ctx, ngx_ssl_server_conf_index, data) == 0) {
202         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
203                       "SSL_CTX_set_ex_data() failed");
204         return NGX_ERROR;
205     }
206
207     ssl->buffer_size = NGX_SSL_BUFSIZE;
208
209     /* client side options */
210
211 #ifdef SSL_OP_MICROSOFT_SESS_ID_BUG
212     SSL_CTX_set_options(ssl->ctx, SSL_OP_MICROSOFT_SESS_ID_BUG);
213 #endif
214
215 #ifdef SSL_OP_NETSCAPE_CHALLENGE_BUG
216     SSL_CTX_set_options(ssl->ctx, SSL_OP_NETSCAPE_CHALLENGE_BUG);
217 #endif
218
219     /* server side options */
220
221 #ifdef SSL_OP_SSLREF2_REUSE_CERT_TYPE_BUG
222     SSL_CTX_set_options(ssl->ctx, SSL_OP_SSLREF2_REUSE_CERT_TYPE_BUG);
223 #endif
224
225 #ifdef SSL_OP_MICROSOFT_BIG_SSLV3_BUFFER
226     SSL_CTX_set_options(ssl->ctx, SSL_OP_MICROSOFT_BIG_SSLV3_BUFFER);
227 #endif
228
229 #ifdef SSL_OP_MSIE_SSLV2_RSA_PADDING
230     /* this option allow a potential SSL 2.0 rollback (CAN-2005-2969) */
231     SSL_CTX_set_options(ssl->ctx, SSL_OP_MSIE_SSLV2_RSA_PADDING);
232 #endif
233
234 #ifdef SSL_OP_SSLEAY_080_CLIENT_DH_BUG
235     SSL_CTX_set_options(ssl->ctx, SSL_OP_SSLEAY_080_CLIENT_DH_BUG);
236 #endif
237
238 #ifdef SSL_OP_TLS_D5_BUG
239     SSL_CTX_set_options(ssl->ctx, SSL_OP_TLS_D5_BUG);
240 #endif
241
242 #ifdef SSL_OP_TLS_BLOCK_PADDING_BUG
243     SSL_CTX_set_options(ssl->ctx, SSL_OP_TLS_BLOCK_PADDING_BUG);
244 #endif
245
246 #ifdef SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS
247     SSL_CTX_set_options(ssl->ctx, SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS);
248 #endif
249
250     SSL_CTX_set_options(ssl->ctx, SSL_OP_SINGLE_DH_USE);
251
252     if (!(protocols & NGX_SSL_SSLv2)) {
253         SSL_CTX_set_options(ssl->ctx, SSL_OP_NO_SSLv2);
254     }
255     if (!(protocols & NGX_SSL_SSLv3)) {
256         SSL_CTX_set_options(ssl->ctx, SSL_OP_NO_SSLv3);
257     }
258     if (!(protocols & NGX_SSL_TLSv1)) {
259         SSL_CTX_set_options(ssl->ctx, SSL_OP_NO_TLSv1);
260     }
261 #ifdef SSL_OP_NO_TLSv1_1
262     if (!(protocols & NGX_SSL_TLSv1_1)) {
263         SSL_CTX_set_options(ssl->ctx, SSL_OP_NO_TLSv1_1);
264     }
265 #endif
```

```
#ifdef SSL_OP_NO_TLSv1_2
    if (!(protocols & NGX_SSL_TLSv1_2)) {
        SSL_CTX_set_options(ssl->ctx, SSL_OP_NO_TLSv1_2);
    }
#endif

#ifdef SSL_OP_NO_COMPRESSION
    SSL_CTX_set_options(ssl->ctx, SSL_OP_NO_COMPRESSION);
#endif

#ifdef SSL_MODE_RELEASE_BUFFERS
    SSL_CTX_set_mode(ssl->ctx, SSL_MODE_RELEASE_BUFFERS);
#endif

    SSL_CTX_set_read_ahead(ssl->ctx, 1);

    SSL_CTX_set_info_callback(ssl->ctx, ngx_ssl_info_callback);

    return NGX_OK;
}


ngx_int_t
ngx_ssl_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *cert,
    ngx_str_t *key, ngx_array_t *passwords)
{
    BIO         *bio;
    X509        *x509;
    u_long       n;
    ngx_str_t   *pwd;
    ngx_uint_t   tries;

    if (ngx_conf_full_name(cf->cycle, cert, 1) != NGX_OK) {
        return NGX_ERROR;
    }

    /*
     * we can't use SSL_CTX_use_certificate_chain_file() as it doesn't
     * allow to access certificate later from SSL_CTX, so we reimplement
     * it here
     */

    bio = BIO_new_file((char *) cert->data, "r");
    if (bio == NULL) {
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "BIO_new_file(\"%s\") failed", cert->data);
        return NGX_ERROR;
    }

    x509 = PEM_read_bio_X509_AUX(bio, NULL, NULL, NULL);
    if (x509 == NULL) {
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "PEM_read_bio_X509_AUX(\"%s\") failed", cert->data);
        BIO_free(bio);
        return NGX_ERROR;
    }

    if (SSL_CTX_use_certificate(ssl->ctx, x509) == 0) {
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "SSL_CTX_use_certificate(\"%s\") failed", cert->data);
        X509_free(x509);
        BIO_free(bio);
        return NGX_ERROR;
    }

    if (SSL_CTX_set_ex_data(ssl->ctx, ngx_ssl_certificate_index, x509)
        == 0)
    {
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "SSL_CTX_set_ex_data() failed");
        X509_free(x509);
        BIO_free(bio);
        return NGX_ERROR;
    }

    X509_free(x509);
```

```
342
343     /* read rest of the chain */
344
345     for ( ;; ) {
346
347         x509 = PEM_read_bio_X509(bio, NULL, NULL, NULL);
348         if (x509 == NULL) {
349             n = ERR_peek_last_error();
350
351             if (ERR_GET_LIB(n) == ERR_LIB_PEM
352                 && ERR_GET_REASON(n) == PEM_R_NO_START_LINE)
353             {
354                 /* end of file */
355                 ERR_clear_error();
356                 break;
357             }
358
359             /* some real error */
360
361             ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
362                           "PEM_read_bio_X509(\"%s\") failed", cert->data);
363             BIO_free(bio);
364             return NGX_ERROR;
365         }
366
367         if (SSL_CTX_add_extra_chain_cert(ssl->ctx, x509) == 0) {
368             ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
369                           "SSL_CTX_add_extra_chain_cert(\"%s\") failed",
370                           cert->data);
371             X509_free(x509);
372             BIO_free(bio);
373             return NGX_ERROR;
374         }
375     }
376
377     BIO_free(bio);
378
379     if (ngx_strncmp(key->data, "engine:", sizeof("engine:") - 1) == 0) {
380
381 #ifndef OPENSSL_NO_ENGINE
382
383         u_char      *p, *last;
384         ENGINE      *engine;
385         EVP_PKEY    *pkey;
386
387         p = key->data + sizeof("engine:") - 1;
388         last = (u_char *) ngx_strchr(p, ':');
389
390         if (last == NULL) {
391             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
392                                "invalid syntax in \"%V\"", key);
393             return NGX_ERROR;
394         }
395
396         *last = '\0';
397
398         engine = ENGINE_by_id((char *) p);
399
400         if (engine == NULL) {
401             ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
402                           "ENGINE_by_id(\"%s\") failed", p);
403             return NGX_ERROR;
404         }
405
406         *last++ = ':';
407
408         pkey = ENGINE_load_private_key(engine, (char *) last, 0, 0);
409
410         if (pkey == NULL) {
411             ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
412                           "ENGINE_load_private_key(\"%s\") failed", last);
413             ENGINE_free(engine);
414             return NGX_ERROR;
415         }
416
417         ENGINE_free(engine);
```

```
418
419         if (SSL_CTX_use_PrivateKey(ssl->ctx, pkey) == 0) {
420             ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
421                           "SSL_CTX_use_PrivateKey(\"%s\") failed", last);
422             EVP_PKEY_free(pkey);
423             return NGX_ERROR;
424         }
425
426         EVP_PKEY_free(pkey);
427
428         return NGX_OK;
429
430 #else
431
432         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
433                            "loading \"engine:...\" certificate keys "
434                            "is not supported");
435         return NGX_ERROR;
436
437 #endif
438     }
439
440     if (ngx_conf_full_name(cf->cycle, key, 1) != NGX_OK) {
441         return NGX_ERROR;
442     }
443
444     if (passwords) {
445         tries = passwords->nelts;
446         pwd = passwords->elts;
447
448         SSL_CTX_set_default_passwd_cb(ssl->ctx, ngx_ssl_password_callback);
449         SSL_CTX_set_default_passwd_cb_userdata(ssl->ctx, pwd);
450
451     } else {
452         tries = 1;
453 #if (NGX_SUPPRESS_WARN)
454         pwd = NULL;
455 #endif
456     }
457
458     for ( ;; ) {
459
460         if (SSL_CTX_use_PrivateKey_file(ssl->ctx, (char *) key->data,
461                                         SSL_FILETYPE_PEM)
462             != 0)
463         {
464             break;
465         }
466
467         if (--tries) {
468             ERR_clear_error();
469             SSL_CTX_set_default_passwd_cb_userdata(ssl->ctx, ++pwd);
470             continue;
471         }
472
473         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
474                       "SSL_CTX_use_PrivateKey_file(\"%s\") failed", key->data);
475         return NGX_ERROR;
476     }
477
478     SSL_CTX_set_default_passwd_cb(ssl->ctx, NULL);
479
480     return NGX_OK;
481 }
482
483
484 static int
485 ngx_ssl_password_callback(char *buf, int size, int rwflag, void *userdata)
486 {
487     ngx_str_t *pwd = userdata;
488
489     if (rwflag) {
490         ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
491                       "ngx_ssl_password_callback() is called for encryption");
492         return 0;
493     }
```

```
494
495     if (pwd->len > (size_t) size) {
496         ngx_log_error(NGX_LOG_ERR, ngx_cycle->log, 0,
497                       "password is truncated to %d bytes", size);
498     } else {
499         size = pwd->len;
500     }
501
502     ngx_memcpy(buf, pwd->data, size);
503
504     return size;
505 }
506
507
508 ngx_int_t
509 ngx_ssl_client_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *cert,
510     ngx_int_t depth)
511 {
512     STACK_OF(X509_NAME)  *list;
513
514     SSL_CTX_set_verify(ssl->ctx, SSL_VERIFY_PEER, ngx_ssl_verify_callback);
515
516     SSL_CTX_set_verify_depth(ssl->ctx, depth);
517
518     if (cert->len == 0) {
519         return NGX_OK;
520     }
521
522     if (ngx_conf_full_name(cf->cycle, cert, 1) != NGX_OK) {
523         return NGX_ERROR;
524     }
525
526     if (SSL_CTX_load_verify_locations(ssl->ctx, (char *) cert->data, NULL)
527         == 0)
528     {
529         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
530                       "SSL_CTX_load_verify_locations(\"%s\") failed",
531                       cert->data);
532         return NGX_ERROR;
533     }
534
535     /*
536      * SSL_CTX_load_verify_locations() may leave errors in the error queue
537      * while returning success
538      */
539
540     ERR_clear_error();
541
542     list = SSL_load_client_CA_file((char *) cert->data);
543
544     if (list == NULL) {
545         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
546                       "SSL_load_client_CA_file(\"%s\") failed", cert->data);
547         return NGX_ERROR;
548     }
549
550     /*
551      * before 0.9.7h and 0.9.8 SSL_load_client_CA_file()
552      * always leaved an error in the error queue
553      */
554
555     ERR_clear_error();
556
557     SSL_CTX_set_client_CA_list(ssl->ctx, list);
558
559     return NGX_OK;
560 }
561
562
563 ngx_int_t
564 ngx_ssl_trusted_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *cert,
565     ngx_int_t depth)
566 {
567     SSL_CTX_set_verify_depth(ssl->ctx, depth);
568
569     if (cert->len == 0) {
```

```
570         return NGX_OK;
571     }
572 
573     if (ngx_conf_full_name(cf->cycle, cert, 1) != NGX_OK) {
574         return NGX_ERROR;
575     }
576 
577     if (SSL_CTX_load_verify_locations(ssl->ctx, (char *) cert->data, NULL)
578         == 0)
579     {
580         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
581                       "SSL_CTX_load_verify_locations(\"%s\") failed",
582                       cert->data);
583         return NGX_ERROR;
584     }
585 
586     /*
587      * SSL_CTX_load_verify_locations() may leave errors in the error queue
588      * while returning success
589      */
590 
591     ERR_clear_error();
592 
593     return NGX_OK;
594 }
595 
596 
597 ngx_int_t
598 ngx_ssl_crl(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *crl)
599 {
600     X509_STORE   *store;
601     X509_LOOKUP  *lookup;
602 
603     if (crl->len == 0) {
604         return NGX_OK;
605     }
606 
607     if (ngx_conf_full_name(cf->cycle, crl, 1) != NGX_OK) {
608         return NGX_ERROR;
609     }
610 
611     store = SSL_CTX_get_cert_store(ssl->ctx);
612 
613     if (store == NULL) {
614         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
615                       "SSL_CTX_get_cert_store() failed");
616         return NGX_ERROR;
617     }
618 
619     lookup = X509_STORE_add_lookup(store, X509_LOOKUP_file());
620 
621     if (lookup == NULL) {
622         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
623                       "X509_STORE_add_lookup() failed");
624         return NGX_ERROR;
625     }
626 
627     if (X509_LOOKUP_load_file(lookup, (char *) crl->data, X509_FILETYPE_PEM)
628         == 0)
629     {
630         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
631                       "X509_LOOKUP_load_file(\"%s\") failed", crl->data);
632         return NGX_ERROR;
633     }
634 
635     X509_STORE_set_flags(store,
636                          X509_V_FLAG_CRL_CHECK|X509_V_FLAG_CRL_CHECK_ALL);
637 
638     return NGX_OK;
639 }
640 
641 
642 static int
643 ngx_ssl_verify_callback(int ok, X509_STORE_CTX *x509_store)
644 {
645 #if (NGX_DEBUG)
```

```
    char               *subject, *issuer;
    int                 err, depth;
    X509               *cert;
    X509_NAME          *sname, *iname;
    ngx_connection_t   *c;
    ngx_ssl_conn_t     *ssl_conn;

    ssl_conn = X509_STORE_CTX_get_ex_data(x509_store,
                                          SSL_get_ex_data_X509_STORE_CTX_idx());

    c = ngx_ssl_get_connection(ssl_conn);

    cert = X509_STORE_CTX_get_current_cert(x509_store);
    err = X509_STORE_CTX_get_error(x509_store);
    depth = X509_STORE_CTX_get_error_depth(x509_store);

    sname = X509_get_subject_name(cert);
    subject = sname ? X509_NAME_oneline(sname, NULL, 0) : "(none)";

    iname = X509_get_issuer_name(cert);
    issuer = iname ? X509_NAME_oneline(iname, NULL, 0) : "(none)";

    ngx_log_debug5(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "verify:%d, error:%d, depth:%d, "
                   "subject:\"%s\", issuer:\"%s\"",
                   ok, err, depth, subject, issuer);

    if (sname) {
        OPENSSL_free(subject);
    }

    if (iname) {
        OPENSSL_free(issuer);
    }
#endif

    return 1;
}


static void
ngx_ssl_info_callback(const ngx_ssl_conn_t *ssl_conn, int where, int ret)
{
    BIO               *rbio, *wbio;
    ngx_connection_t  *c;

    if (where & SSL_CB_HANDSHAKE_START) {
        c = ngx_ssl_get_connection((ngx_ssl_conn_t *) ssl_conn);

        if (c->ssl->handshaked) {
            c->ssl->renegotiation = 1;
            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL renegotiation");
        }
    }

    if ((where & SSL_CB_ACCEPT_LOOP) == SSL_CB_ACCEPT_LOOP) {
        c = ngx_ssl_get_connection((ngx_ssl_conn_t *) ssl_conn);

        if (!c->ssl->handshake_buffer_set) {
            /*
             * By default OpenSSL uses 4k buffer during a handshake,
             * which is too low for long certificate chains and might
             * result in extra round-trips.
             *
             * To adjust a buffer size we detect that buffering was added
             * to write side of the connection by comparing rbio and wbio.
             * If they are different, we assume that it's due to buffering
             * added to wbio, and set buffer size.
             */

            rbio = SSL_get_rbio((ngx_ssl_conn_t *) ssl_conn);
            wbio = SSL_get_wbio((ngx_ssl_conn_t *) ssl_conn);

            if (rbio != wbio) {
                (void) BIO_set_write_buffer_size(wbio, NGX_SSL_BUFSIZE);
                c->ssl->handshake_buffer_set = 1;
```

```
            }
        }
    }
}


RSA *
ngx_ssl_rsa512_key_callback(ngx_ssl_conn_t *ssl_conn, int is_export,
    int key_length)
{
    static RSA  *key;

    if (key_length != 512) {
        return NULL;
    }

#ifndef OPENSSL_NO_DEPRECATED

    if (key == NULL) {
        key = RSA_generate_key(512, RSA_F4, NULL, NULL);
    }

#endif

    return key;
}


ngx_array_t *
ngx_ssl_read_password_file(ngx_conf_t *cf, ngx_str_t *file)
{
    u_char              *p, *last, *end;
    size_t               len;
    ssize_t              n;
    ngx_fd_t             fd;
    ngx_str_t           *pwd;
    ngx_array_t         *passwords;
    ngx_pool_cleanup_t  *cln;
    u_char               buf[NGX_SSL_PASSWORD_BUFFER_SIZE];

    if (ngx_conf_full_name(cf->cycle, file, 1) != NGX_OK) {
        return NULL;
    }

    cln = ngx_pool_cleanup_add(cf->temp_pool, 0);
    passwords = ngx_array_create(cf->temp_pool, 4, sizeof(ngx_str_t));

    if (cln == NULL || passwords == NULL) {
        return NULL;
    }

    cln->handler = ngx_ssl_passwords_cleanup;
    cln->data = passwords;

    fd = ngx_open_file(file->data, NGX_FILE_RDONLY, NGX_FILE_OPEN, 0);
    if (fd == NGX_INVALID_FILE) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
                           ngx_open_file_n " \"%s\" failed", file->data);
        return NULL;
    }

    len = 0;
    last = buf;

    do {
        n = ngx_read_fd(fd, last, NGX_SSL_PASSWORD_BUFFER_SIZE - len);

        if (n == -1) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
                               ngx_read_fd_n " \"%s\" failed", file->data);
            passwords = NULL;
            goto cleanup;
        }

        end = last + n;

```

```
        if (len && n == 0) {
            *end++ = LF;
        }

        p = buf;

        for ( ;; ) {
            last = ngx_strlchr(last, end, LF);

            if (last == NULL) {
                break;
            }

            len = last++ - p;

            if (len && p[len - 1] == CR) {
                len--;
            }

            if (len) {
                pwd = ngx_array_push(passwords);
                if (pwd == NULL) {
                    passwords = NULL;
                    goto cleanup;
                }

                pwd->len = len;
                pwd->data = ngx_pnalloc(cf->temp_pool, len);

                if (pwd->data == NULL) {
                    passwords->nelts--;
                    passwords = NULL;
                    goto cleanup;
                }

                ngx_memcpy(pwd->data, p, len);
            }

            p = last;
        }

        len = end - p;

        if (len == NGX_SSL_PASSWORD_BUFFER_SIZE) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "too long line in \"%s\"", file->data);
            passwords = NULL;
            goto cleanup;
        }

        ngx_memmove(buf, p, len);
        last = buf + len;

    } while (n != 0);

    if (passwords->nelts == 0) {
        pwd = ngx_array_push(passwords);
        if (pwd == NULL) {
            passwords = NULL;
            goto cleanup;
        }

        ngx_memzero(pwd, sizeof(ngx_str_t));
    }

cleanup:

    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
        ngx_conf_log_error(NGX_LOG_ALERT, cf, ngx_errno,
                           ngx_close_file_n " \"%s\" failed", file->data);
    }

    ngx_memzero(buf, NGX_SSL_PASSWORD_BUFFER_SIZE);

    return passwords;
}
```

```c


static void
ngx_ssl_passwords_cleanup(void *data)
{
    ngx_array_t *passwords = data;

    ngx_str_t   *pwd;
    ngx_uint_t   i;

    pwd = passwords->elts;

    for (i = 0; i < passwords->nelts; i++) {
        ngx_memzero(pwd[i].data, pwd[i].len);
    }
}


ngx_int_t
ngx_ssl_dhparam(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *file)
{
    DH   *dh;
    BIO  *bio;

    /*
     * -----BEGIN DH PARAMETERS-----
     * MIGHAoGBALu8LcrYRnSQfEP89YDpz9vZWKP1aLQtSwju1OsPs1BMbAMCducQgAxc
     * y7qokiYUxb7spWWl/fHSh6K8BJvmd4Bg6RqSp1fjBI9osHb302zI8pul34HcLKcl
     * 7OZicMyaUDXYzs7vnqAnSmOrHlj6/UmI0PZdFGdX2gcd8EXP4WubAgEC
     * -----END DH PARAMETERS-----
     */

    static unsigned char dh1024_p[] = {
        0xBB, 0xBC, 0x2D, 0xCA, 0xD8, 0x46, 0x74, 0x90, 0x7C, 0x43, 0xFC, 0xF5,
        0x80, 0xE9, 0xCF, 0xDB, 0xD9, 0x58, 0xA3, 0xF5, 0x68, 0xB4, 0x2D, 0x4B,
        0x08, 0xEE, 0xD4, 0xEB, 0x0F, 0xB3, 0x50, 0x4C, 0x6C, 0x03, 0x02, 0x76,
        0xE7, 0x10, 0x80, 0x0C, 0x5C, 0xCB, 0xBA, 0xA8, 0x92, 0x26, 0x14, 0xC5,
        0xBE, 0xEC, 0xA5, 0x65, 0xA5, 0xFD, 0xF1, 0xD2, 0x87, 0xA2, 0xBC, 0x04,
        0x9B, 0xE6, 0x77, 0x80, 0x60, 0xE9, 0x1A, 0x92, 0xA7, 0x57, 0xE3, 0x04,
        0x8F, 0x68, 0xB0, 0x76, 0xF7, 0xD3, 0x6C, 0xC8, 0xF2, 0x9B, 0xA5, 0xDF,
        0x81, 0xDC, 0x2C, 0xA7, 0x25, 0xEC, 0xE6, 0x62, 0x70, 0xCC, 0x9A, 0x50,
        0x35, 0xD8, 0xCE, 0xCE, 0xEF, 0x9E, 0xA0, 0x27, 0x4A, 0x63, 0xAB, 0x1E,
        0x58, 0xFA, 0xFD, 0x49, 0x88, 0xD0, 0xF6, 0x5D, 0x14, 0x67, 0x57, 0xDA,
        0x07, 0x1D, 0xF0, 0x45, 0xCF, 0xE1, 0x6B, 0x9B
    };

    static unsigned char dh1024_g[] = { 0x02 };


    if (file->len == 0) {

        dh = DH_new();
        if (dh == NULL) {
            ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0, "DH_new() failed");
            return NGX_ERROR;
        }

        dh->p = BN_bin2bn(dh1024_p, sizeof(dh1024_p), NULL);
        dh->g = BN_bin2bn(dh1024_g, sizeof(dh1024_g), NULL);

        if (dh->p == NULL || dh->g == NULL) {
            ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0, "BN_bin2bn() failed");
            DH_free(dh);
            return NGX_ERROR;
        }

        SSL_CTX_set_tmp_dh(ssl->ctx, dh);

        DH_free(dh);

        return NGX_OK;
    }

    if (ngx_conf_full_name(cf->cycle, file, 1) != NGX_OK) {
        return NGX_ERROR;
    }
```

```
950
951     bio = BIO_new_file((char *) file->data, "r");
952     if (bio == NULL) {
953         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
954                       "BIO_new_file(\"%s\") failed", file->data);
955         return NGX_ERROR;
956     }
957
958     dh = PEM_read_bio_DHparams(bio, NULL, NULL, NULL);
959     if (dh == NULL) {
960         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
961                       "PEM_read_bio_DHparams(\"%s\") failed", file->data);
962         BIO_free(bio);
963         return NGX_ERROR;
964     }
965
966     SSL_CTX_set_tmp_dh(ssl->ctx, dh);
967
968     DH_free(dh);
969     BIO_free(bio);
970
971     return NGX_OK;
972 }
973
974
975 ngx_int_t
976 ngx_ssl_ecdh_curve(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *name)
977 {
978 #if OPENSSL_VERSION_NUMBER >= 0x0090800fL
979 #ifndef OPENSSL_NO_ECDH
980     int      nid;
981     EC_KEY  *ecdh;
982
983     /*
984      * Elliptic-Curve Diffie-Hellman parameters are either "named curves"
985      * from RFC 4492 section 5.1.1, or explicitly described curves over
986      * binary fields. OpenSSL only supports the "named curves", which provide
987      * maximum interoperability.
988      */
989
990     nid = OBJ_sn2nid((const char *) name->data);
991     if (nid == 0) {
992         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
993                       "Unknown curve name \"%s\"", name->data);
994         return NGX_ERROR;
995     }
996
997     ecdh = EC_KEY_new_by_curve_name(nid);
998     if (ecdh == NULL) {
999         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
1000                       "Unable to create curve \"%s\"", name->data);
1001         return NGX_ERROR;
1002     }
1003
1004     SSL_CTX_set_options(ssl->ctx, SSL_OP_SINGLE_ECDH_USE);
1005
1006     SSL_CTX_set_tmp_ecdh(ssl->ctx, ecdh);
1007
1008     EC_KEY_free(ecdh);
1009 #endif
1010 #endif
1011
1012     return NGX_OK;
1013 }
1014
1015
1016 ngx_int_t
1017 ngx_ssl_create_connection(ngx_ssl_t *ssl, ngx_connection_t *c, ngx_uint_t flags)
1018 {
1019     ngx_ssl_connection_t  *sc;
1020
1021     sc = ngx_pcalloc(c->pool, sizeof(ngx_ssl_connection_t));
1022     if (sc == NULL) {
1023         return NGX_ERROR;
1024     }
1025
```

```
    sc->buffer = ((flags & NGX_SSL_BUFFER) != 0);
    sc->buffer_size = ssl->buffer_size;

    sc->connection = SSL_new(ssl->ctx);

    if (sc->connection == NULL) {
        ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "SSL_new() failed");
        return NGX_ERROR;
    }

    if (SSL_set_fd(sc->connection, c->fd) == 0) {
        ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "SSL_set_fd() failed");
        return NGX_ERROR;
    }

    if (flags & NGX_SSL_CLIENT) {
        SSL_set_connect_state(sc->connection);

    } else {
        SSL_set_accept_state(sc->connection);
    }

    if (SSL_set_ex_data(sc->connection, ngx_ssl_connection_index, c) == 0) {
        ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "SSL_set_ex_data() failed");
        return NGX_ERROR;
    }

    c->ssl = sc;

    return NGX_OK;
}


ngx_int_t
ngx_ssl_set_session(ngx_connection_t *c, ngx_ssl_session_t *session)
{
    if (session) {
        if (SSL_set_session(c->ssl->connection, session) == 0) {
            ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "SSL_set_session() failed");
            return NGX_ERROR;
        }
    }

    return NGX_OK;
}


ngx_int_t
ngx_ssl_handshake(ngx_connection_t *c)
{
    int        n, sslerr;
    ngx_err_t  err;

    ngx_ssl_clear_error(c->log);

    n = SSL_do_handshake(c->ssl->connection);

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_do_handshake: %d", n);

    if (n == 1) {

        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            return NGX_ERROR;
        }

        if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
            return NGX_ERROR;
        }

#if (NGX_DEBUG)
        {
        char         buf[129], *s, *d;
#if OPENSSL_VERSION_NUMBER >= 0x10000000L
        const
#endif
        SSL_CIPHER  *cipher;
```

```
1102
1103            cipher = SSL_get_current_cipher(c->ssl->connection);
1104
1105            if (cipher) {
1106                SSL_CIPHER_description(cipher, &buf[1], 128);
1107
1108                for (s = &buf[1], d = buf; *s; s++) {
1109                    if (*s == ' ' && *d == ' ') {
1110                        continue;
1111                    }
1112
1113                    if (*s == LF || *s == CR) {
1114                        continue;
1115                    }
1116
1117                    *++d = *s;
1118                }
1119
1120                if (*d != ' ') {
1121                    d++;
1122                }
1123
1124                *d = '\0';
1125
1126                ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
1127                               "SSL: %s, cipher: \"%s\"",
1128                               SSL_get_version(c->ssl->connection), &buf[1]);
1129
1130                if (SSL_session_reused(c->ssl->connection)) {
1131                    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
1132                                   "SSL reused session");
1133                }
1134
1135            } else {
1136                ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
1137                               "SSL no shared ciphers");
1138            }
1139            }
1140    #endif
1141
1142            c->ssl->handshaked = 1;
1143
1144            c->recv = ngx_ssl_recv;
1145            c->send = ngx_ssl_write;
1146            c->recv_chain = ngx_ssl_recv_chain;
1147            c->send_chain = ngx_ssl_send_chain;
1148
1149    #ifdef SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS
1150
1151            /* initial handshake done, disable renegotiation (CVE-2009-3555) */
1152            if (c->ssl->connection->s3) {
1153                c->ssl->connection->s3->flags |= SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS;
1154            }
1155
1156    #endif
1157
1158            return NGX_OK;
1159        }
1160
1161        sslerr = SSL_get_error(c->ssl->connection, n);
1162
1163        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_get_error: %d", sslerr);
1164
1165        if (sslerr == SSL_ERROR_WANT_READ) {
1166            c->read->ready = 0;
1167            c->read->handler = ngx_ssl_handshake_handler;
1168            c->write->handler = ngx_ssl_handshake_handler;
1169
1170            if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
1171                return NGX_ERROR;
1172            }
1173
1174            if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
1175                return NGX_ERROR;
1176            }
1177
```

```
        return NGX_AGAIN;
    }

    if (sslerr == SSL_ERROR_WANT_WRITE) {
        c->write->ready = 0;
        c->read->handler = ngx_ssl_handshake_handler;
        c->write->handler = ngx_ssl_handshake_handler;

        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            return NGX_ERROR;
        }

        if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
            return NGX_ERROR;
        }

        return NGX_AGAIN;
    }

    err = (sslerr == SSL_ERROR_SYSCALL) ? ngx_errno : 0;

    c->ssl->no_wait_shutdown = 1;
    c->ssl->no_send_shutdown = 1;
    c->read->eof = 1;

    if (sslerr == SSL_ERROR_ZERO_RETURN || ERR_peek_error() == 0) {
        ngx_connection_error(c, err,
                             "peer closed connection in SSL handshake");

        return NGX_ERROR;
    }

    c->read->error = 1;

    ngx_ssl_connection_error(c, sslerr, err, "SSL_do_handshake() failed");

    return NGX_ERROR;
}


static void
ngx_ssl_handshake_handler(ngx_event_t *ev)
{
    ngx_connection_t  *c;

    c = ev->data;

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "SSL handshake handler: %d", ev->write);

    if (ev->timedout) {
        c->ssl->handler(c);
        return;
    }

    if (ngx_ssl_handshake(c) == NGX_AGAIN) {
        return;
    }

    c->ssl->handler(c);
}


ssize_t
ngx_ssl_recv_chain(ngx_connection_t *c, ngx_chain_t *cl, off_t limit)
{
    u_char     *last;
    ssize_t     n, bytes, size;
    ngx_buf_t  *b;

    bytes = 0;

    b = cl->buf;
    last = b->last;

    for ( ;; ) {
```

```
        size = b->end - last;

        if (limit) {
            if (bytes >= limit) {
                return bytes;
            }

            if (bytes + size > limit) {
                size = (ssize_t) (limit - bytes);
            }
        }

        n = ngx_ssl_recv(c, last, size);

        if (n > 0) {
            last += n;
            bytes += n;

            if (last == b->end) {
                cl = cl->next;

                if (cl == NULL) {
                    return bytes;
                }

                b = cl->buf;
                last = b->last;
            }

            continue;
        }

        if (bytes) {

            if (n == 0 || n == NGX_ERROR) {
                c->read->ready = 1;
            }

            return bytes;
        }

        return n;
    }
}


ssize_t
ngx_ssl_recv(ngx_connection_t *c, u_char *buf, size_t size)
{
    int  n, bytes;

    if (c->ssl->last == NGX_ERROR) {
        c->read->error = 1;
        return NGX_ERROR;
    }

    if (c->ssl->last == NGX_DONE) {
        c->read->ready = 0;
        c->read->eof = 1;
        return 0;
    }

    bytes = 0;

    ngx_ssl_clear_error(c->log);

    /*
     * SSL_read() may return data in parts, so try to read
     * until SSL_read() would return no data
     */

    for ( ;; ) {

        n = SSL_read(c->ssl->connection, buf, size);

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_read: %d", n);
```

```
1330
1331        if (n > 0) {
1332            bytes += n;
1333        }
1334
1335        c->ssl->last = ngx_ssl_handle_recv(c, n);
1336
1337        if (c->ssl->last == NGX_OK) {
1338
1339            size -= n;
1340
1341            if (size == 0) {
1342                c->read->ready = 1;
1343                return bytes;
1344            }
1345
1346            buf += n;
1347
1348            continue;
1349        }
1350
1351        if (bytes) {
1352            if (c->ssl->last != NGX_AGAIN) {
1353                c->read->ready = 1;
1354            }
1355
1356            return bytes;
1357        }
1358
1359        switch (c->ssl->last) {
1360
1361        case NGX_DONE:
1362            c->read->ready = 0;
1363            c->read->eof = 1;
1364            return 0;
1365
1366        case NGX_ERROR:
1367            c->read->error = 1;
1368
1369            /* fall through */
1370
1371        case NGX_AGAIN:
1372            return c->ssl->last;
1373        }
1374    }
1375 }
1376
1377
1378 static ngx_int_t
1379 ngx_ssl_handle_recv(ngx_connection_t *c, int n)
1380 {
1381     int        sslerr;
1382     ngx_err_t  err;
1383
1384     if (c->ssl->renegotiation) {
1385         /*
1386          * disable renegotiation (CVE-2009-3555):
1387          * OpenSSL (at least up to 0.9.8l) does not handle disabled
1388          * renegotiation gracefully, so drop connection here
1389          */
1390
1391         ngx_log_error(NGX_LOG_NOTICE, c->log, 0, "SSL renegotiation disabled");
1392
1393         while (ERR_peek_error()) {
1394             ngx_ssl_error(NGX_LOG_DEBUG, c->log, 0,
1395                           "ignoring stale global SSL error");
1396         }
1397
1398         ERR_clear_error();
1399
1400         c->ssl->no_wait_shutdown = 1;
1401         c->ssl->no_send_shutdown = 1;
1402
1403         return NGX_ERROR;
1404     }
1405
```

```
1406      if (n > 0) {
1407
1408          if (c->ssl->saved_write_handler) {
1409
1410              c->write->handler = c->ssl->saved_write_handler;
1411              c->ssl->saved_write_handler = NULL;
1412              c->write->ready = 1;
1413
1414              if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
1415                  return NGX_ERROR;
1416              }
1417
1418              ngx_post_event(c->write, &ngx_posted_events);
1419          }
1420
1421          return NGX_OK;
1422      }
1423
1424      sslerr = SSL_get_error(c->ssl->connection, n);
1425
1426      err = (sslerr == SSL_ERROR_SYSCALL) ? ngx_errno : 0;
1427
1428      ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_get_error: %d", sslerr);
1429
1430      if (sslerr == SSL_ERROR_WANT_READ) {
1431          c->read->ready = 0;
1432          return NGX_AGAIN;
1433      }
1434
1435      if (sslerr == SSL_ERROR_WANT_WRITE) {
1436
1437          ngx_log_error(NGX_LOG_INFO, c->log, 0,
1438                        "peer started SSL renegotiation");
1439
1440          c->write->ready = 0;
1441
1442          if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
1443              return NGX_ERROR;
1444          }
1445
1446          /*
1447           * we do not set the timer because there is already the read event timer
1448           */
1449
1450          if (c->ssl->saved_write_handler == NULL) {
1451              c->ssl->saved_write_handler = c->write->handler;
1452              c->write->handler = ngx_ssl_write_handler;
1453          }
1454
1455          return NGX_AGAIN;
1456      }
1457
1458      c->ssl->no_wait_shutdown = 1;
1459      c->ssl->no_send_shutdown = 1;
1460
1461      if (sslerr == SSL_ERROR_ZERO_RETURN || ERR_peek_error() == 0) {
1462          ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
1463                         "peer shutdown SSL cleanly");
1464          return NGX_DONE;
1465      }
1466
1467      ngx_ssl_connection_error(c, sslerr, err, "SSL_read() failed");
1468
1469      return NGX_ERROR;
1470  }
1471
1472
1473  static void
1474  ngx_ssl_write_handler(ngx_event_t *wev)
1475  {
1476      ngx_connection_t  *c;
1477
1478      c = wev->data;
1479
1480      c->read->handler(c->read);
1481  }
```

```c


/*
 * OpenSSL has no SSL_writev() so we copy several bufs into our 16K buffer
 * before the SSL_write() call to decrease a SSL overhead.
 *
 * Besides for protocols such as HTTP it is possible to always buffer
 * the output to decrease a SSL overhead some more.
 */

ngx_chain_t *
ngx_ssl_send_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
{
    int          n;
    ngx_uint_t   flush;
    ssize_t      send, size;
    ngx_buf_t   *buf;

    if (!c->ssl->buffer) {

        while (in) {
            if (ngx_buf_special(in->buf)) {
                in = in->next;
                continue;
            }

            n = ngx_ssl_write(c, in->buf->pos, in->buf->last - in->buf->pos);

            if (n == NGX_ERROR) {
                return NGX_CHAIN_ERROR;
            }

            if (n == NGX_AGAIN) {
                return in;
            }

            in->buf->pos += n;
            c->sent += n;

            if (in->buf->pos == in->buf->last) {
                in = in->next;
            }
        }

        return in;
    }


    /* the maximum limit size is the maximum int32_t value - the page size */

    if (limit == 0 || limit > (off_t) (NGX_MAX_INT32_VALUE - ngx_pagesize)) {
        limit = NGX_MAX_INT32_VALUE - ngx_pagesize;
    }

    buf = c->ssl->buf;

    if (buf == NULL) {
        buf = ngx_create_temp_buf(c->pool, c->ssl->buffer_size);
        if (buf == NULL) {
            return NGX_CHAIN_ERROR;
        }

        c->ssl->buf = buf;
    }

    if (buf->start == NULL) {
        buf->start = ngx_palloc(c->pool, c->ssl->buffer_size);
        if (buf->start == NULL) {
            return NGX_CHAIN_ERROR;
        }

        buf->pos = buf->start;
        buf->last = buf->start;
        buf->end = buf->start + c->ssl->buffer_size;
    }


```

```
    send = buf->last - buf->pos;
    flush = (in == NULL) ? 1 : buf->flush;

    for ( ;; ) {

        while (in && buf->last < buf->end && send < limit) {
            if (in->buf->last_buf || in->buf->flush) {
                flush = 1;
            }

            if (ngx_buf_special(in->buf)) {
                in = in->next;
                continue;
            }

            size = in->buf->last - in->buf->pos;

            if (size > buf->end - buf->last) {
                size = buf->end - buf->last;
            }

            if (send + size > limit) {
                size = (ssize_t) (limit - send);
            }

            ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
                           "SSL buf copy: %d", size);

            ngx_memcpy(buf->last, in->buf->pos, size);

            buf->last += size;
            in->buf->pos += size;
            send += size;

            if (in->buf->pos == in->buf->last) {
                in = in->next;
            }
        }

        if (!flush && send < limit && buf->last < buf->end) {
            break;
        }

        size = buf->last - buf->pos;

        if (size == 0) {
            buf->flush = 0;
            c->buffered &= ~NGX_SSL_BUFFERED;
            return in;
        }

        n = ngx_ssl_write(c, buf->pos, size);

        if (n == NGX_ERROR) {
            return NGX_CHAIN_ERROR;
        }

        if (n == NGX_AGAIN) {
            break;
        }

        buf->pos += n;
        c->sent += n;

        if (n < size) {
            break;
        }

        flush = 0;

        buf->pos = buf->start;
        buf->last = buf->start;

        if (in == NULL || send == limit) {
            break;
        }
```

```
1634     }
1635
1636     buf->flush = flush;
1637
1638     if (buf->pos < buf->last) {
1639         c->buffered |= NGX_SSL_BUFFERED;
1640
1641     } else {
1642         c->buffered &= ~NGX_SSL_BUFFERED;
1643     }
1644
1645     return in;
1646 }
1647
1648
1649 ssize_t
1650 ngx_ssl_write(ngx_connection_t *c, u_char *data, size_t size)
1651 {
1652     int        n, sslerr;
1653     ngx_err_t  err;
1654
1655     ngx_ssl_clear_error(c->log);
1656
1657     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL to write: %d", size);
1658
1659     n = SSL_write(c->ssl->connection, data, size);
1660
1661     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_write: %d", n);
1662
1663     if (n > 0) {
1664
1665         if (c->ssl->saved_read_handler) {
1666
1667             c->read->handler = c->ssl->saved_read_handler;
1668             c->ssl->saved_read_handler = NULL;
1669             c->read->ready = 1;
1670
1671             if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
1672                 return NGX_ERROR;
1673             }
1674
1675             ngx_post_event(c->read, &ngx_posted_events);
1676         }
1677
1678         return n;
1679     }
1680
1681     sslerr = SSL_get_error(c->ssl->connection, n);
1682
1683     err = (sslerr == SSL_ERROR_SYSCALL) ? ngx_errno : 0;
1684
1685     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_get_error: %d", sslerr);
1686
1687     if (sslerr == SSL_ERROR_WANT_WRITE) {
1688         c->write->ready = 0;
1689         return NGX_AGAIN;
1690     }
1691
1692     if (sslerr == SSL_ERROR_WANT_READ) {
1693
1694         ngx_log_error(NGX_LOG_INFO, c->log, 0,
1695                       "peer started SSL renegotiation");
1696
1697         c->read->ready = 0;
1698
1699         if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
1700             return NGX_ERROR;
1701         }
1702
1703         /*
1704          * we do not set the timer because there is already
1705          * the write event timer
1706          */
1707
1708         if (c->ssl->saved_read_handler == NULL) {
1709             c->ssl->saved_read_handler = c->read->handler;
```

```
            c->read->handler = ngx_ssl_read_handler;
        }

        return NGX_AGAIN;
    }

    c->ssl->no_wait_shutdown = 1;
    c->ssl->no_send_shutdown = 1;
    c->write->error = 1;

    ngx_ssl_connection_error(c, sslerr, err, "SSL_write() failed");

    return NGX_ERROR;
}


static void
ngx_ssl_read_handler(ngx_event_t *rev)
{
    ngx_connection_t  *c;

    c = rev->data;

    c->write->handler(c->write);
}


void
ngx_ssl_free_buffer(ngx_connection_t *c)
{
    if (c->ssl->buf && c->ssl->buf->start) {
        if (ngx_pfree(c->pool, c->ssl->buf->start) == NGX_OK) {
            c->ssl->buf->start = NULL;
        }
    }
}


ngx_int_t
ngx_ssl_shutdown(ngx_connection_t *c)
{
    int        n, sslerr, mode;
    ngx_err_t  err;

    if (c->timedout) {
        mode = SSL_RECEIVED_SHUTDOWN|SSL_SENT_SHUTDOWN;
        SSL_set_quiet_shutdown(c->ssl->connection, 1);

    } else {
        mode = SSL_get_shutdown(c->ssl->connection);

        if (c->ssl->no_wait_shutdown) {
            mode |= SSL_RECEIVED_SHUTDOWN;
        }

        if (c->ssl->no_send_shutdown) {
            mode |= SSL_SENT_SHUTDOWN;
        }

        if (c->ssl->no_wait_shutdown && c->ssl->no_send_shutdown) {
            SSL_set_quiet_shutdown(c->ssl->connection, 1);
        }
    }

    SSL_set_shutdown(c->ssl->connection, mode);

    ngx_ssl_clear_error(c->log);

    n = SSL_shutdown(c->ssl->connection);

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "SSL_shutdown: %d", n);

    sslerr = 0;

    /* SSL_shutdown() never returns -1, on error it returns 0 */

```

```
    if (n != 1 && ERR_peek_error()) {
        sslerr = SSL_get_error(c->ssl->connection, n);

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
                       "SSL_get_error: %d", sslerr);
    }

    if (n == 1 || sslerr == 0 || sslerr == SSL_ERROR_ZERO_RETURN) {
        SSL_free(c->ssl->connection);
        c->ssl = NULL;

        return NGX_OK;
    }

    if (sslerr == SSL_ERROR_WANT_READ || sslerr == SSL_ERROR_WANT_WRITE) {
        c->read->handler = ngx_ssl_shutdown_handler;
        c->write->handler = ngx_ssl_shutdown_handler;

        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            return NGX_ERROR;
        }

        if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
            return NGX_ERROR;
        }

        if (sslerr == SSL_ERROR_WANT_READ) {
            ngx_add_timer(c->read, 30000);
        }

        return NGX_AGAIN;
    }

    err = (sslerr == SSL_ERROR_SYSCALL) ? ngx_errno : 0;

    ngx_ssl_connection_error(c, sslerr, err, "SSL_shutdown() failed");

    SSL_free(c->ssl->connection);
    c->ssl = NULL;

    return NGX_ERROR;
}


static void
ngx_ssl_shutdown_handler(ngx_event_t *ev)
{
    ngx_connection_t           *c;
    ngx_connection_handler_pt   handler;

    c = ev->data;
    handler = c->ssl->handler;

    if (ev->timedout) {
        c->timedout = 1;
    }

    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ev->log, 0, "SSL shutdown handler");

    if (ngx_ssl_shutdown(c) == NGX_AGAIN) {
        return;
    }

    handler(c);
}


static void
ngx_ssl_connection_error(ngx_connection_t *c, int sslerr, ngx_err_t err,
    char *text)
{
    int         n;
    ngx_uint_t  level;

    level = NGX_LOG_CRIT;

```

```
1862      if (sslerr == SSL_ERROR_SYSCALL) {
1863
1864          if (err == NGX_ECONNRESET
1865              || err == NGX_EPIPE
1866              || err == NGX_ENOTCONN
1867              || err == NGX_ETIMEDOUT
1868              || err == NGX_ECONNREFUSED
1869              || err == NGX_ENETDOWN
1870              || err == NGX_ENETUNREACH
1871              || err == NGX_EHOSTDOWN
1872              || err == NGX_EHOSTUNREACH)
1873          {
1874              switch (c->log_error) {
1875
1876              case NGX_ERROR_IGNORE_ECONNRESET:
1877              case NGX_ERROR_INFO:
1878                  level = NGX_LOG_INFO;
1879                  break;
1880
1881              case NGX_ERROR_ERR:
1882                  level = NGX_LOG_ERR;
1883                  break;
1884
1885              default:
1886                  break;
1887              }
1888          }
1889
1890      } else if (sslerr == SSL_ERROR_SSL) {
1891
1892          n = ERR_GET_REASON(ERR_peek_error());
1893
1894              /* handshake failures */
1895          if (n == SSL_R_BAD_CHANGE_CIPHER_SPEC                       /*  103 */
1896              || n == SSL_R_BLOCK_CIPHER_PAD_IS_WRONG                 /*  129 */
1897              || n == SSL_R_DIGEST_CHECK_FAILED                       /*  149 */
1898              || n == SSL_R_ERROR_IN_RECEIVED_CIPHER_LIST             /*  151 */
1899              || n == SSL_R_EXCESSIVE_MESSAGE_SIZE                    /*  152 */
1900              || n == SSL_R_LENGTH_MISMATCH                           /*  159 */
1901              || n == SSL_R_NO_CIPHERS_PASSED                         /*  182 */
1902              || n == SSL_R_NO_CIPHERS_SPECIFIED                      /*  183 */
1903              || n == SSL_R_NO_COMPRESSION_SPECIFIED                  /*  187 */
1904              || n == SSL_R_NO_SHARED_CIPHER                          /*  193 */
1905              || n == SSL_R_RECORD_LENGTH_MISMATCH                    /*  213 */
1906  #ifdef SSL_R_PARSE_TLSEXT
1907              || n == SSL_R_PARSE_TLSEXT                              /*  227 */
1908  #endif
1909              || n == SSL_R_UNEXPECTED_MESSAGE                        /*  244 */
1910              || n == SSL_R_UNEXPECTED_RECORD                         /*  245 */
1911              || n == SSL_R_UNKNOWN_ALERT_TYPE                        /*  246 */
1912              || n == SSL_R_UNKNOWN_PROTOCOL                          /*  252 */
1913              || n == SSL_R_WRONG_VERSION_NUMBER                      /*  267 */
1914              || n == SSL_R_DECRYPTION_FAILED_OR_BAD_RECORD_MAC       /*  281 */
1915  #ifdef SSL_R_RENEGOTIATE_EXT_TOO_LONG
1916              || n == SSL_R_RENEGOTIATE_EXT_TOO_LONG                  /*  335 */
1917              || n == SSL_R_RENEGOTIATION_ENCODING_ERR                /*  336 */
1918              || n == SSL_R_RENEGOTIATION_MISMATCH                    /*  337 */
1919  #endif
1920  #ifdef SSL_R_UNSAFE_LEGACY_RENEGOTIATION_DISABLED
1921              || n == SSL_R_UNSAFE_LEGACY_RENEGOTIATION_DISABLED      /*  338 */
1922  #endif
1923  #ifdef SSL_R_SCSV_RECEIVED_WHEN_RENEGOTIATING
1924              || n == SSL_R_SCSV_RECEIVED_WHEN_RENEGOTIATING          /*  345 */
1925  #endif
1926  #ifdef SSL_R_INAPPROPRIATE_FALLBACK
1927              || n == SSL_R_INAPPROPRIATE_FALLBACK                    /*  373 */
1928  #endif
1929              || n == 1000 /* SSL_R_SSLV3_ALERT_CLOSE_NOTIFY */
1930              || n == SSL_R_SSLV3_ALERT_UNEXPECTED_MESSAGE            /* 1010 */
1931              || n == SSL_R_SSLV3_ALERT_BAD_RECORD_MAC                /* 1020 */
1932              || n == SSL_R_TLSV1_ALERT_DECRYPTION_FAILED             /* 1021 */
1933              || n == SSL_R_TLSV1_ALERT_RECORD_OVERFLOW               /* 1022 */
1934              || n == SSL_R_SSLV3_ALERT_DECOMPRESSION_FAILURE         /* 1030 */
1935              || n == SSL_R_SSLV3_ALERT_HANDSHAKE_FAILURE             /* 1040 */
1936              || n == SSL_R_SSLV3_ALERT_NO_CERTIFICATE                /* 1041 */
1937              || n == SSL_R_SSLV3_ALERT_BAD_CERTIFICATE               /* 1042 */
```

```
            || n == SSL_R_SSLV3_ALERT_UNSUPPORTED_CERTIFICATE        /* 1043 */
            || n == SSL_R_SSLV3_ALERT_CERTIFICATE_REVOKED            /* 1044 */
            || n == SSL_R_SSLV3_ALERT_CERTIFICATE_EXPIRED            /* 1045 */
            || n == SSL_R_SSLV3_ALERT_CERTIFICATE_UNKNOWN            /* 1046 */
            || n == SSL_R_SSLV3_ALERT_ILLEGAL_PARAMETER              /* 1047 */
            || n == SSL_R_TLSV1_ALERT_UNKNOWN_CA                     /* 1048 */
            || n == SSL_R_TLSV1_ALERT_ACCESS_DENIED                  /* 1049 */
            || n == SSL_R_TLSV1_ALERT_DECODE_ERROR                   /* 1050 */
            || n == SSL_R_TLSV1_ALERT_DECRYPT_ERROR                  /* 1051 */
            || n == SSL_R_TLSV1_ALERT_EXPORT_RESTRICTION             /* 1060 */
            || n == SSL_R_TLSV1_ALERT_PROTOCOL_VERSION               /* 1070 */
            || n == SSL_R_TLSV1_ALERT_INSUFFICIENT_SECURITY          /* 1071 */
            || n == SSL_R_TLSV1_ALERT_INTERNAL_ERROR                 /* 1080 */
            || n == SSL_R_TLSV1_ALERT_USER_CANCELLED                 /* 1090 */
            || n == SSL_R_TLSV1_ALERT_NO_RENEGOTIATION)              /* 1100 */
        {
            switch (c->log_error) {

            case NGX_ERROR_IGNORE_ECONNRESET:
            case NGX_ERROR_INFO:
                level = NGX_LOG_INFO;
                break;

            case NGX_ERROR_ERR:
                level = NGX_LOG_ERR;
                break;

            default:
                break;
            }
        }
    }

    ngx_ssl_error(level, c->log, err, text);
}


static void
ngx_ssl_clear_error(ngx_log_t *log)
{
    while (ERR_peek_error()) {
        ngx_ssl_error(NGX_LOG_ALERT, log, 0, "ignoring stale global SSL error");
    }

    ERR_clear_error();
}


void ngx_cdecl
ngx_ssl_error(ngx_uint_t level, ngx_log_t *log, ngx_err_t err, char *fmt, ...)
{
    int          flags;
    u_long       n;
    va_list      args;
    u_char      *p, *last;
    u_char       errstr[NGX_MAX_CONF_ERRSTR];
    const char  *data;

    last = errstr + NGX_MAX_CONF_ERRSTR;

    va_start(args, fmt);
    p = ngx_vslprintf(errstr, last - 1, fmt, args);
    va_end(args);

    p = ngx_cpystrn(p, (u_char *) " (SSL:", last - p);

    for ( ;; ) {

        n = ERR_peek_error_line_data(NULL, NULL, &data, &flags);

        if (n == 0) {
            break;
        }

        if (p >= last) {
            goto next;
```

```
2014        }
2015
2016        *p++ = ' ';
2017
2018        ERR_error_string_n(n, (char *) p, last - p);
2019
2020        while (p < last && *p) {
2021            p++;
2022        }
2023
2024        if (p < last && *data && (flags & ERR_TXT_STRING)) {
2025            *p++ = ':';
2026            p = ngx_cpystrn(p, (u_char *) data, last - p);
2027        }
2028
2029    next:
2030
2031        (void) ERR_get_error();
2032    }
2033
2034    ngx_log_error(level, log, err, "%s)", errstr);
2035 }
2036
2037
2038 ngx_int_t
2039 ngx_ssl_session_cache(ngx_ssl_t *ssl, ngx_str_t *sess_ctx,
2040     ssize_t builtin_session_cache, ngx_shm_zone_t *shm_zone, time_t timeout)
2041 {
2042     long  cache_mode;
2043
2044     SSL_CTX_set_timeout(ssl->ctx, (long) timeout);
2045
2046     if (ngx_ssl_session_id_context(ssl, sess_ctx) != NGX_OK) {
2047         return NGX_ERROR;
2048     }
2049
2050     if (builtin_session_cache == NGX_SSL_NO_SCACHE) {
2051         SSL_CTX_set_session_cache_mode(ssl->ctx, SSL_SESS_CACHE_OFF);
2052         return NGX_OK;
2053     }
2054
2055     if (builtin_session_cache == NGX_SSL_NONE_SCACHE) {
2056
2057         /*
2058          * If the server explicitly says that it does not support
2059          * session reuse (see SSL_SESS_CACHE_OFF above), then
2060          * Outlook Express fails to upload a sent email to
2061          * the Sent Items folder on the IMAP server via a separate IMAP
2062          * connection in the background. Therefore we have a special
2063          * mode (SSL_SESS_CACHE_SERVER|SSL_SESS_CACHE_NO_INTERNAL_STORE)
2064          * where the server pretends that it supports session reuse,
2065          * but it does not actually store any session.
2066          */
2067
2068         SSL_CTX_set_session_cache_mode(ssl->ctx,
2069                                        SSL_SESS_CACHE_SERVER
2070                                        |SSL_SESS_CACHE_NO_AUTO_CLEAR
2071                                        |SSL_SESS_CACHE_NO_INTERNAL_STORE);
2072
2073         SSL_CTX_sess_set_cache_size(ssl->ctx, 1);
2074
2075         return NGX_OK;
2076     }
2077
2078     cache_mode = SSL_SESS_CACHE_SERVER;
2079
2080     if (shm_zone && builtin_session_cache == NGX_SSL_NO_BUILTIN_SCACHE) {
2081         cache_mode |= SSL_SESS_CACHE_NO_INTERNAL;
2082     }
2083
2084     SSL_CTX_set_session_cache_mode(ssl->ctx, cache_mode);
2085
2086     if (builtin_session_cache != NGX_SSL_NO_BUILTIN_SCACHE) {
2087
2088         if (builtin_session_cache != NGX_SSL_DFLT_BUILTIN_SCACHE) {
2089             SSL_CTX_sess_set_cache_size(ssl->ctx, builtin_session_cache);
```

```
2090        }
2091    }
2092
2093    if (shm_zone) {
2094        SSL_CTX_sess_set_new_cb(ssl->ctx, ngx_ssl_new_session);
2095        SSL_CTX_sess_set_get_cb(ssl->ctx, ngx_ssl_get_cached_session);
2096        SSL_CTX_sess_set_remove_cb(ssl->ctx, ngx_ssl_remove_session);
2097
2098        if (SSL_CTX_set_ex_data(ssl->ctx, ngx_ssl_session_cache_index, shm_zone)
2099            == 0)
2100        {
2101            ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2102                          "SSL_CTX_set_ex_data() failed");
2103            return NGX_ERROR;
2104        }
2105    }
2106
2107    return NGX_OK;
2108 }
2109
2110
2111 static ngx_int_t
2112 ngx_ssl_session_id_context(ngx_ssl_t *ssl, ngx_str_t *sess_ctx)
2113 {
2114    int                  n, i;
2115    X509                *cert;
2116    X509_NAME           *name;
2117    EVP_MD_CTX           md;
2118    unsigned int         len;
2119    STACK_OF(X509_NAME) *list;
2120    u_char               buf[EVP_MAX_MD_SIZE];
2121
2122    /*
2123     * Session ID context is set based on the string provided,
2124     * the server certificate, and the client CA list.
2125     */
2126
2127    EVP_MD_CTX_init(&md);
2128
2129    if (EVP_DigestInit_ex(&md, EVP_sha1(), NULL) == 0) {
2130        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2131                      "EVP_DigestInit_ex() failed");
2132        goto failed;
2133    }
2134
2135    if (EVP_DigestUpdate(&md, sess_ctx->data, sess_ctx->len) == 0) {
2136        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2137                      "EVP_DigestUpdate() failed");
2138        goto failed;
2139    }
2140
2141    cert = SSL_CTX_get_ex_data(ssl->ctx, ngx_ssl_certificate_index);
2142
2143    if (X509_digest(cert, EVP_sha1(), buf, &len) == 0) {
2144        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2145                      "X509_digest() failed");
2146        goto failed;
2147    }
2148
2149    if (EVP_DigestUpdate(&md, buf, len) == 0) {
2150        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2151                      "EVP_DigestUpdate() failed");
2152        goto failed;
2153    }
2154
2155    list = SSL_CTX_get_client_CA_list(ssl->ctx);
2156
2157    if (list != NULL) {
2158        n = sk_X509_NAME_num(list);
2159
2160        for (i = 0; i < n; i++) {
2161            name = sk_X509_NAME_value(list, i);
2162
2163            if (X509_NAME_digest(name, EVP_sha1(), buf, &len) == 0) {
2164                ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2165                              "X509_NAME_digest() failed");
```

```
2166                goto failed;
2167            }
2168
2169            if (EVP_DigestUpdate(&md, buf, len) == 0) {
2170                ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2171                              "EVP_DigestUpdate() failed");
2172                goto failed;
2173            }
2174        }
2175    }
2176
2177    if (EVP_DigestFinal_ex(&md, buf, &len) == 0) {
2178        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2179                      "EVP_DigestUpdate() failed");
2180        goto failed;
2181    }
2182
2183    EVP_MD_CTX_cleanup(&md);
2184
2185    if (SSL_CTX_set_session_id_context(ssl->ctx, buf, len) == 0) {
2186        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
2187                      "SSL_CTX_set_session_id_context() failed");
2188        return NGX_ERROR;
2189    }
2190
2191    return NGX_OK;
2192
2193 failed:
2194
2195    EVP_MD_CTX_cleanup(&md);
2196
2197    return NGX_ERROR;
2198 }
2199
2200
2201 ngx_int_t
2202 ngx_ssl_session_cache_init(ngx_shm_zone_t *shm_zone, void *data)
2203 {
2204    size_t                  len;
2205    ngx_slab_pool_t        *shpool;
2206    ngx_ssl_session_cache_t  *cache;
2207
2208    if (data) {
2209        shm_zone->data = data;
2210        return NGX_OK;
2211    }
2212
2213    shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;
2214
2215    if (shm_zone->shm.exists) {
2216        shm_zone->data = shpool->data;
2217        return NGX_OK;
2218    }
2219
2220    cache = ngx_slab_alloc(shpool, sizeof(ngx_ssl_session_cache_t));
2221    if (cache == NULL) {
2222        return NGX_ERROR;
2223    }
2224
2225    shpool->data = cache;
2226    shm_zone->data = cache;
2227
2228    ngx_rbtree_init(&cache->session_rbtree, &cache->sentinel,
2229                    ngx_ssl_session_rbtree_insert_value);
2230
2231    ngx_queue_init(&cache->expire_queue);
2232
2233    len = sizeof(" in SSL session shared cache \"\"") + shm_zone->shm.name.len;
2234
2235    shpool->log_ctx = ngx_slab_alloc(shpool, len);
2236    if (shpool->log_ctx == NULL) {
2237        return NGX_ERROR;
2238    }
2239
2240    ngx_sprintf(shpool->log_ctx, " in SSL session shared cache \"%V\"%Z",
2241                &shm_zone->shm.name);
```

```
2242
2243        shpool->log_nomem = 0;
2244
2245        return NGX_OK;
2246    }
2247
2248
2249    /*
2250    * The length of the session id is 16 bytes for SSLv2 sessions and
2251    * between 1 and 32 bytes for SSLv3/TLSv1, typically 32 bytes.
2252    * It seems that the typical length of the external ASN1 representation
2253    * of a session is 118 or 119 bytes for SSLv3/TSLv1.
2254    *
2255    * Thus on 32-bit platforms we allocate separately an rbtree node,
2256    * a session id, and an ASN1 representation, they take accordingly
2257    * 64, 32, and 128 bytes.
2258    *
2259    * On 64-bit platforms we allocate separately an rbtree node + session_id,
2260    * and an ASN1 representation, they take accordingly 128 and 128 bytes.
2261    *
2262    * OpenSSL's i2d_SSL_SESSION() and d2i_SSL_SESSION are slow,
2263    * so they are outside the code locked by shared pool mutex
2264    */
2265
2266    static int
2267    ngx_ssl_new_session(ngx_ssl_conn_t *ssl_conn, ngx_ssl_session_t *sess)
2268    {
2269        int                      len;
2270        u_char                  *p, *id, *cached_sess, *session_id;
2271        uint32_t                 hash;
2272        SSL_CTX                 *ssl_ctx;
2273        unsigned int             session_id_length;
2274        ngx_shm_zone_t          *shm_zone;
2275        ngx_connection_t        *c;
2276        ngx_slab_pool_t         *shpool;
2277        ngx_ssl_sess_id_t       *sess_id;
2278        ngx_ssl_session_cache_t  *cache;
2279        u_char                   buf[NGX_SSL_MAX_SESSION_SIZE];
2280
2281        len = i2d_SSL_SESSION(sess, NULL);
2282
2283        /* do not cache too big session */
2284
2285        if (len > (int) NGX_SSL_MAX_SESSION_SIZE) {
2286            return 0;
2287        }
2288
2289        p = buf;
2290        i2d_SSL_SESSION(sess, &p);
2291
2292        c = ngx_ssl_get_connection(ssl_conn);
2293
2294        ssl_ctx = SSL_get_SSL_CTX(ssl_conn);
2295        shm_zone = SSL_CTX_get_ex_data(ssl_ctx, ngx_ssl_session_cache_index);
2296
2297        cache = shm_zone->data;
2298        shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;
2299
2300        ngx_shmtx_lock(&shpool->mutex);
2301
2302        /* drop one or two expired sessions */
2303        ngx_ssl_expire_sessions(cache, shpool, 1);
2304
2305        cached_sess = ngx_slab_alloc_locked(shpool, len);
2306
2307        if (cached_sess == NULL) {
2308
2309            /* drop the oldest non-expired session and try once more */
2310
2311            ngx_ssl_expire_sessions(cache, shpool, 0);
2312
2313            cached_sess = ngx_slab_alloc_locked(shpool, len);
2314
2315            if (cached_sess == NULL) {
2316                sess_id = NULL;
2317                goto failed;
```

```
        }
    }

    sess_id = ngx_slab_alloc_locked(shpool, sizeof(ngx_ssl_sess_id_t));

    if (sess_id == NULL) {

        /* drop the oldest non-expired session and try once more */

        ngx_ssl_expire_sessions(cache, shpool, 0);

        sess_id = ngx_slab_alloc_locked(shpool, sizeof(ngx_ssl_sess_id_t));

        if (sess_id == NULL) {
            goto failed;
        }
    }

#if OPENSSL_VERSION_NUMBER >= 0x0090800fL

    session_id = (u_char *) SSL_SESSION_get_id(sess, &session_id_length);

#else

    session_id = sess->session_id;
    session_id_length = sess->session_id_length;

#endif

#if (NGX_PTR_SIZE == 8)

    id = sess_id->sess_id;

#else

    id = ngx_slab_alloc_locked(shpool, session_id_length);

    if (id == NULL) {

        /* drop the oldest non-expired session and try once more */

        ngx_ssl_expire_sessions(cache, shpool, 0);

        id = ngx_slab_alloc_locked(shpool, session_id_length);

        if (id == NULL) {
            goto failed;
        }
    }

#endif

    ngx_memcpy(cached_sess, buf, len);

    ngx_memcpy(id, session_id, session_id_length);

    hash = ngx_crc32_short(session_id, session_id_length);

    ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "ssl new session: %08XD:%ud:%d",
                   hash, session_id_length, len);

    sess_id->node.key = hash;
    sess_id->node.data = (u_char) session_id_length;
    sess_id->id = id;
    sess_id->len = len;
    sess_id->session = cached_sess;

    sess_id->expire = ngx_time() + SSL_CTX_get_timeout(ssl_ctx);

    ngx_queue_insert_head(&cache->expire_queue, &sess_id->queue);

    ngx_rbtree_insert(&cache->session_rbtree, &sess_id->node);

    ngx_shmtx_unlock(&shpool->mutex);

```

```
2394     return 0;
2395
2396 failed:
2397
2398     if (cached_sess) {
2399         ngx_slab_free_locked(shpool, cached_sess);
2400     }
2401
2402     if (sess_id) {
2403         ngx_slab_free_locked(shpool, sess_id);
2404     }
2405
2406     ngx_shmtx_unlock(&shpool->mutex);
2407
2408     ngx_log_error(NGX_LOG_ALERT, c->log, 0,
2409                   "could not allocate new session%s", shpool->log_ctx);
2410
2411     return 0;
2412 }
2413
2414
2415 static ngx_ssl_session_t *
2416 ngx_ssl_get_cached_session(ngx_ssl_conn_t *ssl_conn, u_char *id, int len,
2417     int *copy)
2418 {
2419 #if OPENSSL_VERSION_NUMBER >= 0x0090707fL
2420     const
2421 #endif
2422     u_char                  *p;
2423     uint32_t                 hash;
2424     ngx_int_t                rc;
2425     ngx_shm_zone_t          *shm_zone;
2426     ngx_slab_pool_t         *shpool;
2427     ngx_rbtree_node_t       *node, *sentinel;
2428     ngx_ssl_session_t       *sess;
2429     ngx_ssl_sess_id_t       *sess_id;
2430     ngx_ssl_session_cache_t  *cache;
2431     u_char                   buf[NGX_SSL_MAX_SESSION_SIZE];
2432 #if (NGX_DEBUG)
2433     ngx_connection_t        *c;
2434 #endif
2435
2436     hash = ngx_crc32_short(id, (size_t) len);
2437     *copy = 0;
2438
2439 #if (NGX_DEBUG)
2440     c = ngx_ssl_get_connection(ssl_conn);
2441
2442     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
2443                    "ssl get session: %08XD:%d", hash, len);
2444 #endif
2445
2446     shm_zone = SSL_CTX_get_ex_data(SSL_get_SSL_CTX(ssl_conn),
2447                                    ngx_ssl_session_cache_index);
2448
2449     cache = shm_zone->data;
2450
2451     sess = NULL;
2452
2453     shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;
2454
2455     ngx_shmtx_lock(&shpool->mutex);
2456
2457     node = cache->session_rbtree.root;
2458     sentinel = cache->session_rbtree.sentinel;
2459
2460     while (node != sentinel) {
2461
2462         if (hash < node->key) {
2463             node = node->left;
2464             continue;
2465         }
2466
2467         if (hash > node->key) {
2468             node = node->right;
2469             continue;
```

```
2470            }
2471
2472            /* hash == node->key */
2473
2474            sess_id = (ngx_ssl_sess_id_t *) node;
2475
2476            rc = ngx_memn2cmp(id, sess_id->id, (size_t) len, (size_t) node->data);
2477
2478            if (rc == 0) {
2479
2480                if (sess_id->expire > ngx_time()) {
2481                    ngx_memcpy(buf, sess_id->session, sess_id->len);
2482
2483                    ngx_shmtx_unlock(&shpool->mutex);
2484
2485                    p = buf;
2486                    sess = d2i_SSL_SESSION(NULL, &p, sess_id->len);
2487
2488                    return sess;
2489                }
2490
2491                ngx_queue_remove(&sess_id->queue);
2492
2493                ngx_rbtree_delete(&cache->session_rbtree, node);
2494
2495                ngx_slab_free_locked(shpool, sess_id->session);
2496    #if (NGX_PTR_SIZE == 4)
2497                ngx_slab_free_locked(shpool, sess_id->id);
2498    #endif
2499                ngx_slab_free_locked(shpool, sess_id);
2500
2501                sess = NULL;
2502
2503                goto done;
2504            }
2505
2506            node = (rc < 0) ? node->left : node->right;
2507        }
2508
2509    done:
2510
2511        ngx_shmtx_unlock(&shpool->mutex);
2512
2513        return sess;
2514    }
2515
2516
2517    void
2518    ngx_ssl_remove_cached_session(SSL_CTX *ssl, ngx_ssl_session_t *sess)
2519    {
2520         SSL_CTX_remove_session(ssl, sess);
2521
2522         ngx_ssl_remove_session(ssl, sess);
2523    }
2524
2525
2526    static void
2527    ngx_ssl_remove_session(SSL_CTX *ssl, ngx_ssl_session_t *sess)
2528    {
2529        u_char                  *id;
2530        uint32_t                 hash;
2531        ngx_int_t                rc;
2532        unsigned int             len;
2533        ngx_shm_zone_t          *shm_zone;
2534        ngx_slab_pool_t         *shpool;
2535        ngx_rbtree_node_t       *node, *sentinel;
2536        ngx_ssl_sess_id_t       *sess_id;
2537        ngx_ssl_session_cache_t  *cache;
2538
2539        shm_zone = SSL_CTX_get_ex_data(ssl, ngx_ssl_session_cache_index);
2540
2541        if (shm_zone == NULL) {
2542            return;
2543        }
2544
2545        cache = shm_zone->data;
```

```
2546
2547 #if OPENSSL_VERSION_NUMBER >= 0x0090800fL
2548
2549     id = (u_char *) SSL_SESSION_get_id(sess, &len);
2550
2551 #else
2552
2553     id = sess->session_id;
2554     len = sess->session_id_length;
2555
2556 #endif
2557
2558     hash = ngx_crc32_short(id, len);
2559
2560     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ngx_cycle->log, 0,
2561                    "ssl remove session: %08XD:%ud", hash, len);
2562
2563     shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;
2564
2565     ngx_shmtx_lock(&shpool->mutex);
2566
2567     node = cache->session_rbtree.root;
2568     sentinel = cache->session_rbtree.sentinel;
2569
2570     while (node != sentinel) {
2571
2572         if (hash < node->key) {
2573             node = node->left;
2574             continue;
2575         }
2576
2577         if (hash > node->key) {
2578             node = node->right;
2579             continue;
2580         }
2581
2582         /* hash == node->key */
2583
2584         sess_id = (ngx_ssl_sess_id_t *) node;
2585
2586         rc = ngx_memn2cmp(id, sess_id->id, len, (size_t) node->data);
2587
2588         if (rc == 0) {
2589
2590             ngx_queue_remove(&sess_id->queue);
2591
2592             ngx_rbtree_delete(&cache->session_rbtree, node);
2593
2594             ngx_slab_free_locked(shpool, sess_id->session);
2595 #if (NGX_PTR_SIZE == 4)
2596             ngx_slab_free_locked(shpool, sess_id->id);
2597 #endif
2598             ngx_slab_free_locked(shpool, sess_id);
2599
2600             goto done;
2601         }
2602
2603         node = (rc < 0) ? node->left : node->right;
2604     }
2605
2606 done:
2607
2608     ngx_shmtx_unlock(&shpool->mutex);
2609 }
2610
2611
2612 static void
2613 ngx_ssl_expire_sessions(ngx_ssl_session_cache_t *cache,
2614     ngx_slab_pool_t *shpool, ngx_uint_t n)
2615 {
2616     time_t              now;
2617     ngx_queue_t        *q;
2618     ngx_ssl_sess_id_t  *sess_id;
2619
2620     now = ngx_time();
2621
```

```
    while (n < 3) {

        if (ngx_queue_empty(&cache->expire_queue)) {
            return;
        }

        q = ngx_queue_last(&cache->expire_queue);

        sess_id = ngx_queue_data(q, ngx_ssl_sess_id_t, queue);

        if (n++ != 0 && sess_id->expire > now) {
            return;
        }

        ngx_queue_remove(q);

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ngx_cycle->log, 0,
                       "expire session: %08Xi", sess_id->node.key);

        ngx_rbtree_delete(&cache->session_rbtree, &sess_id->node);

        ngx_slab_free_locked(shpool, sess_id->session);
#if (NGX_PTR_SIZE == 4)
        ngx_slab_free_locked(shpool, sess_id->id);
#endif
        ngx_slab_free_locked(shpool, sess_id);
    }
}


static void
ngx_ssl_session_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
    ngx_rbtree_node_t  **p;
    ngx_ssl_sess_id_t   *sess_id, *sess_id_temp;

    for ( ;; ) {

        if (node->key < temp->key) {

            p = &temp->left;

        } else if (node->key > temp->key) {

            p = &temp->right;

        } else { /* node->key == temp->key */

            sess_id = (ngx_ssl_sess_id_t *) node;
            sess_id_temp = (ngx_ssl_sess_id_t *) temp;

            p = (ngx_memn2cmp(sess_id->id, sess_id_temp->id,
                              (size_t) node->data, (size_t) temp->data)
                 < 0) ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
    ngx_rbt_red(node);
}


#ifdef SSL_CTRL_SET_TLSEXT_TICKET_KEY_CB

ngx_int_t
ngx_ssl_session_ticket_keys(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_array_t *paths)
```

```
{
    u_char                          buf[48];
    ssize_t                         n;
    ngx_str_t                      *path;
    ngx_file_t                      file;
    ngx_uint_t                      i;
    ngx_array_t                    *keys;
    ngx_file_info_t                 fi;
    ngx_ssl_session_ticket_key_t   *key;

    if (paths == NULL) {
        return NGX_OK;
    }

    keys = ngx_array_create(cf->pool, paths->nelts,
                            sizeof(ngx_ssl_session_ticket_key_t));
    if (keys == NULL) {
        return NGX_ERROR;
    }

    path = paths->elts;
    for (i = 0; i < paths->nelts; i++) {

        if (ngx_conf_full_name(cf->cycle, &path[i], 1) != NGX_OK) {
            return NGX_ERROR;
        }

        ngx_memzero(&file, sizeof(ngx_file_t));
        file.name = path[i];
        file.log = cf->log;

        file.fd = ngx_open_file(file.name.data, NGX_FILE_RDONLY, 0, 0);
        if (file.fd == NGX_INVALID_FILE) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, ngx_errno,
                               ngx_open_file_n " \"%V\" failed", &file.name);
            return NGX_ERROR;
        }

        if (ngx_fd_info(file.fd, &fi) == NGX_FILE_ERROR) {
            ngx_conf_log_error(NGX_LOG_CRIT, cf, ngx_errno,
                               ngx_fd_info_n " \"%V\" failed", &file.name);
            goto failed;
        }

        if (ngx_file_size(&fi) != 48) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "\"%V\" must be 48 bytes", &file.name);
            goto failed;
        }

        n = ngx_read_file(&file, buf, 48, 0);

        if (n == NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_CRIT, cf, ngx_errno,
                               ngx_read_file_n " \"%V\" failed", &file.name);
            goto failed;
        }

        if (n != 48) {
            ngx_conf_log_error(NGX_LOG_CRIT, cf, 0,
                               ngx_read_file_n " \"%V\" returned only "
                               "%z bytes instead of 48", &file.name, n);
            goto failed;
        }

        key = ngx_array_push(keys);
        if (key == NULL) {
            goto failed;
        }

        ngx_memcpy(key->name, buf, 16);
        ngx_memcpy(key->aes_key, buf + 16, 16);
        ngx_memcpy(key->hmac_key, buf + 32, 16);

        if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
```

```
                        ngx_close_file_n " \"%V\" failed", &file.name);
        }
    }

    if (SSL_CTX_set_ex_data(ssl->ctx, ngx_ssl_session_ticket_keys_index, keys)
        == 0)
    {
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "SSL_CTX_set_ex_data() failed");
        return NGX_ERROR;
    }

    if (SSL_CTX_set_tlsext_ticket_key_cb(ssl->ctx,
                                         ngx_ssl_session_ticket_key_callback)
        == 0)
    {
        ngx_log_error(NGX_LOG_WARN, cf->log, 0,
                      "nginx was built with Session Tickets support, however, "
                      "now it is linked dynamically to an OpenSSL library "
                      "which has no tlsext support, therefore Session Tickets "
                      "are not available");
    }

    return NGX_OK;

failed:

    if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                      ngx_close_file_n " \"%V\" failed", &file.name);
    }

    return NGX_ERROR;
}


#ifdef OPENSSL_NO_SHA256
#define ngx_ssl_session_ticket_md  EVP_sha1
#else
#define ngx_ssl_session_ticket_md  EVP_sha256
#endif


static int
ngx_ssl_session_ticket_key_callback(ngx_ssl_conn_t *ssl_conn,
    unsigned char *name, unsigned char *iv, EVP_CIPHER_CTX *ectx,
    HMAC_CTX *hctx, int enc)
{
    SSL_CTX                       *ssl_ctx;
    ngx_uint_t                     i;
    ngx_array_t                   *keys;
    ngx_ssl_session_ticket_key_t  *key;
#if (NGX_DEBUG)
    u_char                         buf[32];
    ngx_connection_t              *c;
#endif

    ssl_ctx = SSL_get_SSL_CTX(ssl_conn);

    keys = SSL_CTX_get_ex_data(ssl_ctx, ngx_ssl_session_ticket_keys_index);
    if (keys == NULL) {
        return -1;
    }

    key = keys->elts;

#if (NGX_DEBUG)
    c = ngx_ssl_get_connection(ssl_conn);
#endif

    if (enc == 1) {
        /* encrypt session ticket */

        ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
                       "ssl session ticket encrypt, key: \"%*s\" (%s session)",
                       ngx_hex_dump(buf, key[0].name, 16) - buf, buf,
```

```
2850                        SSL_session_reused(ssl_conn) ? "reused" : "new");
2851
2852        RAND_pseudo_bytes(iv, 16);
2853        EVP_EncryptInit_ex(ectx, EVP_aes_128_cbc(), NULL, key[0].aes_key, iv);
2854        HMAC_Init_ex(hctx, key[0].hmac_key, 16,
2855                     ngx_ssl_session_ticket_md(), NULL);
2856        ngx_memcpy(name, key[0].name, 16);
2857
2858        return 0;
2859
2860    } else {
2861        /* decrypt session ticket */
2862
2863        for (i = 0; i < keys->nelts; i++) {
2864            if (ngx_memcmp(name, key[i].name, 16) == 0) {
2865                goto found;
2866            }
2867        }
2868
2869        ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
2870                       "ssl session ticket decrypt, key: \"%*s\" not found",
2871                       ngx_hex_dump(buf, name, 16) - buf, buf);
2872
2873        return 0;
2874
2875    found:
2876
2877        ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
2878                       "ssl session ticket decrypt, key: \"%*s\"%s",
2879                       ngx_hex_dump(buf, key[i].name, 16) - buf, buf,
2880                       (i == 0) ? " (default)" : "");
2881
2882        HMAC_Init_ex(hctx, key[i].hmac_key, 16,
2883                     ngx_ssl_session_ticket_md(), NULL);
2884        EVP_DecryptInit_ex(ectx, EVP_aes_128_cbc(), NULL, key[i].aes_key, iv);
2885
2886        return (i == 0) ? 1 : 2 /* renew */;
2887    }
2888 }
2889
2890 #else
2891
2892 ngx_int_t
2893 ngx_ssl_session_ticket_keys(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_array_t *paths)
2894 {
2895     if (paths) {
2896         ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
2897                       "\"ssl_session_ticket_keys\" ignored, not supported");
2898     }
2899
2900     return NGX_OK;
2901 }
2902
2903 #endif
2904
2905
2906 void
2907 ngx_ssl_cleanup_ctx(void *data)
2908 {
2909     ngx_ssl_t  *ssl = data;
2910
2911     SSL_CTX_free(ssl->ctx);
2912 }
2913
2914
2915 ngx_int_t
2916 ngx_ssl_check_host(ngx_connection_t *c, ngx_str_t *name)
2917 {
2918     X509   *cert;
2919
2920     cert = SSL_get_peer_certificate(c->ssl->connection);
2921     if (cert == NULL) {
2922         return NGX_ERROR;
2923     }
2924
2925 #if (OPENSSL_VERSION_NUMBER >= 0x10002002L && !defined LIBRESSL_VERSION_NUMBER)
```

```
2926
2927        /* X509_check_host() is only available in OpenSSL 1.0.2+ */
2928
2929        if (name->len == 0) {
2930            goto failed;
2931        }
2932
2933        if (X509_check_host(cert, (char *) name->data, name->len, 0, NULL) != 1) {
2934            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
2935                           "X509_check_host(): no match");
2936            goto failed;
2937        }
2938
2939        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
2940                       "X509_check_host(): match");
2941
2942        goto found;
2943
2944    #else
2945        {
2946        int                      n, i;
2947        X509_NAME               *sname;
2948        ASN1_STRING             *str;
2949        X509_NAME_ENTRY         *entry;
2950        GENERAL_NAME            *altname;
2951        STACK_OF(GENERAL_NAME)  *altnames;
2952
2953        /*
2954         * As per RFC6125 and RFC2818, we check subjectAltName extension,
2955         * and if it's not present - commonName in Subject is checked.
2956         */
2957
2958        altnames = X509_get_ext_d2i(cert, NID_subject_alt_name, NULL, NULL);
2959
2960        if (altnames) {
2961            n = sk_GENERAL_NAME_num(altnames);
2962
2963            for (i = 0; i < n; i++) {
2964                altname = sk_GENERAL_NAME_value(altnames, i);
2965
2966                if (altname->type != GEN_DNS) {
2967                    continue;
2968                }
2969
2970                str = altname->d.dNSName;
2971
2972                ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
2973                               "SSL subjectAltName: \"%*s\"",
2974                               ASN1_STRING_length(str), ASN1_STRING_data(str));
2975
2976                if (ngx_ssl_check_name(name, str) == NGX_OK) {
2977                    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
2978                                   "SSL subjectAltName: match");
2979                    GENERAL_NAMES_free(altnames);
2980                    goto found;
2981                }
2982            }
2983
2984            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
2985                           "SSL subjectAltName: no match");
2986
2987            GENERAL_NAMES_free(altnames);
2988            goto failed;
2989        }
2990
2991        /*
2992         * If there is no subjectAltName extension, check commonName
2993         * in Subject.  While RFC2818 requires to only check "most specific"
2994         * CN, both Apache and OpenSSL check all CNs, and so do we.
2995         */
2996
2997        sname = X509_get_subject_name(cert);
2998
2999        if (sname == NULL) {
3000            goto failed;
3001        }
```

```
3002
3003        i = -1;
3004        for ( ;; ) {
3005            i = X509_NAME_get_index_by_NID(sname, NID_commonName, i);
3006
3007            if (i < 0) {
3008                break;
3009            }
3010
3011            entry = X509_NAME_get_entry(sname, i);
3012            str = X509_NAME_ENTRY_get_data(entry);
3013
3014            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
3015                           "SSL commonName: \"%*s\"",
3016                           ASN1_STRING_length(str), ASN1_STRING_data(str));
3017
3018            if (ngx_ssl_check_name(name, str) == NGX_OK) {
3019                ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
3020                               "SSL commonName: match");
3021                goto found;
3022            }
3023        }
3024
3025        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
3026                       "SSL commonName: no match");
3027        }
3028    #endif
3029
3030    failed:
3031
3032        X509_free(cert);
3033        return NGX_ERROR;
3034
3035    found:
3036
3037        X509_free(cert);
3038        return NGX_OK;
3039    }
3040
3041
3042    #if (OPENSSL_VERSION_NUMBER < 0x10002002L || defined LIBRESSL_VERSION_NUMBER)
3043
3044    static ngx_int_t
3045    ngx_ssl_check_name(ngx_str_t *name, ASN1_STRING *pattern)
3046    {
3047        u_char  *s, *p, *end;
3048        size_t   slen, plen;
3049
3050        s = name->data;
3051        slen = name->len;
3052
3053        p = ASN1_STRING_data(pattern);
3054        plen = ASN1_STRING_length(pattern);
3055
3056        if (slen == plen && ngx_strncasecmp(s, p, plen) == 0) {
3057            return NGX_OK;
3058        }
3059
3060        if (plen > 2 && p[0] == '*' && p[1] == '.') {
3061            plen -= 1;
3062            p += 1;
3063
3064            end = s + slen;
3065            s = ngx_strlchr(s, end, '.');
3066
3067            if (s == NULL) {
3068                return NGX_ERROR;
3069            }
3070
3071            slen = end - s;
3072
3073            if (plen == slen && ngx_strncasecmp(s, p, plen) == 0) {
3074                return NGX_OK;
3075            }
3076        }
3077
```

```
3078      return NGX_ERROR;
3079  }
3080
3081  #endif
3082
3083
3084  ngx_int_t
3085  ngx_ssl_get_protocol(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3086  {
3087      s->data = (u_char *) SSL_get_version(c->ssl->connection);
3088      return NGX_OK;
3089  }
3090
3091
3092  ngx_int_t
3093  ngx_ssl_get_cipher_name(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3094  {
3095      s->data = (u_char *) SSL_get_cipher_name(c->ssl->connection);
3096      return NGX_OK;
3097  }
3098
3099
3100  ngx_int_t
3101  ngx_ssl_get_session_id(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3102  {
3103      u_char        *buf;
3104      SSL_SESSION   *sess;
3105      unsigned int   len;
3106
3107      sess = SSL_get0_session(c->ssl->connection);
3108      if (sess == NULL) {
3109          s->len = 0;
3110          return NGX_OK;
3111      }
3112
3113  #if OPENSSL_VERSION_NUMBER >= 0x0090800fL
3114
3115      buf = (u_char *) SSL_SESSION_get_id(sess, &len);
3116
3117  #else
3118
3119      buf = sess->session_id;
3120      len = sess->session_id_length;
3121
3122  #endif
3123
3124      s->len = 2 * len;
3125      s->data = ngx_pnalloc(pool, 2 * len);
3126      if (s->data == NULL) {
3127          return NGX_ERROR;
3128      }
3129
3130      ngx_hex_dump(s->data, buf, len);
3131
3132      return NGX_OK;
3133  }
3134
3135
3136  ngx_int_t
3137  ngx_ssl_get_session_reused(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3138  {
3139      if (SSL_session_reused(c->ssl->connection)) {
3140          ngx_str_set(s, "r");
3141
3142      } else {
3143          ngx_str_set(s, ".");
3144      }
3145
3146      return NGX_OK;
3147  }
3148
3149
3150  ngx_int_t
3151  ngx_ssl_get_server_name(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3152  {
3153  #ifdef SSL_CTRL_SET_TLSEXT_HOSTNAME
```

```
3154
3155        const char  *servername;
3156
3157        servername = SSL_get_servername(c->ssl->connection,
3158                                        TLSEXT_NAMETYPE_host_name);
3159        if (servername) {
3160            s->data = (u_char *) servername;
3161            s->len = ngx_strlen(servername);
3162            return NGX_OK;
3163        }
3164
3165    #endif
3166
3167        s->len = 0;
3168        return NGX_OK;
3169    }
3170
3171
3172    ngx_int_t
3173    ngx_ssl_get_raw_certificate(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3174    {
3175        size_t   len;
3176        BIO     *bio;
3177        X509    *cert;
3178
3179        s->len = 0;
3180
3181        cert = SSL_get_peer_certificate(c->ssl->connection);
3182        if (cert == NULL) {
3183            return NGX_OK;
3184        }
3185
3186        bio = BIO_new(BIO_s_mem());
3187        if (bio == NULL) {
3188            ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "BIO_new() failed");
3189            X509_free(cert);
3190            return NGX_ERROR;
3191        }
3192
3193        if (PEM_write_bio_X509(bio, cert) == 0) {
3194            ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "PEM_write_bio_X509() failed");
3195            goto failed;
3196        }
3197
3198        len = BIO_pending(bio);
3199        s->len = len;
3200
3201        s->data = ngx_pnalloc(pool, len);
3202        if (s->data == NULL) {
3203            goto failed;
3204        }
3205
3206        BIO_read(bio, s->data, len);
3207
3208        BIO_free(bio);
3209        X509_free(cert);
3210
3211        return NGX_OK;
3212
3213    failed:
3214
3215        BIO_free(bio);
3216        X509_free(cert);
3217
3218        return NGX_ERROR;
3219    }
3220
3221
3222    ngx_int_t
3223    ngx_ssl_get_certificate(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3224    {
3225        u_char      *p;
3226        size_t       len;
3227        ngx_uint_t   i;
3228        ngx_str_t    cert;
3229
```

```
    if (ngx_ssl_get_raw_certificate(c, pool, &cert) != NGX_OK) {
        return NGX_ERROR;
    }

    if (cert.len == 0) {
        s->len = 0;
        return NGX_OK;
    }

    len = cert.len - 1;

    for (i = 0; i < cert.len - 1; i++) {
        if (cert.data[i] == LF) {
            len++;
        }
    }

    s->len = len;
    s->data = ngx_pnalloc(pool, len);
    if (s->data == NULL) {
        return NGX_ERROR;
    }

    p = s->data;

    for (i = 0; i < cert.len - 1; i++) {
        *p++ = cert.data[i];
        if (cert.data[i] == LF) {
            *p++ = '\t';
        }
    }

    return NGX_OK;
}


ngx_int_t
ngx_ssl_get_subject_dn(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
{
    char        *p;
    size_t       len;
    X509        *cert;
    X509_NAME   *name;

    s->len = 0;

    cert = SSL_get_peer_certificate(c->ssl->connection);
    if (cert == NULL) {
        return NGX_OK;
    }

    name = X509_get_subject_name(cert);
    if (name == NULL) {
        X509_free(cert);
        return NGX_ERROR;
    }

    p = X509_NAME_oneline(name, NULL, 0);

    for (len = 0; p[len]; len++) { /* void */ }

    s->len = len;
    s->data = ngx_pnalloc(pool, len);
    if (s->data == NULL) {
        OPENSSL_free(p);
        X509_free(cert);
        return NGX_ERROR;
    }

    ngx_memcpy(s->data, p, len);

    OPENSSL_free(p);
    X509_free(cert);

    return NGX_OK;
}
```

```
3306
3307
3308 ngx_int_t
3309 ngx_ssl_get_issuer_dn(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3310 {
3311     char       *p;
3312     size_t      len;
3313     X509       *cert;
3314     X509_NAME  *name;
3315
3316     s->len = 0;
3317
3318     cert = SSL_get_peer_certificate(c->ssl->connection);
3319     if (cert == NULL) {
3320         return NGX_OK;
3321     }
3322
3323     name = X509_get_issuer_name(cert);
3324     if (name == NULL) {
3325         X509_free(cert);
3326         return NGX_ERROR;
3327     }
3328
3329     p = X509_NAME_oneline(name, NULL, 0);
3330
3331     for (len = 0; p[len]; len++) { /* void */ }
3332
3333     s->len = len;
3334     s->data = ngx_pnalloc(pool, len);
3335     if (s->data == NULL) {
3336         OPENSSL_free(p);
3337         X509_free(cert);
3338         return NGX_ERROR;
3339     }
3340
3341     ngx_memcpy(s->data, p, len);
3342
3343     OPENSSL_free(p);
3344     X509_free(cert);
3345
3346     return NGX_OK;
3347 }
3348
3349
3350 ngx_int_t
3351 ngx_ssl_get_serial_number(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3352 {
3353     size_t   len;
3354     X509    *cert;
3355     BIO     *bio;
3356
3357     s->len = 0;
3358
3359     cert = SSL_get_peer_certificate(c->ssl->connection);
3360     if (cert == NULL) {
3361         return NGX_OK;
3362     }
3363
3364     bio = BIO_new(BIO_s_mem());
3365     if (bio == NULL) {
3366         X509_free(cert);
3367         return NGX_ERROR;
3368     }
3369
3370     i2a_ASN1_INTEGER(bio, X509_get_serialNumber(cert));
3371     len = BIO_pending(bio);
3372
3373     s->len = len;
3374     s->data = ngx_pnalloc(pool, len);
3375     if (s->data == NULL) {
3376         BIO_free(bio);
3377         X509_free(cert);
3378         return NGX_ERROR;
3379     }
3380
3381     BIO_read(bio, s->data, len);
```

```
3382      BIO_free(bio);
3383      X509_free(cert);
3384
3385      return NGX_OK;
3386  }
3387
3388
3389  ngx_int_t
3390  ngx_ssl_get_fingerprint(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3391  {
3392      X509          *cert;
3393      unsigned int   len;
3394      u_char         buf[EVP_MAX_MD_SIZE];
3395
3396      s->len = 0;
3397
3398      cert = SSL_get_peer_certificate(c->ssl->connection);
3399      if (cert == NULL) {
3400          return NGX_OK;
3401      }
3402
3403      if (!X509_digest(cert, EVP_sha1(), buf, &len)) {
3404          X509_free(cert);
3405          return NGX_ERROR;
3406      }
3407
3408      s->len = 2 * len;
3409      s->data = ngx_pnalloc(pool, 2 * len);
3410      if (s->data == NULL) {
3411          X509_free(cert);
3412          return NGX_ERROR;
3413      }
3414
3415      ngx_hex_dump(s->data, buf, len);
3416
3417      X509_free(cert);
3418
3419      return NGX_OK;
3420  }
3421
3422
3423  ngx_int_t
3424  ngx_ssl_get_client_verify(ngx_connection_t *c, ngx_pool_t *pool, ngx_str_t *s)
3425  {
3426      X509  *cert;
3427
3428      if (SSL_get_verify_result(c->ssl->connection) != X509_V_OK) {
3429          ngx_str_set(s, "FAILED");
3430          return NGX_OK;
3431      }
3432
3433      cert = SSL_get_peer_certificate(c->ssl->connection);
3434
3435      if (cert) {
3436          ngx_str_set(s, "SUCCESS");
3437
3438      } else {
3439          ngx_str_set(s, "NONE");
3440      }
3441
3442      X509_free(cert);
3443
3444      return NGX_OK;
3445  }
3446
3447
3448  static void *
3449  ngx_openssl_create_conf(ngx_cycle_t *cycle)
3450  {
3451      ngx_openssl_conf_t  *oscf;
3452
3453      oscf = ngx_pcalloc(cycle->pool, sizeof(ngx_openssl_conf_t));
3454      if (oscf == NULL) {
3455          return NULL;
3456      }
3457
```

```
3458      /*
3459       * set by ngx_pcalloc():
3460       *
3461       *     oscf->engine = 0;
3462       */
3463
3464      return oscf;
3465  }
3466
3467
3468  static char *
3469  ngx_openssl_engine(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3470  {
3471  #ifndef OPENSSL_NO_ENGINE
3472
3473      ngx_openssl_conf_t *oscf = conf;
3474
3475      ENGINE     *engine;
3476      ngx_str_t  *value;
3477
3478      if (oscf->engine) {
3479          return "is duplicate";
3480      }
3481
3482      oscf->engine = 1;
3483
3484      value = cf->args->elts;
3485
3486      engine = ENGINE_by_id((const char *) value[1].data);
3487
3488      if (engine == NULL) {
3489          ngx_ssl_error(NGX_LOG_WARN, cf->log, 0,
3490                        "ENGINE_by_id(\"%V\") failed", &value[1]);
3491          return NGX_CONF_ERROR;
3492      }
3493
3494      if (ENGINE_set_default(engine, ENGINE_METHOD_ALL) == 0) {
3495          ngx_ssl_error(NGX_LOG_WARN, cf->log, 0,
3496                        "ENGINE_set_default(\"%V\", ENGINE_METHOD_ALL) failed",
3497                        &value[1]);
3498
3499          ENGINE_free(engine);
3500
3501          return NGX_CONF_ERROR;
3502      }
3503
3504      ENGINE_free(engine);
3505
3506      return NGX_CONF_OK;
3507
3508  #else
3509
3510      return "is not supported";
3511
3512  #endif
3513  }
3514
3515
3516  static void
3517  ngx_openssl_exit(ngx_cycle_t *cycle)
3518  {
3519      EVP_cleanup();
3520  #ifndef OPENSSL_NO_ENGINE
3521      ENGINE_cleanup();
3522  #endif
3523  }
```

One Level Up Top Level

src/core/ngx_slab.h - nginx-1.7.10

Data types defined

- ngx_slab_page_s
- ngx_slab_page_t
- ngx_slab_pool_t

Macros defined

- _NGX_SLAB_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_SLAB_H_INCLUDED_
9  #define _NGX_SLAB_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct ngx_slab_page_s  ngx_slab_page_t;
17
18  struct ngx_slab_page_s {
19      uintptr_t         slab;
20      ngx_slab_page_t  *next;
21      uintptr_t         prev;
22  };
23
24
25  typedef struct {
26      ngx_shmtx_sh_t    lock;
27
28      size_t            min_size;
29      size_t            min_shift;
30
31      ngx_slab_page_t  *pages;
32      ngx_slab_page_t  *last;
33      ngx_slab_page_t   free;
34
35      u_char           *start;
36      u_char           *end;
37
38      ngx_shmtx_t       mutex;
39
40      u_char           *log_ctx;
41      u_char            zero;
42
43      unsigned          log_nomem:1;
44
45      void             *data;
46      void             *addr;
47  } ngx_slab_pool_t;
48
49
50  void ngx_slab_init(ngx_slab_pool_t *pool);
51  void *ngx_slab_alloc(ngx_slab_pool_t *pool, size_t size);
52  void *ngx_slab_alloc_locked(ngx_slab_pool_t *pool, size_t size);
```

```
53 void *ngx_slab_calloc(ngx_slab_pool_t *pool, size_t size);
54 void *ngx_slab_calloc_locked(ngx_slab_pool_t *pool, size_t size);
55 void ngx_slab_free(ngx_slab_pool_t *pool, void *p);
56 void ngx_slab_free_locked(ngx_slab_pool_t *pool, void *p);
57
58
59 #endif /* _NGX_SLAB_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_shmtx.h - nginx-1.7.10

Data types defined

- ngx_shmtx_sh_t
- ngx_shmtx_t

Macros defined

- _NGX_SHMTX_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_SHMTX_H_INCLUDED_
9  #define _NGX_SHMTX_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct {
17      ngx_atomic_t   lock;
18  #if (NGX_HAVE_POSIX_SEM)
19      ngx_atomic_t   wait;
20  #endif
21  } ngx_shmtx_sh_t;
22
23
24  typedef struct {
25  #if (NGX_HAVE_ATOMIC_OPS)
26      ngx_atomic_t  *lock;
27  #if (NGX_HAVE_POSIX_SEM)
28      ngx_atomic_t  *wait;
29      ngx_uint_t     semaphore;
30      sem_t          sem;
31  #endif
32  #else
33      ngx_fd_t       fd;
34      u_char        *name;
35  #endif
36      ngx_uint_t     spin;
37  } ngx_shmtx_t;
38
39
40  ngx_int_t ngx_shmtx_create(ngx_shmtx_t *mtx, ngx_shmtx_sh_t *addr,
41      u_char *name);
42  void ngx_shmtx_destroy(ngx_shmtx_t *mtx);
43  ngx_uint_t ngx_shmtx_trylock(ngx_shmtx_t *mtx);
44  void ngx_shmtx_lock(ngx_shmtx_t *mtx);
45  void ngx_shmtx_unlock(ngx_shmtx_t *mtx);
46  ngx_uint_t ngx_shmtx_force_unlock(ngx_shmtx_t *mtx, ngx_pid_t pid);
47
48
49  #endif /* _NGX_SHMTX_H_INCLUDED_ */
```

src/core/ngx_shmtx.c - nginx-1.7.10

Functions defined

- ngx_shmtx_create
- ngx_shmtx_create
- ngx_shmtx_destroy
- ngx_shmtx_destroy
- ngx_shmtx_force_unlock
- ngx_shmtx_force_unlock
- ngx_shmtx_lock
- ngx_shmtx_lock
- ngx_shmtx_trylock
- ngx_shmtx_trylock
- ngx_shmtx_unlock
- ngx_shmtx_unlock
- ngx_shmtx_wakeup

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  #if (NGX_HAVE_ATOMIC_OPS)
13
14
15  static void ngx_shmtx_wakeup(ngx_shmtx_t *mtx);
16
17
18  ngx_int_t
19  ngx_shmtx_create(ngx_shmtx_t *mtx, ngx_shmtx_sh_t *addr, u_char *name)
20  {
21      mtx->lock = &addr->lock;
22
23      if (mtx->spin == (ngx_uint_t) -1) {
24          return NGX_OK;
25      }
26
27      mtx->spin = 2048;
28
29  #if (NGX_HAVE_POSIX_SEM)
30
31      mtx->wait = &addr->wait;
32
33      if (sem_init(&mtx->sem, 1, 0) == -1) {
34          ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_errno,
35                        "sem_init() failed");
```

```
    } else {
        mtx->semaphore = 1;
    }

#endif

    return NGX_OK;
}


void
ngx_shmtx_destroy(ngx_shmtx_t *mtx)
{
#if (NGX_HAVE_POSIX_SEM)

    if (mtx->semaphore) {
        if (sem_destroy(&mtx->sem) == -1) {
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_errno,
                          "sem_destroy() failed");
        }
    }

#endif
}


ngx_uint_t
ngx_shmtx_trylock(ngx_shmtx_t *mtx)
{
    return (*mtx->lock == 0 && ngx_atomic_cmp_set(mtx->lock, 0, ngx_pid));
}


void
ngx_shmtx_lock(ngx_shmtx_t *mtx)
{
    ngx_uint_t         i, n;

    ngx_log_debug0(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0, "shmtx lock");

    for ( ;; ) {

        if (*mtx->lock == 0 && ngx_atomic_cmp_set(mtx->lock, 0, ngx_pid)) {
            return;
        }

        if (ngx_ncpu > 1) {

            for (n = 1; n < mtx->spin; n <<= 1) {

                for (i = 0; i < n; i++) {
                    ngx_cpu_pause();
                }

                if (*mtx->lock == 0
                    && ngx_atomic_cmp_set(mtx->lock, 0, ngx_pid))
                {
                    return;
                }
            }
        }

#if (NGX_HAVE_POSIX_SEM)

         if (mtx->semaphore) {
             (void) ngx_atomic_fetch_add(mtx->wait, 1);

             if (*mtx->lock == 0 && ngx_atomic_cmp_set(mtx->lock, 0, ngx_pid)) {
                 (void) ngx_atomic_fetch_add(mtx->wait, -1);
                 return;
             }

             ngx_log_debug1(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
                            "shmtx wait %uA", *mtx->wait);

             while (sem_wait(&mtx->sem) == -1) {
```

```
                ngx_err_t  err;

                err = ngx_errno;

                if (err != NGX_EINTR) {
                    ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, err,
                                  "sem_wait() failed while waiting on shmtx");
                    break;
                }
            }

            ngx_log_debug0(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
                           "shmtx awoke");

            continue;
        }

#endif

        ngx_sched_yield();
    }
}


void
ngx_shmtx_unlock(ngx_shmtx_t *mtx)
{
    if (mtx->spin != (ngx_uint_t) -1) {
        ngx_log_debug0(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0, "shmtx unlock");
    }

    if (ngx_atomic_cmp_set(mtx->lock, ngx_pid, 0)) {
        ngx_shmtx_wakeup(mtx);
    }
}


ngx_uint_t
ngx_shmtx_force_unlock(ngx_shmtx_t *mtx, ngx_pid_t pid)
{
    ngx_log_debug0(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
                   "shmtx forced unlock");

    if (ngx_atomic_cmp_set(mtx->lock, pid, 0)) {
        ngx_shmtx_wakeup(mtx);
        return 1;
    }

    return 0;
}


static void
ngx_shmtx_wakeup(ngx_shmtx_t *mtx)
{
#if (NGX_HAVE_POSIX_SEM)
    ngx_atomic_uint_t  wait;

    if (!mtx->semaphore) {
        return;
    }

    for ( ;; ) {

        wait = *mtx->wait;

        if ((ngx_atomic_int_t) wait <= 0) {
            return;
        }

        if (ngx_atomic_cmp_set(mtx->wait, wait, wait - 1)) {
            break;
        }
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
```

```
                  "shmtx wake %uA", wait);

    if (sem_post(&mtx->sem) == -1) {
        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_errno,
                      "sem_post() failed while wake shmtx");
    }

#endif
}


#else


ngx_int_t
ngx_shmtx_create(ngx_shmtx_t *mtx, ngx_shmtx_sh_t *addr, u_char *name)
{
    if (mtx->name) {

        if (ngx_strcmp(name, mtx->name) == 0) {
            mtx->name = name;
            return NGX_OK;
        }

        ngx_shmtx_destroy(mtx);
    }

    mtx->fd = ngx_open_file(name, NGX_FILE_RDWR, NGX_FILE_CREATE_OR_OPEN,
                            NGX_FILE_DEFAULT_ACCESS);

    if (mtx->fd == NGX_INVALID_FILE) {
        ngx_log_error(NGX_LOG_EMERG, ngx_cycle->log, ngx_errno,
                      ngx_open_file_n " \"%s\" failed", name);
        return NGX_ERROR;
    }

    if (ngx_delete_file(name) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_errno,
                      ngx_delete_file_n " \"%s\" failed", name);
    }

    mtx->name = name;

    return NGX_OK;
}


void
ngx_shmtx_destroy(ngx_shmtx_t *mtx)
{
    if (ngx_close_file(mtx->fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", mtx->name);
    }
}


ngx_uint_t
ngx_shmtx_trylock(ngx_shmtx_t *mtx)
{
    ngx_err_t  err;

    err = ngx_trylock_fd(mtx->fd);

    if (err == 0) {
        return 1;
    }

    if (err == NGX_EAGAIN) {
        return 0;
    }

#if __osf__ /* Tru64 UNIX */

    if (err == NGX_EACCES) {
        return 0;
```

```
264     }
265
266 #endif
267
268     ngx_log_abort(err, ngx_trylock_fd_n " %s failed", mtx->name);
269
270     return 0;
271 }
272
273
274 void
275 ngx_shmtx_lock(ngx_shmtx_t *mtx)
276 {
277     ngx_err_t  err;
278
279     err = ngx_lock_fd(mtx->fd);
280
281     if (err == 0) {
282         return;
283     }
284
285     ngx_log_abort(err, ngx_lock_fd_n " %s failed", mtx->name);
286 }
287
288
289 void
290 ngx_shmtx_unlock(ngx_shmtx_t *mtx)
291 {
292     ngx_err_t  err;
293
294     err = ngx_unlock_fd(mtx->fd);
295
296     if (err == 0) {
297         return;
298     }
299
300     ngx_log_abort(err, ngx_unlock_fd_n " %s failed", mtx->name);
301 }
302
303
304 ngx_uint_t
305 ngx_shmtx_force_unlock(ngx_shmtx_t *mtx, ngx_pid_t pid)
306 {
307     return 0;
308 }
309
310 #endif
```

One Level Up Top Level

src/os/unix/ngx_process_cycle.c - nginx-1.7.10

Global variables defined

- master_process
- ngx_cache_loader_ctx
- ngx_cache_manager_ctx
- ngx_change_binary
- ngx_daemonized
- ngx_debug_quit
- ngx_exit_cycle
- ngx_exit_log
- ngx_exit_log_file
- ngx_exiting
- ngx_inherited
- ngx_new_binary
- ngx_noaccept
- ngx_noaccepting
- ngx_pid
- ngx_process
- ngx_quit
- ngx_reap
- ngx_reconfigure
- ngx_reopen
- ngx_restart
- ngx_sigalrm
- ngx_sigio
- ngx_terminate
- ngx_threaded
- ngx_threads
- ngx_threads_n

Functions defined

- ngx_cache_loader_process_handler

- ngx_cache_manager_process_cycle
- ngx_cache_manager_process_handler
- ngx_channel_handler
- ngx_master_process_cycle
- ngx_master_process_exit
- ngx_pass_open_channel
- ngx_reap_children
- ngx_signal_worker_processes
- ngx_single_process_cycle
- ngx_start_cache_manager_processes
- ngx_start_worker_processes
- ngx_wakeup_worker_threads
- ngx_worker_process_cycle
- ngx_worker_process_exit
- ngx_worker_process_init
- ngx_worker_thread_cycle

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_channel.h>
12
13
14  static void ngx_start_worker_processes(ngx_cycle_t *cycle, ngx_int_t n,
15      ngx_int_t type);
16  static void ngx_start_cache_manager_processes(ngx_cycle_t *cycle,
17      ngx_uint_t respawn);
18  static void ngx_pass_open_channel(ngx_cycle_t *cycle, ngx_channel_t *ch);
19  static void ngx_signal_worker_processes(ngx_cycle_t *cycle, int signo);
20  static ngx_uint_t ngx_reap_children(ngx_cycle_t *cycle);
21  static void ngx_master_process_exit(ngx_cycle_t *cycle);
22  static void ngx_worker_process_cycle(ngx_cycle_t *cycle, void *data);
23  static void ngx_worker_process_init(ngx_cycle_t *cycle, ngx_int_t worker);
24  static void ngx_worker_process_exit(ngx_cycle_t *cycle);
25  static void ngx_channel_handler(ngx_event_t *ev);
26  #if (NGX_THREADS)
27  static void ngx_wakeup_worker_threads(ngx_cycle_t *cycle);
28  static ngx_thread_value_t ngx_worker_thread_cycle(void *data);
29  #endif
30  static void ngx_cache_manager_process_cycle(ngx_cycle_t *cycle, void *data);
31  static void ngx_cache_manager_process_handler(ngx_event_t *ev);
32  static void ngx_cache_loader_process_handler(ngx_event_t *ev);
33
34
35  ngx_uint_t    ngx_process;
36  ngx_pid_t     ngx_pid;
```

```
ngx_uint_t    ngx_threaded;

sig_atomic_t  ngx_reap;
sig_atomic_t  ngx_sigio;
sig_atomic_t  ngx_sigalrm;
sig_atomic_t  ngx_terminate;
sig_atomic_t  ngx_quit;
sig_atomic_t  ngx_debug_quit;
ngx_uint_t    ngx_exiting;
sig_atomic_t  ngx_reconfigure;
sig_atomic_t  ngx_reopen;

sig_atomic_t  ngx_change_binary;
ngx_pid_t     ngx_new_binary;
ngx_uint_t    ngx_inherited;
ngx_uint_t    ngx_daemonized;

sig_atomic_t  ngx_noaccept;
ngx_uint_t    ngx_noaccepting;
ngx_uint_t    ngx_restart;


#if (NGX_THREADS)
volatile ngx_thread_t  ngx_threads[NGX_MAX_THREADS];
ngx_int_t              ngx_threads_n;
#endif


static u_char  master_process[] = "master process";


static ngx_cache_manager_ctx_t  ngx_cache_manager_ctx = {
    ngx_cache_manager_process_handler, "cache manager process", 0
};

static ngx_cache_manager_ctx_t  ngx_cache_loader_ctx = {
    ngx_cache_loader_process_handler, "cache loader process", 60000
};


static ngx_cycle_t      ngx_exit_cycle;
static ngx_log_t        ngx_exit_log;
static ngx_open_file_t  ngx_exit_log_file;


void
ngx_master_process_cycle(ngx_cycle_t *cycle)
{
    char              *title;
    u_char            *p;
    size_t             size;
    ngx_int_t          i;
    ngx_uint_t         n, sigio;
    sigset_t           set;
    struct itimerval   itv;
    ngx_uint_t         live;
    ngx_msec_t         delay;
    ngx_listening_t   *ls;
    ngx_core_conf_t   *ccf;

    sigemptyset(&set);
    sigaddset(&set, SIGCHLD);
    sigaddset(&set, SIGALRM);
     sigaddset(&set, SIGIO);
     sigaddset(&set, SIGINT);
     sigaddset(&set, ngx_signal_value(NGX_RECONFIGURE_SIGNAL));
     sigaddset(&set, ngx_signal_value(NGX_REOPEN_SIGNAL));
     sigaddset(&set, ngx_signal_value(NGX_NOACCEPT_SIGNAL));
     sigaddset(&set, ngx_signal_value(NGX_TERMINATE_SIGNAL));
     sigaddset(&set, ngx_signal_value(NGX_SHUTDOWN_SIGNAL));
     sigaddset(&set, ngx_signal_value(NGX_CHANGEBIN_SIGNAL));

     if (sigprocmask(SIG_BLOCK, &set, NULL) == -1) {
         ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                       "sigprocmask() failed");
     }
```

```
    sigemptyset(&set);


    size = sizeof(master_process);

    for (i = 0; i < ngx_argc; i++) {
        size += ngx_strlen(ngx_argv[i]) + 1;
    }

    title = ngx_pnalloc(cycle->pool, size);
    if (title == NULL) {
        /* fatal */
        exit(2);
    }

    p = ngx_cpymem(title, master_process, sizeof(master_process) - 1);
    for (i = 0; i < ngx_argc; i++) {
        *p++ = ' ';
        p = ngx_cpystrn(p, (u_char *) ngx_argv[i], size);
    }

    ngx_setproctitle(title);


    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);

    ngx_start_worker_processes(cycle, ccf->worker_processes,
                               NGX_PROCESS_RESPAWN);
    ngx_start_cache_manager_processes(cycle, 0);

    ngx_new_binary = 0;
    delay = 0;
    sigio = 0;
    live = 1;

    for ( ;; ) {
        if (delay) {
            if (ngx_sigalrm) {
                sigio = 0;
                delay *= 2;
                ngx_sigalrm = 0;
            }

            ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "termination cycle: %d", delay);

            itv.it_interval.tv_sec = 0;
            itv.it_interval.tv_usec = 0;
            itv.it_value.tv_sec = delay / 1000;
            itv.it_value.tv_usec = (delay % 1000 ) * 1000;

            if (setitimer(ITIMER_REAL, &itv, NULL) == -1) {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                              "setitimer() failed");
            }
        }

        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, cycle->log, 0, "sigsuspend");

        sigsuspend(&set);

        ngx_time_update();

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "wake up, sigio %i", sigio);

        if (ngx_reap) {
            ngx_reap = 0;
            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, cycle->log, 0, "reap children");

            live = ngx_reap_children(cycle);
        }

        if (!live && (ngx_terminate || ngx_quit)) {
            ngx_master_process_exit(cycle);
```

```
        }

        if (ngx_terminate) {
            if (delay == 0) {
                delay = 50;
            }

            if (sigio) {
                sigio--;
                continue;
            }

            sigio = ccf->worker_processes + 2 /* cache processes */;

            if (delay > 1000) {
                ngx_signal_worker_processes(cycle, SIGKILL);
            } else {
                ngx_signal_worker_processes(cycle,
                                            ngx_signal_value(NGX_TERMINATE_SIGNAL));
            }

            continue;
        }

        if (ngx_quit) {
            ngx_signal_worker_processes(cycle,
                                        ngx_signal_value(NGX_SHUTDOWN_SIGNAL));

            ls = cycle->listening.elts;
            for (n = 0; n < cycle->listening.nelts; n++) {
                if (ngx_close_socket(ls[n].fd) == -1) {
                    ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_socket_errno,
                                  ngx_close_socket_n " %V failed",
                                  &ls[n].addr_text);
                }
            }
            cycle->listening.nelts = 0;

            continue;
        }

        if (ngx_reconfigure) {
            ngx_reconfigure = 0;

            if (ngx_new_binary) {
                ngx_start_worker_processes(cycle, ccf->worker_processes,
                                           NGX_PROCESS_RESPAWN);
                ngx_start_cache_manager_processes(cycle, 0);
                ngx_noaccepting = 0;

                continue;
            }

            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "reconfiguring");

            cycle = ngx_init_cycle(cycle);
            if (cycle == NULL) {
                cycle = (ngx_cycle_t *) ngx_cycle;
                continue;
            }

            ngx_cycle = cycle;
            ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx,
                                                   ngx_core_module);
            ngx_start_worker_processes(cycle, ccf->worker_processes,
                                       NGX_PROCESS_JUST_RESPAWN);
            ngx_start_cache_manager_processes(cycle, 1);

            /* allow new processes to start */
            ngx_msleep(100);

            live = 1;
            ngx_signal_worker_processes(cycle,
                                        ngx_signal_value(NGX_SHUTDOWN_SIGNAL));
        }

```

```
        if (ngx_restart) {
            ngx_restart = 0;
            ngx_start_worker_processes(cycle, ccf->worker_processes,
                                       NGX_PROCESS_RESPAWN);
            ngx_start_cache_manager_processes(cycle, 0);
            live = 1;
        }

        if (ngx_reopen) {
            ngx_reopen = 0;
            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "reopening logs");
            ngx_reopen_files(cycle, ccf->user);
            ngx_signal_worker_processes(cycle,
                                        ngx_signal_value(NGX_REOPEN_SIGNAL));
        }

        if (ngx_change_binary) {
            ngx_change_binary = 0;
            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "changing binary");
            ngx_new_binary = ngx_exec_new_binary(cycle, ngx_argv);
        }

        if (ngx_noaccept) {
            ngx_noaccept = 0;
            ngx_noaccepting = 1;
            ngx_signal_worker_processes(cycle,
                                        ngx_signal_value(NGX_SHUTDOWN_SIGNAL));
        }
    }
}


void
ngx_single_process_cycle(ngx_cycle_t *cycle)
{
    ngx_uint_t  i;

    if (ngx_set_environment(cycle, NULL) == NULL) {
        /* fatal */
        exit(2);
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->init_process) {
            if (ngx_modules[i]->init_process(cycle) == NGX_ERROR) {
                /* fatal */
                exit(2);
            }
        }
    }

    for ( ;; ) {
        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, cycle->log, 0, "worker cycle");

        ngx_process_events_and_timers(cycle);

        if (ngx_terminate || ngx_quit) {

            for (i = 0; ngx_modules[i]; i++) {
                if (ngx_modules[i]->exit_process) {
                    ngx_modules[i]->exit_process(cycle);
                }
            }

            ngx_master_process_exit(cycle);
        }

        if (ngx_reconfigure) {
            ngx_reconfigure = 0;
            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "reconfiguring");

            cycle = ngx_init_cycle(cycle);
            if (cycle == NULL) {
                cycle = (ngx_cycle_t *) ngx_cycle;
                continue;
            }
```

```
            ngx_cycle = cycle;
        }

        if (ngx_reopen) {
            ngx_reopen = 0;
            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "reopening logs");
            ngx_reopen_files(cycle, (ngx_uid_t) -1);
        }
    }
}


static void
ngx_start_worker_processes(ngx_cycle_t *cycle, ngx_int_t n, ngx_int_t type)
{
    ngx_int_t      i;
    ngx_channel_t  ch;

    ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "start worker processes");

    ngx_memzero(&ch, sizeof(ngx_channel_t));

    ch.command = NGX_CMD_OPEN_CHANNEL;

    for (i = 0; i < n; i++) {

        ngx_spawn_process(cycle, ngx_worker_process_cycle,
                          (void *) (intptr_t) i, "worker process", type);

        ch.pid = ngx_processes[ngx_process_slot].pid;
        ch.slot = ngx_process_slot;
        ch.fd = ngx_processes[ngx_process_slot].channel[0];

        ngx_pass_open_channel(cycle, &ch);
    }
}


static void
ngx_start_cache_manager_processes(ngx_cycle_t *cycle, ngx_uint_t respawn)
{
    ngx_uint_t       i, manager, loader;
    ngx_path_t     **path;
    ngx_channel_t    ch;

    manager = 0;
    loader = 0;

    path = ngx_cycle->paths.elts;
    for (i = 0; i < ngx_cycle->paths.nelts; i++) {

        if (path[i]->manager) {
            manager = 1;
        }

        if (path[i]->loader) {
            loader = 1;
        }
    }

    if (manager == 0) {
        return;
    }

    ngx_spawn_process(cycle, ngx_cache_manager_process_cycle,
                      &ngx_cache_manager_ctx, "cache manager process",
                      respawn ? NGX_PROCESS_JUST_RESPAWN : NGX_PROCESS_RESPAWN);

    ngx_memzero(&ch, sizeof(ngx_channel_t));

    ch.command = NGX_CMD_OPEN_CHANNEL;
    ch.pid = ngx_processes[ngx_process_slot].pid;
    ch.slot = ngx_process_slot;
    ch.fd = ngx_processes[ngx_process_slot].channel[0];

```

```
    ngx_pass_open_channel(cycle, &ch);

    if (loader == 0) {
        return;
    }

    ngx_spawn_process(cycle, ngx_cache_manager_process_cycle,
                      &ngx_cache_loader_ctx, "cache loader process",
                      respawn ? NGX_PROCESS_JUST_SPAWN : NGX_PROCESS_NORESPAWN);

    ch.command = NGX_CMD_OPEN_CHANNEL;
    ch.pid = ngx_processes[ngx_process_slot].pid;
    ch.slot = ngx_process_slot;
    ch.fd = ngx_processes[ngx_process_slot].channel[0];

    ngx_pass_open_channel(cycle, &ch);
}


static void
ngx_pass_open_channel(ngx_cycle_t *cycle, ngx_channel_t *ch)
{
    ngx_int_t  i;

    for (i = 0; i < ngx_last_process; i++) {

        if (i == ngx_process_slot
            || ngx_processes[i].pid == -1
            || ngx_processes[i].channel[0] == -1)
        {
            continue;
        }

        ngx_log_debug6(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                       "pass channel s:%d pid:%P fd:%d to s:%i pid:%P fd:%d",
                       ch->slot, ch->pid, ch->fd,
                       i, ngx_processes[i].pid,
                       ngx_processes[i].channel[0]);

        /* TODO: NGX_AGAIN */

        ngx_write_channel(ngx_processes[i].channel[0],
                          ch, sizeof(ngx_channel_t), cycle->log);
    }
}


static void
ngx_signal_worker_processes(ngx_cycle_t *cycle, int signo)
{
    ngx_int_t      i;
    ngx_err_t      err;
    ngx_channel_t  ch;

    ngx_memzero(&ch, sizeof(ngx_channel_t));

#if (NGX_BROKEN_SCM_RIGHTS)

    ch.command = 0;

#else

    switch (signo) {

    case ngx_signal_value(NGX_SHUTDOWN_SIGNAL):
        ch.command = NGX_CMD_QUIT;
        break;

    case ngx_signal_value(NGX_TERMINATE_SIGNAL):
        ch.command = NGX_CMD_TERMINATE;
        break;

    case ngx_signal_value(NGX_REOPEN_SIGNAL):
        ch.command = NGX_CMD_REOPEN;
        break;

```

```
    default:
        ch.command = 0;
    }

#endif

    ch.fd = -1;


    for (i = 0; i < ngx_last_process; i++) {

        ngx_log_debug7(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "child: %d %P e:%d t:%d d:%d r:%d j:%d",
                       i,
                       ngx_processes[i].pid,
                       ngx_processes[i].exiting,
                       ngx_processes[i].exited,
                       ngx_processes[i].detached,
                       ngx_processes[i].respawn,
                       ngx_processes[i].just_spawn);

        if (ngx_processes[i].detached || ngx_processes[i].pid == -1) {
            continue;
        }

        if (ngx_processes[i].just_spawn) {
            ngx_processes[i].just_spawn = 0;
            continue;
        }

        if (ngx_processes[i].exiting
            && signo == ngx_signal_value(NGX_SHUTDOWN_SIGNAL))
        {
            continue;
        }

        if (ch.command) {
            if (ngx_write_channel(ngx_processes[i].channel[0],
                                  &ch, sizeof(ngx_channel_t), cycle->log)
                == NGX_OK)
            {
                if (signo != ngx_signal_value(NGX_REOPEN_SIGNAL)) {
                    ngx_processes[i].exiting = 1;
                }

                continue;
            }
        }

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                       "kill (%P, %d)", ngx_processes[i].pid, signo);

        if (kill(ngx_processes[i].pid, signo) == -1) {
            err = ngx_errno;
            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          "kill(%P, %d) failed", ngx_processes[i].pid, signo);

            if (err == NGX_ESRCH) {
                ngx_processes[i].exited = 1;
                ngx_processes[i].exiting = 0;
                ngx_reap = 1;
            }

            continue;
        }

        if (signo != ngx_signal_value(NGX_REOPEN_SIGNAL)) {
            ngx_processes[i].exiting = 1;
        }
    }
}


static ngx_uint_t
ngx_reap_children(ngx_cycle_t *cycle)
{
```

```
    ngx_int_t         i, n;
    ngx_uint_t        live;
    ngx_channel_t     ch;
    ngx_core_conf_t  *ccf;

    ngx_memzero(&ch, sizeof(ngx_channel_t));

    ch.command = NGX_CMD_CLOSE_CHANNEL;
    ch.fd = -1;

    live = 0;
    for (i = 0; i < ngx_last_process; i++) {

        ngx_log_debug7(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "child: %d %P e:%d t:%d d:%d r:%d j:%d",
                       i,
                       ngx_processes[i].pid,
                       ngx_processes[i].exiting,
                       ngx_processes[i].exited,
                       ngx_processes[i].detached,
                       ngx_processes[i].respawn,
                       ngx_processes[i].just_spawn);

        if (ngx_processes[i].pid == -1) {
            continue;
        }

        if (ngx_processes[i].exited) {

            if (!ngx_processes[i].detached) {
                ngx_close_channel(ngx_processes[i].channel, cycle->log);

                ngx_processes[i].channel[0] = -1;
                ngx_processes[i].channel[1] = -1;

                ch.pid = ngx_processes[i].pid;
                ch.slot = i;

                for (n = 0; n < ngx_last_process; n++) {
                    if (ngx_processes[n].exited
                        || ngx_processes[n].pid == -1
                        || ngx_processes[n].channel[0] == -1)
                    {
                        continue;
                    }

                    ngx_log_debug3(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                                   "pass close channel s:%i pid:%P to:%P",
                                   ch.slot, ch.pid, ngx_processes[n].pid);

                    /* TODO: NGX_AGAIN */

                    ngx_write_channel(ngx_processes[n].channel[0],
                                      &ch, sizeof(ngx_channel_t), cycle->log);
                }
            }

            if (ngx_processes[i].respawn
                && !ngx_processes[i].exiting
                && !ngx_terminate
                && !ngx_quit)
            {
                if (ngx_spawn_process(cycle, ngx_processes[i].proc,
                                      ngx_processes[i].data,
                                      ngx_processes[i].name, i)
                    == NGX_INVALID_PID)
                {
                    ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                                  "could not respawn %s",
                                  ngx_processes[i].name);
                    continue;
                }


                ch.command = NGX_CMD_OPEN_CHANNEL;
                ch.pid = ngx_processes[ngx_process_slot].pid;
```

```
                ch.slot = ngx_process_slot;
                ch.fd = ngx_processes[ngx_process_slot].channel[0];

                ngx_pass_open_channel(cycle, &ch);

                live = 1;

                continue;
            }

            if (ngx_processes[i].pid == ngx_new_binary) {

                ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx,
                                                       ngx_core_module);

                if (ngx_rename_file((char *) ccf->oldpid.data,
                                    (char *) ccf->pid.data)
                    == NGX_FILE_ERROR)
                {
                    ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                                  ngx_rename_file_n " %s back to %s failed "
                                  "after the new binary process \"%s\" exited",
                                  ccf->oldpid.data, ccf->pid.data, ngx_argv[0]);
                }

                ngx_new_binary = 0;
                if (ngx_noaccepting) {
                    ngx_restart = 1;
                    ngx_noaccepting = 0;
                }
            }

            if (i == ngx_last_process - 1) {
                ngx_last_process--;

            } else {
                ngx_processes[i].pid = -1;
            }

        } else if (ngx_processes[i].exiting || !ngx_processes[i].detached) {
            live = 1;
        }
    }

    return live;
}


static void
ngx_master_process_exit(ngx_cycle_t *cycle)
{
    ngx_uint_t  i;

    ngx_delete_pidfile(cycle);

    ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "exit");

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->exit_master) {
            ngx_modules[i]->exit_master(cycle);
        }
    }

    ngx_close_listening_sockets(cycle);

    /*
     * Copy ngx_cycle->log related data to the special static exit cycle,
     * log, and log file structures enough to allow a signal handler to log.
     * The handler may be called when standard ngx_cycle->log allocated from
     * ngx_cycle->pool is already destroyed.
     */


    ngx_exit_log = *ngx_log_get_file_log(ngx_cycle->log);

    ngx_exit_log_file.fd = ngx_exit_log.file->fd;
```

```
    ngx_exit_log.file = &ngx_exit_log_file;
    ngx_exit_log.next = NULL;
    ngx_exit_log.writer = NULL;

    ngx_exit_cycle.log = &ngx_exit_log;
    ngx_exit_cycle.files = ngx_cycle->files;
    ngx_exit_cycle.files_n = ngx_cycle->files_n;
    ngx_cycle = &ngx_exit_cycle;

    ngx_destroy_pool(cycle->pool);

    exit(0);
}


static void
ngx_worker_process_cycle(ngx_cycle_t *cycle, void *data)
{
    ngx_int_t worker = (intptr_t) data;

    ngx_uint_t         i;
    ngx_connection_t  *c;

    ngx_process = NGX_PROCESS_WORKER;

    ngx_worker_process_init(cycle, worker);

    ngx_setproctitle("worker process");

#if (NGX_THREADS)
    {
    ngx_int_t         n;
    ngx_err_t         err;
    ngx_core_conf_t  *ccf;

    ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);

    if (ngx_threads_n) {
        if (ngx_init_threads(ngx_threads_n, ccf->thread_stack_size, cycle)
            == NGX_ERROR)
        {
            /* fatal */
            exit(2);
        }

        err = ngx_thread_key_create(&ngx_core_tls_key);
        if (err != 0) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          ngx_thread_key_create_n " failed");
            /* fatal */
            exit(2);
        }

        for (n = 0; n < ngx_threads_n; n++) {

            ngx_threads[n].cv = ngx_cond_init(cycle->log);

            if (ngx_threads[n].cv == NULL) {
                /* fatal */
                exit(2);
            }

            if (ngx_create_thread((ngx_tid_t *) &ngx_threads[n].tid,
                                  ngx_worker_thread_cycle,
                                  (void *) &ngx_threads[n], cycle->log)
                != 0)
            {
                /* fatal */
                exit(2);
            }
        }
    }
    }
#endif

    for ( ;; ) {
```

```
797
798         if (ngx_exiting) {
799
800             c = cycle->connections;
801
802             for (i = 0; i < cycle->connection_n; i++) {
803
804                 /* THREAD: lock */
805
806                 if (c[i].fd != -1 && c[i].idle) {
807                     c[i].close = 1;
808                     c[i].read->handler(c[i].read);
809                 }
810             }
811
812             ngx_event_cancel_timers();
813
814             if (ngx_event_timer_rbtree.root == ngx_event_timer_rbtree.sentinel)
815             {
816                 ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "exiting");
817
818                 ngx_worker_process_exit(cycle);
819             }
820         }
821
822         ngx_log_debug0(NGX_LOG_DEBUG_EVENT, cycle->log, 0, "worker cycle");
823
824         ngx_process_events_and_timers(cycle);
825
826         if (ngx_terminate) {
827             ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "exiting");
828
829             ngx_worker_process_exit(cycle);
830         }
831
832         if (ngx_quit) {
833             ngx_quit = 0;
834             ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0,
835                           "gracefully shutting down");
836             ngx_setproctitle("worker process is shutting down");
837
838             if (!ngx_exiting) {
839                 ngx_close_listening_sockets(cycle);
840                 ngx_exiting = 1;
841             }
842         }
843
844         if (ngx_reopen) {
845             ngx_reopen = 0;
846             ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "reopening logs");
847             ngx_reopen_files(cycle, -1);
848         }
849     }
850 }
851
852
853 static void
854 ngx_worker_process_init(ngx_cycle_t *cycle, ngx_int_t worker)
855 {
856     sigset_t          set;
857     uint64_t          cpu_affinity;
858     ngx_int_t         n;
859     ngx_uint_t        i;
860     struct rlimit     rlmt;
861     ngx_core_conf_t  *ccf;
862     ngx_listening_t  *ls;
863
864     if (ngx_set_environment(cycle, NULL) == NULL) {
865         /* fatal */
866         exit(2);
867     }
868
869     ccf = (ngx_core_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_core_module);
870
871     if (worker >= 0 && ccf->priority != 0) {
872         if (setpriority(PRIO_PROCESS, 0, ccf->priority) == -1) {
```

```
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "setpriority(%d) failed", ccf->priority);
        }
    }

    if (ccf->rlimit_nofile != NGX_CONF_UNSET) {
        rlmt.rlim_cur = (rlim_t) ccf->rlimit_nofile;
        rlmt.rlim_max = (rlim_t) ccf->rlimit_nofile;

        if (setrlimit(RLIMIT_NOFILE, &rlmt) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "setrlimit(RLIMIT_NOFILE, %i) failed",
                          ccf->rlimit_nofile);
        }
    }

    if (ccf->rlimit_core != NGX_CONF_UNSET) {
        rlmt.rlim_cur = (rlim_t) ccf->rlimit_core;
        rlmt.rlim_max = (rlim_t) ccf->rlimit_core;

        if (setrlimit(RLIMIT_CORE, &rlmt) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "setrlimit(RLIMIT_CORE, %O) failed",
                          ccf->rlimit_core);
        }
    }

#ifdef RLIMIT_SIGPENDING
    if (ccf->rlimit_sigpending != NGX_CONF_UNSET) {
        rlmt.rlim_cur = (rlim_t) ccf->rlimit_sigpending;
        rlmt.rlim_max = (rlim_t) ccf->rlimit_sigpending;

        if (setrlimit(RLIMIT_SIGPENDING, &rlmt) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "setrlimit(RLIMIT_SIGPENDING, %i) failed",
                          ccf->rlimit_sigpending);
        }
    }
#endif

    if (geteuid() == 0) {
        if (setgid(ccf->group) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "setgid(%d) failed", ccf->group);
            /* fatal */
            exit(2);
        }

        if (initgroups(ccf->username, ccf->group) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "initgroups(%s, %d) failed",
                          ccf->username, ccf->group);
        }

        if (setuid(ccf->user) == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "setuid(%d) failed", ccf->user);
            /* fatal */
            exit(2);
        }
    }

    if (worker >= 0) {
        cpu_affinity = ngx_get_cpu_affinity(worker);

        if (cpu_affinity) {
            ngx_setaffinity(cpu_affinity, cycle->log);
        }
    }

#if (NGX_HAVE_PR_SET_DUMPABLE)

    /* allow coredump after setuid() in Linux 2.4.x */

    if (prctl(PR_SET_DUMPABLE, 1, 0, 0, 0) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
```

```
                      "prctl(PR_SET_DUMPABLE) failed");
    }

#endif

    if (ccf->working_directory.len) {
        if (chdir((char *) ccf->working_directory.data) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "chdir(\"%s\") failed", ccf->working_directory.data);
            /* fatal */
            exit(2);
        }
    }

    sigemptyset(&set);

    if (sigprocmask(SIG_SETMASK, &set, NULL) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "sigprocmask() failed");
    }

    srandom((ngx_pid << 16) ^ ngx_time());

    /*
     * disable deleting previous events for the listening sockets because
     * in the worker processes there are no events at all at this point
     */
    ls = cycle->listening.elts;
    for (i = 0; i < cycle->listening.nelts; i++) {
        ls[i].previous = NULL;
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->init_process) {
            if (ngx_modules[i]->init_process(cycle) == NGX_ERROR) {
                /* fatal */
                exit(2);
            }
        }
    }

    for (n = 0; n < ngx_last_process; n++) {

        if (ngx_processes[n].pid == -1) {
            continue;
        }

        if (n == ngx_process_slot) {
            continue;
        }

         if (ngx_processes[n].channel[1] == -1) {
             continue;
         }

         if (close(ngx_processes[n].channel[1]) == -1) {
             ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                           "close() channel failed");
         }
     }

     if (close(ngx_processes[ngx_process_slot].channel[0]) == -1) {
         ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                       "close() channel failed");
     }

 #if 0
     ngx_last_process = 0;
 #endif

     if (ngx_add_channel_event(cycle, ngx_channel, NGX_READ_EVENT,
                               ngx_channel_handler)
         == NGX_ERROR)
     {
         /* fatal */
         exit(2);
```

```
    }
}


static void
ngx_worker_process_exit(ngx_cycle_t *cycle)
{
    ngx_uint_t         i;
    ngx_connection_t  *c;

#if (NGX_THREADS)
    ngx_terminate = 1;

    ngx_wakeup_worker_threads(cycle);
#endif

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->exit_process) {
            ngx_modules[i]->exit_process(cycle);
        }
    }

    if (ngx_exiting) {
        c = cycle->connections;
        for (i = 0; i < cycle->connection_n; i++) {
            if (c[i].fd != -1
                && c[i].read
                && !c[i].read->accept
                && !c[i].read->channel
                && !c[i].read->resolver)
            {
                ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                              "*%uA open socket #%d left in connection %ui",
                              c[i].number, c[i].fd, i);
                ngx_debug_quit = 1;
            }
        }

        if (ngx_debug_quit) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, 0, "aborting");
            ngx_debug_point();
        }
    }

    /*
     * Copy ngx_cycle->log related data to the special static exit cycle,
     * log, and log file structures enough to allow a signal handler to log.
     * The handler may be called when standard ngx_cycle->log allocated from
     * ngx_cycle->pool is already destroyed.
     */

    ngx_exit_log = *ngx_log_get_file_log(ngx_cycle->log);

    ngx_exit_log_file.fd = ngx_exit_log.file->fd;
    ngx_exit_log.file = &ngx_exit_log_file;
    ngx_exit_log.next = NULL;
    ngx_exit_log.writer = NULL;

    ngx_exit_cycle.log = &ngx_exit_log;
    ngx_exit_cycle.files = ngx_cycle->files;
    ngx_exit_cycle.files_n = ngx_cycle->files_n;
    ngx_cycle = &ngx_exit_cycle;

    ngx_destroy_pool(cycle->pool);

    ngx_log_error(NGX_LOG_NOTICE, ngx_cycle->log, 0, "exit");

    exit(0);
}


static void
ngx_channel_handler(ngx_event_t *ev)
{
    ngx_int_t          n;
    ngx_channel_t      ch;
```

```
    ngx_connection_t  *c;

    if (ev->timedout) {
        ev->timedout = 0;
        return;
    }

    c = ev->data;

    ngx_log_debug0(NGX_LOG_DEBUG_CORE, ev->log, 0, "channel handler");

    for ( ;; ) {

        n = ngx_read_channel(c->fd, &ch, sizeof(ngx_channel_t), ev->log);

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, ev->log, 0, "channel: %i", n);

        if (n == NGX_ERROR) {

            if (ngx_event_flags & NGX_USE_EPOLL_EVENT) {
                ngx_del_conn(c, 0);
            }

            ngx_close_connection(c);
            return;
        }

        if (ngx_event_flags & NGX_USE_EVENTPORT_EVENT) {
            if (ngx_add_event(ev, NGX_READ_EVENT, 0) == NGX_ERROR) {
                return;
            }
        }

        if (n == NGX_AGAIN) {
            return;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, ev->log, 0,
                       "channel command: %d", ch.command);

        switch (ch.command) {

        case NGX_CMD_QUIT:
            ngx_quit = 1;
            break;

        case NGX_CMD_TERMINATE:
            ngx_terminate = 1;
            break;

        case NGX_CMD_REOPEN:
            ngx_reopen = 1;
            break;

        case NGX_CMD_OPEN_CHANNEL:

            ngx_log_debug3(NGX_LOG_DEBUG_CORE, ev->log, 0,
                           "get channel s:%i pid:%P fd:%d",
                           ch.slot, ch.pid, ch.fd);

            ngx_processes[ch.slot].pid = ch.pid;
            ngx_processes[ch.slot].channel[0] = ch.fd;
            break;

        case NGX_CMD_CLOSE_CHANNEL:

            ngx_log_debug4(NGX_LOG_DEBUG_CORE, ev->log, 0,
                           "close channel s:%i pid:%P our:%P fd:%d",
                           ch.slot, ch.pid, ngx_processes[ch.slot].pid,
                           ngx_processes[ch.slot].channel[0]);

            if (close(ngx_processes[ch.slot].channel[0]) == -1) {
                ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
                              "close() channel failed");
            }

```

```
            ngx_processes[ch.slot].channel[0] = -1;
            break;
        }
    }
}


#if (NGX_THREADS)

static void
ngx_wakeup_worker_threads(ngx_cycle_t *cycle)
{
    ngx_int_t   i;
    ngx_uint_t  live;

    for ( ;; ) {

        live = 0;

        for (i = 0; i < ngx_threads_n; i++) {
            if (ngx_threads[i].state < NGX_THREAD_EXIT) {
                if (ngx_cond_signal(ngx_threads[i].cv) == NGX_ERROR) {
                    ngx_threads[i].state = NGX_THREAD_DONE;

                } else {
                    live = 1;
                }
            }

            if (ngx_threads[i].state == NGX_THREAD_EXIT) {
                ngx_thread_join(ngx_threads[i].tid, NULL);
                ngx_threads[i].state = NGX_THREAD_DONE;
            }
        }

        if (live == 0) {
            ngx_log_debug0(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                           "all worker threads are joined");

            /* STUB */
            ngx_done_events(cycle);

            return;
        }

        ngx_sched_yield();
    }
}


static ngx_thread_value_t
ngx_worker_thread_cycle(void *data)
{
    ngx_thread_t  *thr = data;

    sigset_t          set;
    ngx_err_t         err;
    ngx_core_tls_t   *tls;
    ngx_cycle_t      *cycle;

    cycle = (ngx_cycle_t *) ngx_cycle;

    sigemptyset(&set);
    sigaddset(&set, ngx_signal_value(NGX_RECONFIGURE_SIGNAL));
    sigaddset(&set, ngx_signal_value(NGX_REOPEN_SIGNAL));
    sigaddset(&set, ngx_signal_value(NGX_CHANGEBIN_SIGNAL));

    err = ngx_thread_sigmask(SIG_BLOCK, &set, NULL);
    if (err) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                      ngx_thread_sigmask_n " failed");
        return (ngx_thread_value_t) 1;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                   "thread " NGX_TID_T_FMT " started", ngx_thread_self());
```

```
1253
1254        ngx_setthrtitle("worker thread");
1255
1256        tls = ngx_calloc(sizeof(ngx_core_tls_t), cycle->log);
1257        if (tls == NULL) {
1258            return (ngx_thread_value_t) 1;
1259        }
1260
1261        err = ngx_thread_set_tls(ngx_core_tls_key, tls);
1262        if (err != 0) {
1263            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
1264                          ngx_thread_set_tls_n " failed");
1265            return (ngx_thread_value_t) 1;
1266        }
1267
1268        for ( ;; ) {
1269            thr->state = NGX_THREAD_FREE;
1270
1271    #if 0
1272            if (ngx_cond_wait(thr->cv, ngx_posted_events_mutex) == NGX_ERROR) {
1273                return (ngx_thread_value_t) 1;
1274            }
1275    #endif
1276
1277            if (ngx_terminate) {
1278                thr->state = NGX_THREAD_EXIT;
1279
1280                ngx_log_debug1(NGX_LOG_DEBUG_CORE, cycle->log, 0,
1281                               "thread " NGX_TID_T_FMT " is done",
1282                               ngx_thread_self());
1283
1284                return (ngx_thread_value_t) 0;
1285            }
1286
1287            thr->state = NGX_THREAD_BUSY;
1288
1289    #if 0
1290            if (ngx_event_thread_process_posted(cycle) == NGX_ERROR) {
1291                return (ngx_thread_value_t) 1;
1292            }
1293
1294            if (ngx_event_thread_process_posted(cycle) == NGX_ERROR) {
1295                return (ngx_thread_value_t) 1;
1296            }
1297    #endif
1298
1299            if (ngx_process_changes) {
1300                if (ngx_process_changes(cycle, 1) == NGX_ERROR) {
1301                    return (ngx_thread_value_t) 1;
1302                }
1303            }
1304        }
1305    }
1306
1307    #endif
1308
1309
1310    static void
1311    ngx_cache_manager_process_cycle(ngx_cycle_t *cycle, void *data)
1312    {
1313        ngx_cache_manager_ctx_t *ctx = data;
1314
1315        void         *ident[4];
1316        ngx_event_t   ev;
1317
1318        /*
1319         * Set correct process type since closing listening Unix domain socket
1320         * in a master process also removes the Unix domain socket file.
1321         */
1322        ngx_process = NGX_PROCESS_HELPER;
1323
1324        ngx_close_listening_sockets(cycle);
1325
1326        /* Set a moderate number of connections for a helper process. */
1327        cycle->connection_n = 512;
1328
```

```
    ngx_worker_process_init(cycle, -1);

    ngx_memzero(&ev, sizeof(ngx_event_t));
    ev.handler = ctx->handler;
    ev.data = ident;
    ev.log = cycle->log;
    ident[3] = (void *) -1;

    ngx_use_accept_mutex = 0;

    ngx_setproctitle(ctx->name);

    ngx_add_timer(&ev, ctx->delay);

    for ( ;; ) {

        if (ngx_terminate || ngx_quit) {
            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "exiting");
            exit(0);
        }

        if (ngx_reopen) {
            ngx_reopen = 0;
            ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "reopening logs");
            ngx_reopen_files(cycle, -1);
        }

        ngx_process_events_and_timers(cycle);
    }
}


static void
ngx_cache_manager_process_handler(ngx_event_t *ev)
{
    time_t        next, n;
    ngx_uint_t    i;
    ngx_path_t  **path;

    next = 60 * 60;

    path = ngx_cycle->paths.elts;
    for (i = 0; i < ngx_cycle->paths.nelts; i++) {

        if (path[i]->manager) {
            n = path[i]->manager(path[i]->data);

            next = (n <= next) ? n : next;

            ngx_time_update();
        }
    }

    if (next == 0) {
        next = 1;
    }

    ngx_add_timer(ev, next * 1000);
}


static void
ngx_cache_loader_process_handler(ngx_event_t *ev)
{
    ngx_uint_t     i;
    ngx_path_t   **path;
    ngx_cycle_t   *cycle;

    cycle = (ngx_cycle_t *) ngx_cycle;

    path = cycle->paths.elts;
    for (i = 0; i < cycle->paths.nelts; i++) {

        if (ngx_terminate || ngx_quit) {
            break;
        }
```

```
1405
1406            if (path[i]->loader) {
1407                path[i]->loader(path[i]->data);
1408                ngx_time_update();
1409            }
1410        }
1411
1412        exit(0);
1413    }
```

One Level Up Top Level

src/os/unix/ngx_process.c - nginx-1.7.10

Global variables defined

- ngx_argc
- ngx_argv
- ngx_channel
- ngx_last_process
- ngx_os_argv
- ngx_process_slot
- ngx_processes
- signals

Data types defined

- ngx_signal_t

Functions defined

- ngx_debug_point
- ngx_execute
- ngx_execute_proc
- ngx_init_signals
- ngx_os_signal_process
- ngx_process_get_status
- ngx_signal_handler
- ngx_spawn_process
- ngx_unlock_mutexes

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_channel.h>
12
13
14  typedef struct {
15      int     signo;
16      char   *signame;
17      char   *name;
```

```
    void  (*handler)(int signo);
} ngx_signal_t;



static void ngx_execute_proc(ngx_cycle_t *cycle, void *data);
static void ngx_signal_handler(int signo);
static void ngx_process_get_status(void);
static void ngx_unlock_mutexes(ngx_pid_t pid);


int              ngx_argc;
char           **ngx_argv;
char           **ngx_os_argv;

ngx_int_t        ngx_process_slot;
ngx_socket_t     ngx_channel;
ngx_int_t        ngx_last_process;
ngx_process_t    ngx_processes[NGX_MAX_PROCESSES];


ngx_signal_t  signals[] = {
    { ngx_signal_value(NGX_RECONFIGURE_SIGNAL),
      "SIG" ngx_value(NGX_RECONFIGURE_SIGNAL),
      "reload",
      ngx_signal_handler },

    { ngx_signal_value(NGX_REOPEN_SIGNAL),
      "SIG" ngx_value(NGX_REOPEN_SIGNAL),
      "reopen",
      ngx_signal_handler },

    { ngx_signal_value(NGX_NOACCEPT_SIGNAL),
      "SIG" ngx_value(NGX_NOACCEPT_SIGNAL),
      "",
      ngx_signal_handler },

    { ngx_signal_value(NGX_TERMINATE_SIGNAL),
      "SIG" ngx_value(NGX_TERMINATE_SIGNAL),
      "stop",
      ngx_signal_handler },

    { ngx_signal_value(NGX_SHUTDOWN_SIGNAL),
      "SIG" ngx_value(NGX_SHUTDOWN_SIGNAL),
      "quit",
      ngx_signal_handler },

    { ngx_signal_value(NGX_CHANGEBIN_SIGNAL),
      "SIG" ngx_value(NGX_CHANGEBIN_SIGNAL),
      "",
      ngx_signal_handler },

    { SIGALRM, "SIGALRM", "", ngx_signal_handler },

    { SIGINT, "SIGINT", "", ngx_signal_handler },

    { SIGIO, "SIGIO", "", ngx_signal_handler },

    { SIGCHLD, "SIGCHLD", "", ngx_signal_handler },

    { SIGSYS, "SIGSYS, SIG_IGN", "", SIG_IGN },

    { SIGPIPE, "SIGPIPE, SIG_IGN", "", SIG_IGN },

    { 0, NULL, "", NULL }
};


ngx_pid_t
ngx_spawn_process(ngx_cycle_t *cycle, ngx_spawn_proc_pt proc, void *data,
    char *name, ngx_int_t respawn)
{
    u_long     on;
    ngx_pid_t  pid;
    ngx_int_t  s;

```

```
    if (respawn >= 0) {
        s = respawn;

    } else {
        for (s = 0; s < ngx_last_process; s++) {
            if (ngx_processes[s].pid == -1) {
                break;
            }
        }

        if (s == NGX_MAX_PROCESSES) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                          "no more than %d processes can be spawned",
                          NGX_MAX_PROCESSES);
            return NGX_INVALID_PID;
        }
    }


    if (respawn != NGX_PROCESS_DETACHED) {

        /* Solaris 9 still has no AF_LOCAL */

        if (socketpair(AF_UNIX, SOCK_STREAM, 0, ngx_processes[s].channel) == -1)
        {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "socketpair() failed while spawning \"%s\"", name);
            return NGX_INVALID_PID;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, cycle->log, 0,
                       "channel %d:%d",
                       ngx_processes[s].channel[0],
                       ngx_processes[s].channel[1]);

        if (ngx_nonblocking(ngx_processes[s].channel[0]) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          ngx_nonblocking_n " failed while spawning \"%s\"",
                          name);
            ngx_close_channel(ngx_processes[s].channel, cycle->log);
            return NGX_INVALID_PID;
        }

        if (ngx_nonblocking(ngx_processes[s].channel[1]) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          ngx_nonblocking_n " failed while spawning \"%s\"",
                          name);
            ngx_close_channel(ngx_processes[s].channel, cycle->log);
            return NGX_INVALID_PID;
        }

        on = 1;
        if (ioctl(ngx_processes[s].channel[0], FIOASYNC, &on) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "ioctl(FIOASYNC) failed while spawning \"%s\"", name);
            ngx_close_channel(ngx_processes[s].channel, cycle->log);
            return NGX_INVALID_PID;
        }

        if (fcntl(ngx_processes[s].channel[0], F_SETOWN, ngx_pid) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "fcntl(F_SETOWN) failed while spawning \"%s\"", name);
            ngx_close_channel(ngx_processes[s].channel, cycle->log);
            return NGX_INVALID_PID;
        }

        if (fcntl(ngx_processes[s].channel[0], F_SETFD, FD_CLOEXEC) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "fcntl(FD_CLOEXEC) failed while spawning \"%s\"",
                           name);
            ngx_close_channel(ngx_processes[s].channel, cycle->log);
            return NGX_INVALID_PID;
        }

        if (fcntl(ngx_processes[s].channel[1], F_SETFD, FD_CLOEXEC) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
```

```
                        "fcntl(FD_CLOEXEC) failed while spawning \"%s\"",
                        name);
            ngx_close_channel(ngx_processes[s].channel, cycle->log);
            return NGX_INVALID_PID;
        }

        ngx_channel = ngx_processes[s].channel[1];

    } else {
        ngx_processes[s].channel[0] = -1;
        ngx_processes[s].channel[1] = -1;
    }

    ngx_process_slot = s;


    pid = fork();

    switch (pid) {

    case -1:
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "fork() failed while spawning \"%s\"", name);
        ngx_close_channel(ngx_processes[s].channel, cycle->log);
        return NGX_INVALID_PID;

    case 0:
        ngx_pid = ngx_getpid();
        proc(cycle, data);
        break;

    default:
        break;
    }

    ngx_log_error(NGX_LOG_NOTICE, cycle->log, 0, "start %s %P", name, pid);

    ngx_processes[s].pid = pid;
    ngx_processes[s].exited = 0;

    if (respawn >= 0) {
        return pid;
    }

    ngx_processes[s].proc = proc;
    ngx_processes[s].data = data;
    ngx_processes[s].name = name;
    ngx_processes[s].exiting = 0;

    switch (respawn) {

    case NGX_PROCESS_NORESPAWN:
        ngx_processes[s].respawn = 0;
        ngx_processes[s].just_spawn = 0;
        ngx_processes[s].detached = 0;
        break;

    case NGX_PROCESS_JUST_SPAWN:
        ngx_processes[s].respawn = 0;
        ngx_processes[s].just_spawn = 1;
        ngx_processes[s].detached = 0;
        break;

    case NGX_PROCESS_RESPAWN:
        ngx_processes[s].respawn = 1;
        ngx_processes[s].just_spawn = 0;
        ngx_processes[s].detached = 0;
        break;

    case NGX_PROCESS_JUST_RESPAWN:
        ngx_processes[s].respawn = 1;
        ngx_processes[s].just_spawn = 1;
        ngx_processes[s].detached = 0;
        break;

    case NGX_PROCESS_DETACHED:
```

```
246         ngx_processes[s].respawn = 0;
247         ngx_processes[s].just_spawn = 0;
248         ngx_processes[s].detached = 1;
249         break;
250     }
251
252     if (s == ngx_last_process) {
253         ngx_last_process++;
254     }
255
256     return pid;
257 }
258
259
260 ngx_pid_t
261 ngx_execute(ngx_cycle_t *cycle, ngx_exec_ctx_t *ctx)
262 {
263     return ngx_spawn_process(cycle, ngx_execute_proc, ctx, ctx->name,
264                              NGX_PROCESS_DETACHED);
265 }
266
267
268 static void
269 ngx_execute_proc(ngx_cycle_t *cycle, void *data)
270 {
271     ngx_exec_ctx_t  *ctx = data;
272
273     if (execve(ctx->path, ctx->argv, ctx->envp) == -1) {
274         ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
275                       "execve() failed while executing %s \"%s\"",
276                       ctx->name, ctx->path);
277     }
278
279     exit(1);
280 }
281
282
283 ngx_int_t
284 ngx_init_signals(ngx_log_t *log)
285 {
286     ngx_signal_t      *sig;
287     struct sigaction   sa;
288
289     for (sig = signals; sig->signo != 0; sig++) {
290         ngx_memzero(&sa, sizeof(struct sigaction));
291         sa.sa_handler = sig->handler;
292         sigemptyset(&sa.sa_mask);
293         if (sigaction(sig->signo, &sa, NULL) == -1) {
294 #if (NGX_VALGRIND)
295             ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
296                           "sigaction(%s) failed, ignored", sig->signame);
297 #else
298             ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
299                           "sigaction(%s) failed", sig->signame);
300             return NGX_ERROR;
301 #endif
302         }
303     }
304
305     return NGX_OK;
306 }
307
308
309 void
310 ngx_signal_handler(int signo)
311 {
312     char            *action;
313     ngx_int_t        ignore;
314     ngx_err_t        err;
315     ngx_signal_t    *sig;
316
317     ignore = 0;
318
319     err = ngx_errno;
320
321     for (sig = signals; sig->signo != 0; sig++) {
```

```
        if (sig->signo == signo) {
            break;
        }
    }

    ngx_time_sigsafe_update();

    action = "";

    switch (ngx_process) {

    case NGX_PROCESS_MASTER:
    case NGX_PROCESS_SINGLE:
        switch (signo) {

        case ngx_signal_value(NGX_SHUTDOWN_SIGNAL):
            ngx_quit = 1;
            action = ", shutting down";
            break;

        case ngx_signal_value(NGX_TERMINATE_SIGNAL):
        case SIGINT:
            ngx_terminate = 1;
            action = ", exiting";
            break;

        case ngx_signal_value(NGX_NOACCEPT_SIGNAL):
            if (ngx_daemonized) {
                ngx_noaccept = 1;
                action = ", stop accepting connections";
            }
            break;

        case ngx_signal_value(NGX_RECONFIGURE_SIGNAL):
            ngx_reconfigure = 1;
            action = ", reconfiguring";
            break;

        case ngx_signal_value(NGX_REOPEN_SIGNAL):
            ngx_reopen = 1;
            action = ", reopening logs";
            break;

        case ngx_signal_value(NGX_CHANGEBIN_SIGNAL):
            if (getppid() > 1 || ngx_new_binary > 0) {

                /*
                 * Ignore the signal in the new binary if its parent is
                 * not the init process, i.e. the old binary's process
                 * is still running.  Or ignore the signal in the old binary's
                 * process if the new binary's process is already running.
                 */

                action = ", ignoring";
                ignore = 1;
                break;
            }

            ngx_change_binary = 1;
            action = ", changing binary";
            break;

        case SIGALRM:
            ngx_sigalrm = 1;
            break;

        case SIGIO:
            ngx_sigio = 1;
            break;

        case SIGCHLD:
            ngx_reap = 1;
            break;
        }

        break;
```

```
398
399     case NGX_PROCESS_WORKER:
400     case NGX_PROCESS_HELPER:
401         switch (signo) {
402
403         case ngx_signal_value(NGX_NOACCEPT_SIGNAL):
404             if (!ngx_daemonized) {
405                 break;
406             }
407             ngx_debug_quit = 1;
408         case ngx_signal_value(NGX_SHUTDOWN_SIGNAL):
409             ngx_quit = 1;
410             action = ", shutting down";
411             break;
412
413         case ngx_signal_value(NGX_TERMINATE_SIGNAL):
414         case SIGINT:
415             ngx_terminate = 1;
416             action = ", exiting";
417             break;
418
419         case ngx_signal_value(NGX_REOPEN_SIGNAL):
420             ngx_reopen = 1;
421             action = ", reopening logs";
422             break;
423
424         case ngx_signal_value(NGX_RECONFIGURE_SIGNAL):
425         case ngx_signal_value(NGX_CHANGEBIN_SIGNAL):
426         case SIGIO:
427             action = ", ignoring";
428             break;
429         }
430
431         break;
432     }
433
434     ngx_log_error(NGX_LOG_NOTICE, ngx_cycle->log, 0,
435                   "signal %d (%s) received%s", signo, sig->signame, action);
436
437     if (ignore) {
438         ngx_log_error(NGX_LOG_CRIT, ngx_cycle->log, 0,
439                       "the changing binary signal is ignored: "
440                       "you should shutdown or terminate "
441                       "before either old or new binary's process");
442     }
443
444     if (signo == SIGCHLD) {
445         ngx_process_get_status();
446     }
447
448     ngx_set_errno(err);
449 }
450
451
452 static void
453 ngx_process_get_status(void)
454 {
455     int              status;
456     char            *process;
457     ngx_pid_t        pid;
458     ngx_err_t        err;
459     ngx_int_t        i;
460     ngx_uint_t       one;
461
462     one = 0;
463
464     for ( ;; ) {
465         pid = waitpid(-1, &status, WNOHANG);
466
467         if (pid == 0) {
468             return;
469         }
470
471         if (pid == -1) {
472             err = ngx_errno;
473
```

```
            if (err == NGX_EINTR) {
                continue;
            }

            if (err == NGX_ECHILD && one) {
                return;
            }

            /*
             * Solaris always calls the signal handler for each exited process
             * despite waitpid() may be already called for this process.
             *
             * When several processes exit at the same time FreeBSD may
             * erroneously call the signal handler for exited process
             * despite waitpid() may be already called for this process.
             */

            if (err == NGX_ECHILD) {
                ngx_log_error(NGX_LOG_INFO, ngx_cycle->log, err,
                              "waitpid() failed");
                return;
            }

            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, err,
                          "waitpid() failed");
            return;
        }


        one = 1;
        process = "unknown process";

        for (i = 0; i < ngx_last_process; i++) {
            if (ngx_processes[i].pid == pid) {
                ngx_processes[i].status = status;
                ngx_processes[i].exited = 1;
                process = ngx_processes[i].name;
                break;
            }
        }

        if (WTERMSIG(status)) {
#ifdef WCOREDUMP
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                          "%s %P exited on signal %d%s",
                          process, pid, WTERMSIG(status),
                          WCOREDUMP(status) ? " (core dumped)" : "");
#else
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                          "%s %P exited on signal %d",
                          process, pid, WTERMSIG(status));
#endif

        } else {
            ngx_log_error(NGX_LOG_NOTICE, ngx_cycle->log, 0,
                          "%s %P exited with code %d",
                          process, pid, WEXITSTATUS(status));
        }

        if (WEXITSTATUS(status) == 2 && ngx_processes[i].respawn) {
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                          "%s %P exited with fatal code %d "
                          "and cannot be respawned",
                          process, pid, WEXITSTATUS(status));
            ngx_processes[i].respawn = 0;
        }

        ngx_unlock_mutexes(pid);
    }
}


static void
ngx_unlock_mutexes(ngx_pid_t pid)
{
    ngx_uint_t        i;
```

```
    ngx_shm_zone_t   *shm_zone;
    ngx_list_part_t  *part;
    ngx_slab_pool_t  *sp;

    /*
     * unlock the accept mutex if the abnormally exited process
     * held it
     */

    if (ngx_accept_mutex_ptr) {
        (void) ngx_shmtx_force_unlock(&ngx_accept_mutex, pid);
    }

    /*
     * unlock shared memory mutexes if held by the abnormally exited
     * process
     */

    part = (ngx_list_part_t *) &ngx_cycle->shared_memory.part;
    shm_zone = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }
            part = part->next;
            shm_zone = part->elts;
            i = 0;
        }

        sp = (ngx_slab_pool_t *) shm_zone[i].shm.addr;

        if (ngx_shmtx_force_unlock(&sp->mutex, pid)) {
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                          "shared memory zone \"%V\" was locked by %P",
                          &shm_zone[i].shm.name, pid);
        }
    }
}


void
ngx_debug_point(void)
{
    ngx_core_conf_t  *ccf;

    ccf = (ngx_core_conf_t *) ngx_get_conf(ngx_cycle->conf_ctx,
                                           ngx_core_module);

    switch (ccf->debug_points) {

    case NGX_DEBUG_POINTS_STOP:
        raise(SIGSTOP);
        break;

    case NGX_DEBUG_POINTS_ABORT:
        ngx_abort();
    }
}


ngx_int_t
ngx_os_signal_process(ngx_cycle_t *cycle, char *name, ngx_int_t pid)
{
    ngx_signal_t  *sig;

    for (sig = signals; sig->signo != 0; sig++) {
        if (ngx_strcmp(name, sig->name) == 0) {
            if (kill(pid, sig->signo) != -1) {
                return 0;
            }

            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "kill(%P, %d) failed", pid, sig->signo);
```

```
626         }
627     }
628 
629     return 1;
630 }
```

One Level Up Top Level

src/os/unix/ngx_process.h - nginx-1.7.10

Data types defined

- ngx_exec_ctx_t
- ngx_pid_t
- ngx_process_t
- ngx_spawn_proc_pt

Macros defined

- NGX_INVALID_PID
- NGX_MAX_PROCESSES
- NGX_PROCESS_DETACHED
- NGX_PROCESS_JUST_RESPAWN
- NGX_PROCESS_JUST_SPAWN
- NGX_PROCESS_NORESPAWN
- NGX_PROCESS_RESPAWN
- _NGX_PROCESS_H_INCLUDED_
- ngx_getpid
- ngx_log_pid
- ngx_sched_yield
- ngx_sched_yield

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_PROCESS_H_INCLUDED_
9  #define _NGX_PROCESS_H_INCLUDED_
10
11
12  #include <ngx_setaffinity.h>
13  #include <ngx_setproctitle.h>
14
15
16  typedef pid_t       ngx_pid_t;
17
18  #define NGX_INVALID_PID  -1
19
20  typedef void (*ngx_spawn_proc_pt) (ngx_cycle_t *cycle, void *data);
21
22  typedef struct {
23      ngx_pid_t           pid;
24      int                 status;
25      ngx_socket_t        channel[2];
```

```
26
27      ngx_spawn_proc_pt   proc;
28      void               *data;
29      char               *name;
30
31      unsigned            respawn:1;
32      unsigned            just_spawn:1;
33      unsigned            detached:1;
34      unsigned            exiting:1;
35      unsigned            exited:1;
36  } ngx_process_t;
37
38
39  typedef struct {
40      char         *path;
41      char         *name;
42      char *const  *argv;
43      char *const  *envp;
44  } ngx_exec_ctx_t;
45
46
47  #define NGX_MAX_PROCESSES         1024
48
49  #define NGX_PROCESS_NORESPAWN     -1
50  #define NGX_PROCESS_JUST_SPAWN    -2
51  #define NGX_PROCESS_RESPAWN       -3
52  #define NGX_PROCESS_JUST_RESPAWN  -4
53  #define NGX_PROCESS_DETACHED      -5
54
55
56  #define ngx_getpid   getpid
57
58  #ifndef ngx_log_pid
59  #define ngx_log_pid  ngx_pid
60  #endif
61
62
63  ngx_pid_t ngx_spawn_process(ngx_cycle_t *cycle,
64      ngx_spawn_proc_pt proc, void *data, char *name, ngx_int_t respawn);
65  ngx_pid_t ngx_execute(ngx_cycle_t *cycle, ngx_exec_ctx_t *ctx);
66  ngx_int_t ngx_init_signals(ngx_log_t *log);
67  void ngx_debug_point(void);
68
69
70  #if (NGX_HAVE_SCHED_YIELD)
71  #define ngx_sched_yield()  sched_yield()
72  #else
73  #define ngx_sched_yield()  usleep(1)
74  #endif
75
76
77  extern int            ngx_argc;
78  extern char         **ngx_argv;
79  extern char         **ngx_os_argv;
80
81  extern ngx_pid_t      ngx_pid;
82  extern ngx_socket_t   ngx_channel;
83  extern ngx_int_t      ngx_process_slot;
84  extern ngx_int_t      ngx_last_process;
85  extern ngx_process_t  ngx_processes[NGX_MAX_PROCESSES];
86
87
88  #endif /* _NGX_PROCESS_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_setaffinity.c - nginx-1.7.10

Functions defined

- ngx_setaffinity
- ngx_setaffinity

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  */
5
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9
10
11 #if (NGX_HAVE_CPUSET_SETAFFINITY)
12
13 #include <sys/cpuset.h>
14
15 void
16 ngx_setaffinity(uint64_t cpu_affinity, ngx_log_t *log)
17 {
18     cpuset_t    mask;
19     ngx_uint_t  i;
20
21     ngx_log_error(NGX_LOG_NOTICE, log, 0,
22                   "cpuset_setaffinity(0x%08Xl)", cpu_affinity);
23
24     CPU_ZERO(&mask);
25     i = 0;
26     do {
27         if (cpu_affinity & 1) {
28             CPU_SET(i, &mask);
29         }
30         i++;
31         cpu_affinity >>= 1;
32     } while (cpu_affinity);
33
34     if (cpuset_setaffinity(CPU_LEVEL_WHICH, CPU_WHICH_PID, -1,
35                            sizeof(cpuset_t), &mask) == -1)
36     {
37         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
38                       "cpuset_setaffinity() failed");
39     }
40 }
41
42 #elif (NGX_HAVE_SCHED_SETAFFINITY)
43
44 void
45 ngx_setaffinity(uint64_t cpu_affinity, ngx_log_t *log)
46 {
47     cpu_set_t   mask;
48     ngx_uint_t  i;
49
50     ngx_log_error(NGX_LOG_NOTICE, log, 0,
51                   "sched_setaffinity(0x%08Xl)", cpu_affinity);
52
53     CPU_ZERO(&mask);
54     i = 0;
55     do {
56         if (cpu_affinity & 1) {
57             CPU_SET(i, &mask);
58         }
59         i++;
60         cpu_affinity >>= 1;
```

```
61      } while (cpu_affinity);
62
63      if (sched_setaffinity(0, sizeof(cpu_set_t), &mask) == -1) {
64          ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
65                        "sched_setaffinity() failed");
66      }
67  }
68
69  #endif
```

One Level Up Top Level

src/os/unix/ngx_setproctitle.h - nginx-1.7.10

Macros defined

- NGX_SETPROCTITLE_PAD
- NGX_SETPROCTITLE_PAD
- NGX_SETPROCTITLE_USES_ENV
- NGX_SETPROCTITLE_USES_ENV
- _NGX_SETPROCTITLE_H_INCLUDED_
- ngx_init_setproctitle
- ngx_init_setproctitle
- ngx_setproctitle
- ngx_setproctitle

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_SETPROCTITLE_H_INCLUDED_
9  #define _NGX_SETPROCTITLE_H_INCLUDED_
10
11
12  #if (NGX_HAVE_SETPROCTITLE)
13
14  /* FreeBSD, NetBSD, OpenBSD */
15
16  #define ngx_init_setproctitle(log) NGX_OK
17  #define ngx_setproctitle(title)    setproctitle("%s", title)
18
19
20  #else /* !NGX_HAVE_SETPROCTITLE */
21
22  #if !defined NGX_SETPROCTITLE_USES_ENV
23
24  #if (NGX_SOLARIS)
25
26  #define NGX_SETPROCTITLE_USES_ENV  1
27  #define NGX_SETPROCTITLE_PAD       ' '
28
29  ngx_int_t ngx_init_setproctitle(ngx_log_t *log);
30  void ngx_setproctitle(char *title);
31
32  #elif (NGX_LINUX) || (NGX_DARWIN)
33
34  #define NGX_SETPROCTITLE_USES_ENV  1
35  #define NGX_SETPROCTITLE_PAD       '\0'
36
37  ngx_int_t ngx_init_setproctitle(ngx_log_t *log);
38  void ngx_setproctitle(char *title);
39
40  #else
41
42  #define ngx_init_setproctitle(log) NGX_OK
43  #define ngx_setproctitle(title)
44
```

```
45 #endif /* OSes */
46
47 #endif /* NGX_SETPROCTITLE_USES_ENV */
48
49 #endif /* NGX_HAVE_SETPROCTITLE */
50
51
52 #endif /* _NGX_SETPROCTITLE_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_channel.c - nginx-1.7.10

Functions defined

- ngx_add_channel_event
- ngx_close_channel
- ngx_read_channel
- ngx_write_channel

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_channel.h>
11
12
13  ngx_int_t
14  ngx_write_channel(ngx_socket_t s, ngx_channel_t *ch, size_t size,
15      ngx_log_t *log)
16  {
17      ssize_t             n;
18      ngx_err_t           err;
19      struct iovec        iov[1];
20      struct msghdr       msg;
21
22  #if (NGX_HAVE_MSGHDR_MSG_CONTROL)
23
24      union {
25          struct cmsghdr  cm;
26          char            space[CMSG_SPACE(sizeof(int))];
27      } cmsg;
28
29      if (ch->fd == -1) {
30          msg.msg_control = NULL;
31          msg.msg_controllen = 0;
32
33      } else {
34          msg.msg_control = (caddr_t) &cmsg;
35          msg.msg_controllen = sizeof(cmsg);
36
37          ngx_memzero(&cmsg, sizeof(cmsg));
38
39          cmsg.cm.cmsg_len = CMSG_LEN(sizeof(int));
40          cmsg.cm.cmsg_level = SOL_SOCKET;
41          cmsg.cm.cmsg_type = SCM_RIGHTS;
42
43          /*
44           * We have to use ngx_memcpy() instead of simple
45           *   *(int *) CMSG_DATA(&cmsg.cm) = ch->fd;
46           * because some gcc 4.4 with -O2/3/s optimization issues the warning:
47           *   dereferencing type-punned pointer will break strict-aliasing rules
48           *
49           * Fortunately, gcc with -O1 compiles this ngx_memcpy()
50           * in the same simple assignment as in the code above
51           */
52
53          ngx_memcpy(CMSG_DATA(&cmsg.cm), &ch->fd, sizeof(int));
54      }
55
56      msg.msg_flags = 0;
```

```
57
58  #else
59
60      if (ch->fd == -1) {
61          msg.msg_accrights = NULL;
62          msg.msg_accrightslen = 0;
63
64      } else {
65          msg.msg_accrights = (caddr_t) &ch->fd;
66          msg.msg_accrightslen = sizeof(int);
67      }
68
69  #endif
70
71      iov[0].iov_base = (char *) ch;
72      iov[0].iov_len = size;
73
74      msg.msg_name = NULL;
75      msg.msg_namelen = 0;
76      msg.msg_iov = iov;
77      msg.msg_iovlen = 1;
78
79      n = sendmsg(s, &msg, 0);
80
81      if (n == -1) {
82          err = ngx_errno;
83          if (err == NGX_EAGAIN) {
84              return NGX_AGAIN;
85          }
86
87          ngx_log_error(NGX_LOG_ALERT, log, err, "sendmsg() failed");
88          return NGX_ERROR;
89      }
90
91      return NGX_OK;
92  }
93
94
95  ngx_int_t
96  ngx_read_channel(ngx_socket_t s, ngx_channel_t *ch, size_t size, ngx_log_t *log)
97  {
98      ssize_t             n;
99      ngx_err_t           err;
100      struct iovec        iov[1];
101      struct msghdr       msg;
102
103  #if (NGX_HAVE_MSGHDR_MSG_CONTROL)
104      union {
105          struct cmsghdr  cm;
106          char            space[CMSG_SPACE(sizeof(int))];
107      } cmsg;
108  #else
109      int                 fd;
110  #endif
111
112      iov[0].iov_base = (char *) ch;
113      iov[0].iov_len = size;
114
115      msg.msg_name = NULL;
116      msg.msg_namelen = 0;
117      msg.msg_iov = iov;
118      msg.msg_iovlen = 1;
119
120  #if (NGX_HAVE_MSGHDR_MSG_CONTROL)
121      msg.msg_control = (caddr_t) &cmsg;
122      msg.msg_controllen = sizeof(cmsg);
123  #else
124      msg.msg_accrights = (caddr_t) &fd;
125      msg.msg_accrightslen = sizeof(int);
126  #endif
127
128      n = recvmsg(s, &msg, 0);
129
130      if (n == -1) {
131          err = ngx_errno;
132          if (err == NGX_EAGAIN) {
```

```
            return NGX_AGAIN;
        }

        ngx_log_error(NGX_LOG_ALERT, log, err, "recvmsg() failed");
        return NGX_ERROR;
    }

    if (n == 0) {
        ngx_log_debug0(NGX_LOG_DEBUG_CORE, log, 0, "recvmsg() returned zero");
        return NGX_ERROR;
    }

    if ((size_t) n < sizeof(ngx_channel_t)) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "recvmsg() returned not enough data: %z", n);
        return NGX_ERROR;
    }

#if (NGX_HAVE_MSGHDR_MSG_CONTROL)

    if (ch->command == NGX_CMD_OPEN_CHANNEL) {

        if (cmsg.cm.cmsg_len < (socklen_t) CMSG_LEN(sizeof(int))) {
            ngx_log_error(NGX_LOG_ALERT, log, 0,
                          "recvmsg() returned too small ancillary data");
            return NGX_ERROR;
        }

        if (cmsg.cm.cmsg_level != SOL_SOCKET || cmsg.cm.cmsg_type != SCM_RIGHTS)
        {
            ngx_log_error(NGX_LOG_ALERT, log, 0,
                          "recvmsg() returned invalid ancillary data "
                          "level %d or type %d",
                          cmsg.cm.cmsg_level, cmsg.cm.cmsg_type);
            return NGX_ERROR;
        }

        /* ch->fd = *(int *) CMSG_DATA(&cmsg.cm); */

        ngx_memcpy(&ch->fd, CMSG_DATA(&cmsg.cm), sizeof(int));
    }

    if (msg.msg_flags & (MSG_TRUNC|MSG_CTRUNC)) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "recvmsg() truncated data");
    }

#else

    if (ch->command == NGX_CMD_OPEN_CHANNEL) {
        if (msg.msg_accrightslen != sizeof(int)) {
            ngx_log_error(NGX_LOG_ALERT, log, 0,
                          "recvmsg() returned no ancillary data");
            return NGX_ERROR;
        }

        ch->fd = fd;
    }

#endif

    return n;
}


ngx_int_t
ngx_add_channel_event(ngx_cycle_t *cycle, ngx_fd_t fd, ngx_int_t event,
    ngx_event_handler_pt handler)
{
    ngx_event_t       *ev, *rev, *wev;
    ngx_connection_t  *c;

    c = ngx_get_connection(fd, cycle->log);

    if (c == NULL) {
        return NGX_ERROR;
```

```
209     }
210
211     c->pool = cycle->pool;
212
213     rev = c->read;
214     wev = c->write;
215
216     rev->log = cycle->log;
217     wev->log = cycle->log;
218
219     rev->channel = 1;
220     wev->channel = 1;
221
222     ev = (event == NGX_READ_EVENT) ? rev : wev;
223
224     ev->handler = handler;
225
226     if (ngx_add_conn && (ngx_event_flags & NGX_USE_EPOLL_EVENT) == 0) {
227         if (ngx_add_conn(c) == NGX_ERROR) {
228             ngx_free_connection(c);
229             return NGX_ERROR;
230         }
231
232     } else {
233         if (ngx_add_event(ev, event, 0) == NGX_ERROR) {
234             ngx_free_connection(c);
235             return NGX_ERROR;
236         }
237     }
238
239     return NGX_OK;
240 }
241
242
243 void
244 ngx_close_channel(ngx_fd_t *fd, ngx_log_t *log)
245 {
246     if (close(fd[0]) == -1) {
247         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno, "close() channel failed");
248     }
249
250     if (close(fd[1]) == -1) {
251         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno, "close() channel failed");
252     }
253 }
```

One Level Up Top Level

src/os/unix/ngx_channel.h - nginx-1.7.10

Data types defined

- ngx_channel_t

Macros defined

- _NGX_CHANNEL_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_CHANNEL_H_INCLUDED_
9  #define _NGX_CHANNEL_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15
16
17  typedef struct {
18      ngx_uint_t  command;
19      ngx_pid_t   pid;
20      ngx_int_t   slot;
21      ngx_fd_t    fd;
22  } ngx_channel_t;
23
24
25  ngx_int_t ngx_write_channel(ngx_socket_t s, ngx_channel_t *ch, size_t size,
26      ngx_log_t *log);
27  ngx_int_t ngx_read_channel(ngx_socket_t s, ngx_channel_t *ch, size_t size,
28      ngx_log_t *log);
29  ngx_int_t ngx_add_channel_event(ngx_cycle_t *cycle, ngx_fd_t fd,
30      ngx_int_t event, ngx_event_handler_pt handler);
31  void ngx_close_channel(ngx_fd_t *fd, ngx_log_t *log);
32
33
34  #endif /* _NGX_CHANNEL_H_INCLUDED_ */
```

src/os/unix/ngx_process_cycle.h - nginx-1.7.10

Data types defined

- ngx_cache_manager_ctx_t

Macros defined

- NGX_CMD_CLOSE_CHANNEL
- NGX_CMD_OPEN_CHANNEL
- NGX_CMD_QUIT
- NGX_CMD_REOPEN
- NGX_CMD_TERMINATE
- NGX_PROCESS_HELPER
- NGX_PROCESS_MASTER
- NGX_PROCESS_SIGNALLER
- NGX_PROCESS_SINGLE
- NGX_PROCESS_WORKER
- _NGX_PROCESS_CYCLE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_PROCESS_CYCLE_H_INCLUDED_
9  #define _NGX_PROCESS_CYCLE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #define NGX_CMD_OPEN_CHANNEL   1
17  #define NGX_CMD_CLOSE_CHANNEL  2
18  #define NGX_CMD_QUIT           3
19  #define NGX_CMD_TERMINATE      4
20  #define NGX_CMD_REOPEN         5
21
22
23  #define NGX_PROCESS_SINGLE     0
24  #define NGX_PROCESS_MASTER     1
25  #define NGX_PROCESS_SIGNALLER  2
26  #define NGX_PROCESS_WORKER     3
27  #define NGX_PROCESS_HELPER     4
28
29
30  typedef struct {
31      ngx_event_handler_pt       handler;
32      char                      *name;
33      ngx_msec_t                 delay;
34  } ngx_cache_manager_ctx_t;
```

```
35
36
37  void ngx_master_process_cycle(ngx_cycle_t *cycle);
38  void ngx_single_process_cycle(ngx_cycle_t *cycle);
39
40
41  extern ngx_uint_t      ngx_process;
42  extern ngx_pid_t       ngx_pid;
43  extern ngx_pid_t       ngx_new_binary;
44  extern ngx_uint_t      ngx_inherited;
45  extern ngx_uint_t      ngx_daemonized;
46  extern ngx_uint_t      ngx_threaded;
47  extern ngx_uint_t      ngx_exiting;
48
49  extern sig_atomic_t    ngx_reap;
50  extern sig_atomic_t    ngx_sigio;
51  extern sig_atomic_t    ngx_sigalrm;
52  extern sig_atomic_t    ngx_quit;
53  extern sig_atomic_t    ngx_debug_quit;
54  extern sig_atomic_t    ngx_terminate;
55  extern sig_atomic_t    ngx_noaccept;
56  extern sig_atomic_t    ngx_reconfigure;
57  extern sig_atomic_t    ngx_reopen;
58  extern sig_atomic_t    ngx_change_binary;
59
60
61  #endif /* _NGX_PROCESS_CYCLE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_file.h - nginx-1.7.10

Data types defined

- ngx_copy_file_t
- ngx_ext_rename_file_t
- ngx_file_s
- ngx_path_init_t
- ngx_path_loader_pt
- ngx_path_manager_pt
- ngx_path_t
- ngx_temp_file_t
- ngx_tree_ctx_s
- ngx_tree_ctx_t
- ngx_tree_handler_pt
- ngx_tree_init_handler_pt

Macros defined

- NGX_MAX_PATH_LEVEL
- _NGX_FILE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_FILE_H_INCLUDED_
9  #define _NGX_FILE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  struct ngx_file_s {
17      ngx_fd_t                   fd;
18      ngx_str_t                  name;
19      ngx_file_info_t            info;
20
21      off_t                      offset;
22      off_t                      sys_offset;
23
24      ngx_log_t                 *log;
25
26  #if (NGX_HAVE_FILE_AIO)
27      ngx_event_aio_t           *aio;
28  #endif
29
```

```
    unsigned                  valid_info:1;
    unsigned                  directio:1;
};


#define NGX_MAX_PATH_LEVEL  3


typedef time_t (*ngx_path_manager_pt) (void *data);
typedef void (*ngx_path_loader_pt) (void *data);


typedef struct {
    ngx_str_t                 name;
    size_t                    len;
    size_t                    level[3];

    ngx_path_manager_pt       manager;
    ngx_path_loader_pt        loader;
    void                     *data;

    u_char                   *conf_file;
    ngx_uint_t                line;
} ngx_path_t;


typedef struct {
    ngx_str_t                 name;
    size_t                    level[3];
} ngx_path_init_t;


typedef struct {
    ngx_file_t                file;
    off_t                     offset;
    ngx_path_t               *path;
    ngx_pool_t               *pool;
    char                     *warn;

    ngx_uint_t                access;

    unsigned                  log_level:8;
    unsigned                  persistent:1;
    unsigned                  clean:1;
} ngx_temp_file_t;


typedef struct {
    ngx_uint_t                access;
    ngx_uint_t                path_access;
    time_t                    time;
    ngx_fd_t                  fd;

    unsigned                  create_path:1;
    unsigned                  delete_file:1;

    ngx_log_t                *log;
} ngx_ext_rename_file_t;


typedef struct {
    off_t                     size;
    size_t                    buf_size;

    ngx_uint_t                access;
    time_t                    time;

    ngx_log_t                *log;
} ngx_copy_file_t;


 typedef struct ngx_tree_ctx_s  ngx_tree_ctx_t;

 typedef ngx_int_t (*ngx_tree_init_handler_pt) (void *ctx, void *prev);
 typedef ngx_int_t (*ngx_tree_handler_pt) (ngx_tree_ctx_t *ctx, ngx_str_t *name);

```

```
106 struct ngx_tree_ctx_s {
107     off_t                       size;
108     off_t                       fs_size;
109     ngx_uint_t                  access;
110     time_t                      mtime;
111
112     ngx_tree_init_handler_pt    init_handler;
113     ngx_tree_handler_pt         file_handler;
114     ngx_tree_handler_pt         pre_tree_handler;
115     ngx_tree_handler_pt         post_tree_handler;
116     ngx_tree_handler_pt         spec_handler;
117
118     void                       *data;
119     size_t                      alloc;
120
121     ngx_log_t                  *log;
122 };
123
124
125 ngx_int_t ngx_get_full_name(ngx_pool_t *pool, ngx_str_t *prefix,
126     ngx_str_t *name);
127
128 ssize_t ngx_write_chain_to_temp_file(ngx_temp_file_t *tf, ngx_chain_t *chain);
129 ngx_int_t ngx_create_temp_file(ngx_file_t *file, ngx_path_t *path,
130     ngx_pool_t *pool, ngx_uint_t persistent, ngx_uint_t clean,
131     ngx_uint_t access);
132 void ngx_create_hashed_filename(ngx_path_t *path, u_char *file, size_t len);
133 ngx_int_t ngx_create_path(ngx_file_t *file, ngx_path_t *path);
134 ngx_err_t ngx_create_full_path(u_char *dir, ngx_uint_t access);
135 ngx_int_t ngx_add_path(ngx_conf_t *cf, ngx_path_t **slot);
136 ngx_int_t ngx_create_paths(ngx_cycle_t *cycle, ngx_uid_t user);
137 ngx_int_t ngx_ext_rename_file(ngx_str_t *src, ngx_str_t *to,
138     ngx_ext_rename_file_t *ext);
139 ngx_int_t ngx_copy_file(u_char *from, u_char *to, ngx_copy_file_t *cf);
140 ngx_int_t ngx_walk_tree(ngx_tree_ctx_t *ctx, ngx_str_t *tree);
141
142 ngx_atomic_uint_t ngx_next_temp_number(ngx_uint_t collision);
143
144 char *ngx_conf_set_path_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
145 char *ngx_conf_merge_path_value(ngx_conf_t *cf, ngx_path_t **path,
146     ngx_path_t *prev, ngx_path_init_t *init);
147 char *ngx_conf_set_access_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
148
149
150 extern ngx_atomic_t      *ngx_temp_number;
151 extern ngx_atomic_int_t   ngx_random_number;
152
153
154 #endif /* _NGX_FILE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_file.c - nginx-1.7.10

Global variables defined

- ngx_random_number
- ngx_temp_number
- temp_number

Functions defined

- ngx_add_path
- ngx_conf_merge_path_value
- ngx_conf_set_access_slot
- ngx_conf_set_path_slot
- ngx_copy_file
- ngx_create_full_path
- ngx_create_hashed_filename
- ngx_create_path
- ngx_create_paths
- ngx_create_temp_file
- ngx_ext_rename_file
- ngx_get_full_name
- ngx_next_temp_number
- ngx_test_full_name
- ngx_walk_tree
- ngx_write_chain_to_temp_file

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  static ngx_int_t ngx_test_full_name(ngx_str_t *name);
13
14
15  static ngx_atomic_t   temp_number = 0;
16  ngx_atomic_t         *ngx_temp_number = &temp_number;
17  ngx_atomic_int_t      ngx_random_number = 123456;
18
```

```
ngx_int_t
ngx_get_full_name(ngx_pool_t *pool, ngx_str_t *prefix, ngx_str_t *name)
{
    size_t      len;
    u_char     *p, *n;
    ngx_int_t   rc;

    rc = ngx_test_full_name(name);

    if (rc == NGX_OK) {
        return rc;
    }

    len = prefix->len;

#if (NGX_WIN32)

    if (rc == 2) {
        len = rc;
    }

#endif

    n = ngx_pnalloc(pool, len + name->len + 1);
    if (n == NULL) {
        return NGX_ERROR;
    }

    p = ngx_cpymem(n, prefix->data, len);
    ngx_cpystrn(p, name->data, name->len + 1);

    name->len += len;
    name->data = n;

    return NGX_OK;
}


static ngx_int_t
ngx_test_full_name(ngx_str_t *name)
{
#if (NGX_WIN32)
    u_char  c0, c1;

    c0 = name->data[0];

    if (name->len < 2) {
        if (c0 == '/') {
            return 2;
        }

        return NGX_DECLINED;
    }

    c1 = name->data[1];

    if (c1 == ':') {
        c0 |= 0x20;

        if ((c0 >= 'a' && c0 <= 'z')) {
            return NGX_OK;
        }

        return NGX_DECLINED;
    }

    if (c1 == '/') {
        return NGX_OK;
    }

    if (c0 == '/') {
        return 2;
    }

    return NGX_DECLINED;
```

```
95
96  #else
97
98      if (name->data[0] == '/') {
99          return NGX_OK;
100     }
101
102     return NGX_DECLINED;
103
104 #endif
105 }
106
107
108 ssize_t
109 ngx_write_chain_to_temp_file(ngx_temp_file_t *tf, ngx_chain_t *chain)
110 {
111     ngx_int_t  rc;
112
113     if (tf->file.fd == NGX_INVALID_FILE) {
114         rc = ngx_create_temp_file(&tf->file, tf->path, tf->pool,
115                                   tf->persistent, tf->clean, tf->access);
116
117         if (rc != NGX_OK) {
118             return rc;
119         }
120
121         if (tf->log_level) {
122             ngx_log_error(tf->log_level, tf->file.log, 0, "%s %V",
123                           tf->warn, &tf->file.name);
124         }
125     }
126
127     return ngx_write_chain_to_file(&tf->file, chain, tf->offset, tf->pool);
128 }
129
130
131 ngx_int_t
132 ngx_create_temp_file(ngx_file_t *file, ngx_path_t *path, ngx_pool_t *pool,
133     ngx_uint_t persistent, ngx_uint_t clean, ngx_uint_t access)
134 {
135     uint32_t                  n;
136     ngx_err_t                 err;
137     ngx_pool_cleanup_t       *cln;
138     ngx_pool_cleanup_file_t  *clnf;
139
140     file->name.len = path->name.len + 1 + path->len + 10;
141
142     file->name.data = ngx_pnalloc(pool, file->name.len + 1);
143     if (file->name.data == NULL) {
144         return NGX_ERROR;
145     }
146
147 #if 0
148     for (i = 0; i < file->name.len; i++) {
149          file->name.data[i] = 'X';
150     }
151 #endif
152
153     ngx_memcpy(file->name.data, path->name.data, path->name.len);
154
155     n = (uint32_t) ngx_next_temp_number(0);
156
157     cln = ngx_pool_cleanup_add(pool, sizeof(ngx_pool_cleanup_file_t));
158     if (cln == NULL) {
159         return NGX_ERROR;
160     }
161
162     for ( ;; ) {
163         (void) ngx_sprintf(file->name.data + path->name.len + 1 + path->len,
164                            "%010uD%Z", n);
165
166         ngx_create_hashed_filename(path, file->name.data, file->name.len);
167
168         ngx_log_debug1(NGX_LOG_DEBUG_CORE, file->log, 0,
169                        "hashed path: %s", file->name.data);
170
```

```
        file->fd = ngx_open_tempfile(file->name.data, persistent, access);

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, file->log, 0,
                       "temp fd:%d", file->fd);

        if (file->fd != NGX_INVALID_FILE) {

            cln->handler = clean ? ngx_pool_delete_file : ngx_pool_cleanup_file;
            clnf = cln->data;

            clnf->fd = file->fd;
            clnf->name = file->name.data;
            clnf->log = pool->log;

            return NGX_OK;
        }

        err = ngx_errno;

        if (err == NGX_EEXIST) {
            n = (uint32_t) ngx_next_temp_number(1);
            continue;
        }

        if ((path->level[0] == 0) || (err != NGX_ENOPATH)) {
            ngx_log_error(NGX_LOG_CRIT, file->log, err,
                          ngx_open_tempfile_n " \"%s\" failed",
                          file->name.data);
            return NGX_ERROR;
        }

        if (ngx_create_path(file, path) == NGX_ERROR) {
            return NGX_ERROR;
        }
    }
}


void
ngx_create_hashed_filename(ngx_path_t *path, u_char *file, size_t len)
{
    size_t      i, level;
    ngx_uint_t  n;

    i = path->name.len + 1;

    file[path->name.len + path->len]  = '/';

    for (n = 0; n < 3; n++) {
        level = path->level[n];

        if (level == 0) {
            break;
        }

        len -= level;
        file[i - 1] = '/';
        ngx_memcpy(&file[i], &file[len], level);
        i += level + 1;
    }
}


ngx_int_t
ngx_create_path(ngx_file_t *file, ngx_path_t *path)
{
    size_t      pos;
    ngx_err_t   err;
    ngx_uint_t  i;

    pos = path->name.len;

    for (i = 0; i < 3; i++) {
        if (path->level[i] == 0) {
            break;
        }
```

```
        pos += path->level[i] + 1;

        file->name.data[pos] = '\0';

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, file->log, 0,
                       "temp file: \"%s\"", file->name.data);

        if (ngx_create_dir(file->name.data, 0700) == NGX_FILE_ERROR) {
            err = ngx_errno;
            if (err != NGX_EEXIST) {
                ngx_log_error(NGX_LOG_CRIT, file->log, err,
                              ngx_create_dir_n " \"%s\" failed",
                              file->name.data);
                return NGX_ERROR;
            }
        }

        file->name.data[pos] = '/';
    }

    return NGX_OK;
}


ngx_err_t
ngx_create_full_path(u_char *dir, ngx_uint_t access)
{
    u_char     *p, ch;
    ngx_err_t   err;

    err = 0;

#if (NGX_WIN32)
    p = dir + 3;
#else
    p = dir + 1;
#endif

    for ( /* void */ ; *p; p++) {
        ch = *p;

        if (ch != '/') {
            continue;
        }

        *p = '\0';

        if (ngx_create_dir(dir, access) == NGX_FILE_ERROR) {
            err = ngx_errno;

            switch (err) {
            case NGX_EEXIST:
                err = 0;
            case NGX_EACCES:
                break;

            default:
                return err;
            }
        }

        *p = '/';
    }

    return err;
}


ngx_atomic_uint_t
ngx_next_temp_number(ngx_uint_t collision)
{
    ngx_atomic_uint_t  n, add;

    add = collision ? ngx_random_number : 1;

```

```
    n = ngx_atomic_fetch_add(ngx_temp_number, add);

    return n + add;
}


char *
ngx_conf_set_path_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    ssize_t      level;
    ngx_str_t   *value;
    ngx_uint_t   i, n;
    ngx_path_t  *path, **slot;

    slot = (ngx_path_t **) (p + cmd->offset);

    if (*slot) {
        return "is duplicate";
    }

    path = ngx_pcalloc(cf->pool, sizeof(ngx_path_t));
    if (path == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    path->name = value[1];

    if (path->name.data[path->name.len - 1] == '/') {
        path->name.len--;
    }

    if (ngx_conf_full_name(cf->cycle, &path->name, 0) != NGX_OK) {
        return NULL;
    }

    path->conf_file = cf->conf_file->file.name.data;
    path->line = cf->conf_file->line;

    for (i = 0, n = 2; n < cf->args->nelts; i++, n++) {
        level = ngx_atoi(value[n].data, value[n].len);
        if (level == NGX_ERROR || level == 0) {
            return "invalid value";
        }

        path->level[i] = level;
        path->len += level + 1;
    }

    while (i < 3) {
        path->level[i++] = 0;
    }

    *slot = path;

    if (ngx_add_path(cf, slot) == NGX_ERROR) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_merge_path_value(ngx_conf_t *cf, ngx_path_t **path, ngx_path_t *prev,
    ngx_path_init_t *init)
{
    if (*path) {
        return NGX_CONF_OK;
    }

    if (prev) {
        *path = prev;
```

```
        return NGX_CONF_OK;
    }

    *path = ngx_pcalloc(cf->pool, sizeof(ngx_path_t));
    if (*path == NULL) {
        return NGX_CONF_ERROR;
    }

    (*path)->name = init->name;

    if (ngx_conf_full_name(cf->cycle, &(*path)->name, 0) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    (*path)->level[0] = init->level[0];
    (*path)->level[1] = init->level[1];
    (*path)->level[2] = init->level[2];

    (*path)->len = init->level[0] + (init->level[0] ? 1 : 0)
                  + init->level[1] + (init->level[1] ? 1 : 0)
                  + init->level[2] + (init->level[2] ? 1 : 0);

    if (ngx_add_path(cf, path) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


char *
ngx_conf_set_access_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *confp = conf;

    u_char      *p;
    ngx_str_t   *value;
    ngx_uint_t   i, right, shift, *access;

    access = (ngx_uint_t *) (confp + cmd->offset);

    if (*access != NGX_CONF_UNSET_UINT) {
        return "is duplicate";
    }

    value = cf->args->elts;

    *access = 0600;

    for (i = 1; i < cf->args->nelts; i++) {

        p = value[i].data;

        if (ngx_strncmp(p, "user:", sizeof("user:") - 1) == 0) {
            shift = 6;
            p += sizeof("user:") - 1;

        } else if (ngx_strncmp(p, "group:", sizeof("group:") - 1) == 0) {
            shift = 3;
            p += sizeof("group:") - 1;

        } else if (ngx_strncmp(p, "all:", sizeof("all:") - 1) == 0) {
            shift = 0;
            p += sizeof("all:") - 1;

        } else {
            goto invalid;
        }

        if (ngx_strcmp(p, "rw") == 0) {
            right = 6;

        } else if (ngx_strcmp(p, "r") == 0) {
            right = 4;

        } else {
```

```
            goto invalid;
        }

        *access |= right << shift;
    }

    return NGX_CONF_OK;

invalid:

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid value \"%V\"", &value[i]);

    return NGX_CONF_ERROR;
}


ngx_int_t
ngx_add_path(ngx_conf_t *cf, ngx_path_t **slot)
{
    ngx_uint_t   i, n;
    ngx_path_t  *path, **p;

    path = *slot;

    p = cf->cycle->paths.elts;
    for (i = 0; i < cf->cycle->paths.nelts; i++) {
        if (p[i]->name.len == path->name.len
            && ngx_strcmp(p[i]->name.data, path->name.data) == 0)
        {
            if (p[i]->data != path->data) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "the same path name \"%V\" "
                                   "used in %s:%ui and",
                                   &p[i]->name, p[i]->conf_file, p[i]->line);
                return NGX_ERROR;
            }

            for (n = 0; n < 3; n++) {
                if (p[i]->level[n] != path->level[n]) {
                    if (path->conf_file == NULL) {
                        if (p[i]->conf_file == NULL) {
                            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                                          "the default path name \"%V\" has "
                                          "the same name as another default path, "
                                          "but the different levels, you need to "
                                          "redefine one of them in http section",
                                          &p[i]->name);
                            return NGX_ERROR;
                        }

                        ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                                      "the path name \"%V\" in %s:%ui has "
                                      "the same name as default path, but "
                                      "the different levels, you need to "
                                      "define default path in http section",
                                      &p[i]->name, p[i]->conf_file, p[i]->line);
                        return NGX_ERROR;
                    }

                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "the same path name \"%V\" in %s:%ui "
                                       "has the different levels than",
                                       &p[i]->name, p[i]->conf_file, p[i]->line);
                    return NGX_ERROR;
                }

                if (p[i]->level[n] == 0) {
                    break;
                }
            }

            *slot = p[i];

            return NGX_OK;
        }
    }
```

```
551
552     p = ngx_array_push(&cf->cycle->paths);
553     if (p == NULL) {
554         return NGX_ERROR;
555     }
556
557     *p = path;
558
559     return NGX_OK;
560 }
561
562
563 ngx_int_t
564 ngx_create_paths(ngx_cycle_t *cycle, ngx_uid_t user)
565 {
566     ngx_err_t         err;
567     ngx_uint_t        i;
568     ngx_path_t      **path;
569
570     path = cycle->paths.elts;
571     for (i = 0; i < cycle->paths.nelts; i++) {
572
573         if (ngx_create_dir(path[i]->name.data, 0700) == NGX_FILE_ERROR) {
574             err = ngx_errno;
575             if (err != NGX_EEXIST) {
576                 ngx_log_error(NGX_LOG_EMERG, cycle->log, err,
577                               ngx_create_dir_n " \"%s\" failed",
578                               path[i]->name.data);
579                 return NGX_ERROR;
580             }
581         }
582
583         if (user == (ngx_uid_t) NGX_CONF_UNSET_UINT) {
584             continue;
585         }
586
587 #if !(NGX_WIN32)
588         {
589         ngx_file_info_t   fi;
590
591         if (ngx_file_info((const char *) path[i]->name.data, &fi)
592             == NGX_FILE_ERROR)
593         {
594             ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
595                           ngx_file_info_n " \"%s\" failed", path[i]->name.data);
596             return NGX_ERROR;
597         }
598
599         if (fi.st_uid != user) {
600             if (chown((const char *) path[i]->name.data, user, -1) == -1) {
601                 ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
602                               "chown(\"%s\", %d) failed",
603                               path[i]->name.data, user);
604                 return NGX_ERROR;
605             }
606         }
607
608         if ((fi.st_mode & (S_IRUSR|S_IWUSR|S_IXUSR))
609                                                 != (S_IRUSR|S_IWUSR|S_IXUSR))
610         {
611             fi.st_mode |= (S_IRUSR|S_IWUSR|S_IXUSR);
612
613             if (chmod((const char *) path[i]->name.data, fi.st_mode) == -1) {
614                 ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
615                               "chmod() \"%s\" failed", path[i]->name.data);
616                 return NGX_ERROR;
617             }
618         }
619         }
620 #endif
621     }
622
623     return NGX_OK;
624 }
625
626
```

```
ngx_int_t
ngx_ext_rename_file(ngx_str_t *src, ngx_str_t *to, ngx_ext_rename_file_t *ext)
{
    u_char           *name;
    ngx_err_t         err;
    ngx_copy_file_t   cf;

#if !(NGX_WIN32)

    if (ext->access) {
        if (ngx_change_file_access(src->data, ext->access) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_CRIT, ext->log, ngx_errno,
                          ngx_change_file_access_n " \"%s\" failed", src->data);
            err = 0;
            goto failed;
        }
    }

#endif

    if (ext->time != -1) {
        if (ngx_set_file_time(src->data, ext->fd, ext->time) != NGX_OK) {
            ngx_log_error(NGX_LOG_CRIT, ext->log, ngx_errno,
                          ngx_set_file_time_n " \"%s\" failed", src->data);
            err = 0;
            goto failed;
        }
    }

    if (ngx_rename_file(src->data, to->data) != NGX_FILE_ERROR) {
        return NGX_OK;
    }

    err = ngx_errno;

    if (err == NGX_ENOPATH) {

        if (!ext->create_path) {
            goto failed;
        }

        err = ngx_create_full_path(to->data, ngx_dir_access(ext->path_access));

        if (err) {
            ngx_log_error(NGX_LOG_CRIT, ext->log, err,
                          ngx_create_dir_n " \"%s\" failed", to->data);
            err = 0;
            goto failed;
        }

        if (ngx_rename_file(src->data, to->data) != NGX_FILE_ERROR) {
            return NGX_OK;
        }

        err = ngx_errno;
    }

#if (NGX_WIN32)

    if (err == NGX_EEXIST) {
        err = ngx_win32_rename_file(src, to, ext->log);

        if (err == 0) {
            return NGX_OK;
        }
    }

#endif

    if (err == NGX_EXDEV) {

        cf.size = -1;
        cf.buf_size = 0;
        cf.access = ext->access;
        cf.time = ext->time;
        cf.log = ext->log;
```

```
703
704         name = ngx_alloc(to->len + 1 + 10 + 1, ext->log);
705         if (name == NULL) {
706             return NGX_ERROR;
707         }
708
709         (void) ngx_sprintf(name, "%*s.%010uD%Z", to->len, to->data,
710                            (uint32_t) ngx_next_temp_number(0));
711
712         if (ngx_copy_file(src->data, name, &cf) == NGX_OK) {
713
714             if (ngx_rename_file(name, to->data) != NGX_FILE_ERROR) {
715                 ngx_free(name);
716
717                 if (ngx_delete_file(src->data) == NGX_FILE_ERROR) {
718                     ngx_log_error(NGX_LOG_CRIT, ext->log, ngx_errno,
719                                   ngx_delete_file_n " \"%s\" failed",
720                                   src->data);
721                     return NGX_ERROR;
722                 }
723
724                 return NGX_OK;
725             }
726
727             ngx_log_error(NGX_LOG_CRIT, ext->log, ngx_errno,
728                           ngx_rename_file_n " \"%s\" to \"%s\" failed",
729                           name, to->data);
730
731             if (ngx_delete_file(name) == NGX_FILE_ERROR) {
732                 ngx_log_error(NGX_LOG_CRIT, ext->log, ngx_errno,
733                               ngx_delete_file_n " \"%s\" failed", name);
734
735             }
736         }
737
738         ngx_free(name);
739
740         err = 0;
741     }
742
743 failed:
744
745     if (ext->delete_file) {
746         if (ngx_delete_file(src->data) == NGX_FILE_ERROR) {
747             ngx_log_error(NGX_LOG_CRIT, ext->log, ngx_errno,
748                           ngx_delete_file_n " \"%s\" failed", src->data);
749         }
750     }
751
752     if (err) {
753         ngx_log_error(NGX_LOG_CRIT, ext->log, err,
754                       ngx_rename_file_n " \"%s\" to \"%s\" failed",
755                       src->data, to->data);
756     }
757
758     return NGX_ERROR;
759 }
760
761
762 ngx_int_t
763 ngx_copy_file(u_char *from, u_char *to, ngx_copy_file_t *cf)
764 {
765     char             *buf;
766     off_t             size;
767     size_t            len;
768     ssize_t           n;
769     ngx_fd_t          fd, nfd;
770     ngx_int_t         rc;
771     ngx_file_info_t   fi;
772
773     rc = NGX_ERROR;
774     buf = NULL;
775     nfd = NGX_INVALID_FILE;
776
777     fd = ngx_open_file(from, NGX_FILE_RDONLY, NGX_FILE_OPEN, 0);
778
```

```
    if (fd == NGX_INVALID_FILE) {
        ngx_log_error(NGX_LOG_CRIT, cf->log, ngx_errno,
                      ngx_open_file_n " \"%s\" failed", from);
        goto failed;
    }

    if (cf->size != -1) {
        size = cf->size;

    } else {
        if (ngx_fd_info(fd, &fi) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                          ngx_fd_info_n " \"%s\" failed", from);

            goto failed;
        }

        size = ngx_file_size(&fi);
    }

    len = cf->buf_size ? cf->buf_size : 65536;

    if ((off_t) len > size) {
        len = (size_t) size;
    }

    buf = ngx_alloc(len, cf->log);
    if (buf == NULL) {
        goto failed;
    }

    nfd = ngx_open_file(to, NGX_FILE_WRONLY, NGX_FILE_CREATE_OR_OPEN,
                        cf->access);

    if (nfd == NGX_INVALID_FILE) {
        ngx_log_error(NGX_LOG_CRIT, cf->log, ngx_errno,
                      ngx_open_file_n " \"%s\" failed", to);
        goto failed;
    }

    while (size > 0) {

        if ((off_t) len > size) {
            len = (size_t) size;
        }

        n = ngx_read_fd(fd, buf, len);

        if (n == -1) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                          ngx_read_fd_n " \"%s\" failed", from);
            goto failed;
        }

        if ((size_t) n != len) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, 0,
                          ngx_read_fd_n " has read only %z of %uz from %s",
                          n, size, from);
            goto failed;
        }

        n = ngx_write_fd(nfd, buf, len);

        if (n == -1) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                          ngx_write_fd_n " \"%s\" failed", to);
            goto failed;
        }

        if ((size_t) n != len) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, 0,
                          ngx_write_fd_n " has written only %z of %uz to %s",
                          n, size, to);
            goto failed;
        }

```

```
        size -= n;
    }

    if (cf->time != -1) {
        if (ngx_set_file_time(to, nfd, cf->time) != NGX_OK) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                          ngx_set_file_time_n " \"%s\" failed", to);
            goto failed;
        }
    }

    rc = NGX_OK;

failed:

    if (nfd != NGX_INVALID_FILE) {
        if (ngx_close_file(nfd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                          ngx_close_file_n " \"%s\" failed", to);
        }
    }

    if (fd != NGX_INVALID_FILE) {
        if (ngx_close_file(fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
                          ngx_close_file_n " \"%s\" failed", from);
        }
    }

    if (buf) {
        ngx_free(buf);
    }

    return rc;
}


/*
* ctx->init_handler() - see ctx->alloc
* ctx->file_handler() - file handler
* ctx->pre_tree_handler() - handler is called before entering directory
* ctx->post_tree_handler() - handler is called after leaving directory
* ctx->spec_handler() - special (socket, FIFO, etc.) file handler
*
* ctx->data - some data structure, it may be the same on all levels, or
*     reallocated if ctx->alloc is nonzero
*
* ctx->alloc - a size of data structure that is allocated at every level
*     and is initialized by ctx->init_handler()
*
* ctx->log - a log
*
* on fatal (memory) error handler must return NGX_ABORT to stop walking tree
*/

ngx_int_t
ngx_walk_tree(ngx_tree_ctx_t *ctx, ngx_str_t *tree)
{
    void       *data, *prev;
    u_char     *p, *name;
    size_t      len;
    ngx_int_t   rc;
    ngx_err_t   err;
    ngx_str_t   file, buf;
    ngx_dir_t   dir;

    ngx_str_null(&buf);

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, ctx->log, 0,
                   "walk tree \"%V\"", tree);

    if (ngx_open_dir(tree, &dir) == NGX_ERROR) {
        ngx_log_error(NGX_LOG_CRIT, ctx->log, ngx_errno,
                      ngx_open_dir_n " \"%s\" failed", tree->data);
        return NGX_ERROR;
    }
```

```
931
932     prev = ctx->data;
933
934     if (ctx->alloc) {
935         data = ngx_alloc(ctx->alloc, ctx->log);
936         if (data == NULL) {
937             goto failed;
938         }
939
940         if (ctx->init_handler(data, prev) == NGX_ABORT) {
941             goto failed;
942         }
943
944         ctx->data = data;
945
946     } else {
947         data = NULL;
948     }
949
950     for ( ;; ) {
951
952         ngx_set_errno(0);
953
954         if (ngx_read_dir(&dir) == NGX_ERROR) {
955             err = ngx_errno;
956
957             if (err == NGX_ENOMOREFILES) {
958                 rc = NGX_OK;
959
960             } else {
961                 ngx_log_error(NGX_LOG_CRIT, ctx->log, err,
962                               ngx_read_dir_n " \"%s\" failed", tree->data);
963                 rc = NGX_ERROR;
964             }
965
966             goto done;
967         }
968
969         len = ngx_de_namelen(&dir);
970         name = ngx_de_name(&dir);
971
972         ngx_log_debug2(NGX_LOG_DEBUG_CORE, ctx->log, 0,
973                        "tree name %uz:\"%s\"", len, name);
974
975         if (len == 1 && name[0] == '.') {
976             continue;
977         }
978
979         if (len == 2 && name[0] == '.' && name[1] == '.') {
980             continue;
981         }
982
983         file.len = tree->len + 1 + len;
984
985         if (file.len + NGX_DIR_MASK_LEN > buf.len) {
986
987             if (buf.len) {
988                 ngx_free(buf.data);
989             }
990
991             buf.len = tree->len + 1 + len + NGX_DIR_MASK_LEN;
992
993             buf.data = ngx_alloc(buf.len + 1, ctx->log);
994             if (buf.data == NULL) {
995                 goto failed;
996             }
997         }
998
999         p = ngx_cpymem(buf.data, tree->data, tree->len);
1000         *p++ = '/';
1001         ngx_memcpy(p, name, len + 1);
1002
1003         file.data = buf.data;
1004
1005         ngx_log_debug1(NGX_LOG_DEBUG_CORE, ctx->log, 0,
1006                        "tree path \"%s\"", file.data);
```

```
1007
1008            if (!dir.valid_info) {
1009                if (ngx_de_info(file.data, &dir) == NGX_FILE_ERROR) {
1010                    ngx_log_error(NGX_LOG_CRIT, ctx->log, ngx_errno,
1011                                  ngx_de_info_n " \"%s\" failed", file.data);
1012                    continue;
1013                }
1014            }
1015
1016            if (ngx_de_is_file(&dir)) {
1017
1018                ngx_log_debug1(NGX_LOG_DEBUG_CORE, ctx->log, 0,
1019                               "tree file \"%s\"", file.data);
1020
1021                ctx->size = ngx_de_size(&dir);
1022                ctx->fs_size = ngx_de_fs_size(&dir);
1023                ctx->access = ngx_de_access(&dir);
1024                ctx->mtime = ngx_de_mtime(&dir);
1025
1026                if (ctx->file_handler(ctx, &file) == NGX_ABORT) {
1027                    goto failed;
1028                }
1029
1030            } else if (ngx_de_is_dir(&dir)) {
1031
1032                ngx_log_debug1(NGX_LOG_DEBUG_CORE, ctx->log, 0,
1033                               "tree enter dir \"%s\"", file.data);
1034
1035                ctx->access = ngx_de_access(&dir);
1036                ctx->mtime = ngx_de_mtime(&dir);
1037
1038                rc = ctx->pre_tree_handler(ctx, &file);
1039
1040                if (rc == NGX_ABORT) {
1041                    goto failed;
1042                }
1043
1044                if (rc == NGX_DECLINED) {
1045                    ngx_log_debug1(NGX_LOG_DEBUG_CORE, ctx->log, 0,
1046                                   "tree skip dir \"%s\"", file.data);
1047                    continue;
1048                }
1049
1050                if (ngx_walk_tree(ctx, &file) == NGX_ABORT) {
1051                    goto failed;
1052                }
1053
1054                ctx->access = ngx_de_access(&dir);
1055                ctx->mtime = ngx_de_mtime(&dir);
1056
1057                if (ctx->post_tree_handler(ctx, &file) == NGX_ABORT) {
1058                    goto failed;
1059                }
1060
1061            } else {
1062
1063                ngx_log_debug1(NGX_LOG_DEBUG_CORE, ctx->log, 0,
1064                               "tree special \"%s\"", file.data);
1065
1066                if (ctx->spec_handler(ctx, &file) == NGX_ABORT) {
1067                    goto failed;
1068                }
1069            }
1070        }
1071
1072    failed:
1073
1074        rc = NGX_ABORT;
1075
1076    done:
1077
1078        if (buf.len) {
1079            ngx_free(buf.data);
1080        }
1081
1082        if (data) {
```

```
1083            ngx_free(data);
1084            ctx->data = prev;
1085        }
1086
1087        if (ngx_close_dir(&dir) == NGX_ERROR) {
1088            ngx_log_error(NGX_LOG_CRIT, ctx->log, ngx_errno,
1089                          ngx_close_dir_n " \"%s\" failed", tree->data);
1090        }
1091
1092        return rc;
1093    }
```

One Level Up Top Level

src/event/ngx_event_timer.c - nginx-1.7.10

Global variables defined

- ngx_event_timer_rbtree
- ngx_event_timer_sentinel

Functions defined

- ngx_event_cancel_timers
- ngx_event_expire_timers
- ngx_event_find_timer
- ngx_event_timer_init

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ngx_rbtree_t              ngx_event_timer_rbtree;
14  static ngx_rbtree_node_t  ngx_event_timer_sentinel;
15
16  /*
17  * the event timer rbtree may contain the duplicate keys, however,
18  * it should not be a problem, because we use the rbtree to find
19  * a minimum timer value only
20  */
21
22  ngx_int_t
23  ngx_event_timer_init(ngx_log_t *log)
24  {
25      ngx_rbtree_init(&ngx_event_timer_rbtree, &ngx_event_timer_sentinel,
26                      ngx_rbtree_insert_timer_value);
27
28      return NGX_OK;
29  }
30
31
32  ngx_msec_t
33  ngx_event_find_timer(void)
34  {
35      ngx_msec_int_t      timer;
36      ngx_rbtree_node_t  *node, *root, *sentinel;
37
38      if (ngx_event_timer_rbtree.root == &ngx_event_timer_sentinel) {
39          return NGX_TIMER_INFINITE;
40      }
41
42      root = ngx_event_timer_rbtree.root;
43      sentinel = ngx_event_timer_rbtree.sentinel;
44
45      node = ngx_rbtree_min(root, sentinel);
46
47      timer = (ngx_msec_int_t) (node->key - ngx_current_msec);
48
```

```
    return (ngx_msec_t) (timer > 0 ? timer : 0);
}


void
ngx_event_expire_timers(void)
{
    ngx_event_t        *ev;
    ngx_rbtree_node_t  *node, *root, *sentinel;

    sentinel = ngx_event_timer_rbtree.sentinel;

    for ( ;; ) {
        root = ngx_event_timer_rbtree.root;

        if (root == sentinel) {
            return;
        }

        node = ngx_rbtree_min(root, sentinel);

        /* node->key > ngx_current_time */

        if ((ngx_msec_int_t) (node->key - ngx_current_msec) > 0) {
            return;
        }

        ev = (ngx_event_t *) ((char *) node - offsetof(ngx_event_t, timer));

        ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                       "event timer del: %d: %M",
                       ngx_event_ident(ev->data), ev->timer.key);

        ngx_rbtree_delete(&ngx_event_timer_rbtree, &ev->timer);

#if (NGX_DEBUG)
        ev->timer.left = NULL;
        ev->timer.right = NULL;
        ev->timer.parent = NULL;
#endif

        ev->timer_set = 0;

        ev->timedout = 1;

        ev->handler(ev);
    }
}


void
 ngx_event_cancel_timers(void)
 {
     ngx_event_t        *ev;
     ngx_rbtree_node_t  *node, *root, *sentinel;

     sentinel = ngx_event_timer_rbtree.sentinel;

     for ( ;; ) {
         root = ngx_event_timer_rbtree.root;

         if (root == sentinel) {
             return;
         }

         node = ngx_rbtree_min(root, sentinel);

         ev = (ngx_event_t *) ((char *) node - offsetof(ngx_event_t, timer));

         if (!ev->cancelable) {
             return;
         }

         ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                        "event timer cancel: %d: %M",
                        ngx_event_ident(ev->data), ev->timer.key);
```

```
125
126             ngx_rbtree_delete(&ngx_event_timer_rbtree, &ev->timer);
127
128     #if (NGX_DEBUG)
129             ev->timer.left = NULL;
130             ev->timer.right = NULL;
131             ev->timer.parent = NULL;
132     #endif
133
134             ev->timer_set = 0;
135
136             ev->handler(ev);
137         }
138     }
```

One Level Up Top Level

src/event/ngx_event_timer.h - nginx-1.7.10

Functions defined

- ngx_event_add_timer
- ngx_event_del_timer

Macros defined

- NGX_TIMER_INFINITE
- NGX_TIMER_LAZY_DELAY
- _NGX_EVENT_TIMER_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_EVENT_TIMER_H_INCLUDED_
9  #define _NGX_EVENT_TIMER_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15
16
17  #define NGX_TIMER_INFINITE  (ngx_msec_t) -1
18
19  #define NGX_TIMER_LAZY_DELAY  300
20
21
22  ngx_int_t ngx_event_timer_init(ngx_log_t *log);
23  ngx_msec_t ngx_event_find_timer(void);
24  void ngx_event_expire_timers(void);
25  void ngx_event_cancel_timers(void);
26
27
28  extern ngx_rbtree_t  ngx_event_timer_rbtree;
29
30
31  static ngx_inline void
32  ngx_event_del_timer(ngx_event_t *ev)
33  {
34      ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
35                     "event timer del: %d: %M",
36                      ngx_event_ident(ev->data), ev->timer.key);
37
38      ngx_rbtree_delete(&ngx_event_timer_rbtree, &ev->timer);
39
40  #if (NGX_DEBUG)
41      ev->timer.left = NULL;
42      ev->timer.right = NULL;
43      ev->timer.parent = NULL;
44  #endif
45
46      ev->timer_set = 0;
47  }
48
49
50  static ngx_inline void
```

```
ngx_event_add_timer(ngx_event_t *ev, ngx_msec_t timer)
{
    ngx_msec_t      key;
    ngx_msec_int_t  diff;

    key = ngx_current_msec + timer;

    if (ev->timer_set) {

        /*
         * Use a previous timer value if difference between it and a new
         * value is less than NGX_TIMER_LAZY_DELAY milliseconds: this allows
         * to minimize the rbtree operations for fast connections.
         */

        diff = (ngx_msec_int_t) (key - ev->timer.key);

        if (ngx_abs(diff) < NGX_TIMER_LAZY_DELAY) {
            ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                           "event timer: %d, old: %M, new: %M",
                            ngx_event_ident(ev->data), ev->timer.key, key);
            return;
        }

        ngx_del_timer(ev);
    }

    ev->timer.key = key;

    ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "event timer add: %d: %M:%M",
                    ngx_event_ident(ev->data), timer, ev->timer.key);

    ngx_rbtree_insert(&ngx_event_timer_rbtree, &ev->timer);

    ev->timer_set = 1;
}


#endif /* _NGX_EVENT_TIMER_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_times.h - nginx-1.7.10

Data types defined

- ngx_time_t

Macros defined

- _NGX_TIMES_H_INCLUDED_
- ngx_next_time_n
- ngx_time
- ngx_timeofday

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_TIMES_H_INCLUDED_
9  #define _NGX_TIMES_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  typedef struct {
17      time_t      sec;
18      ngx_uint_t  msec;
19      ngx_int_t   gmtoff;
20  } ngx_time_t;
21
22
23  void ngx_time_init(void);
24  void ngx_time_update(void);
25  void ngx_time_sigsafe_update(void);
26  u_char *ngx_http_time(u_char *buf, time_t t);
27  u_char *ngx_http_cookie_time(u_char *buf, time_t t);
28  void ngx_gmtime(time_t t, ngx_tm_t *tp);
29
30  time_t ngx_next_time(time_t when);
31  #define ngx_next_time_n      "mktime()"
32
33
34  extern volatile ngx_time_t  *ngx_cached_time;
35
36  #define ngx_time()           ngx_cached_time->sec
37  #define ngx_timeofday()      (ngx_time_t *) ngx_cached_time
38
39  extern volatile ngx_str_t    ngx_cached_err_log_time;
40  extern volatile ngx_str_t    ngx_cached_http_time;
41  extern volatile ngx_str_t    ngx_cached_http_log_time;
42  extern volatile ngx_str_t    ngx_cached_http_log_iso8601;
43  extern volatile ngx_str_t    ngx_cached_syslog_time;
44
45  /*
46  * milliseconds elapsed since epoch and truncated to ngx_msec_t,
47  * used in event timers
48  */
49  extern volatile ngx_msec_t  ngx_current_msec;
50
```

```
51
52  #endif /* _NGX_TIMES_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_slab.c - nginx-1.7.10

Global variables defined

- ngx_slab_exact_shift
- ngx_slab_exact_size
- ngx_slab_max_size

Functions defined

- ngx_slab_alloc
- ngx_slab_alloc_locked
- ngx_slab_alloc_pages
- ngx_slab_calloc
- ngx_slab_calloc_locked
- ngx_slab_error
- ngx_slab_free
- ngx_slab_free_locked
- ngx_slab_free_pages
- ngx_slab_init

Macros defined

- NGX_SLAB_BIG
- NGX_SLAB_BUSY
- NGX_SLAB_BUSY
- NGX_SLAB_EXACT
- NGX_SLAB_MAP_MASK
- NGX_SLAB_MAP_MASK
- NGX_SLAB_MAP_SHIFT
- NGX_SLAB_MAP_SHIFT
- NGX_SLAB_PAGE
- NGX_SLAB_PAGE_BUSY
- NGX_SLAB_PAGE_BUSY
- NGX_SLAB_PAGE_FREE
- NGX_SLAB_PAGE_FREE
- NGX_SLAB_PAGE_MASK

- NGX_SLAB_PAGE_START
- NGX_SLAB_PAGE_START
- NGX_SLAB_SHIFT_MASK
- NGX_SLAB_SHIFT_MASK
- NGX_SLAB_SMALL
- ngx_slab_junk
- ngx_slab_junk
- ngx_slab_junk

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9
10
11  #define NGX_SLAB_PAGE_MASK   3
12  #define NGX_SLAB_PAGE        0
13  #define NGX_SLAB_BIG         1
14  #define NGX_SLAB_EXACT       2
15  #define NGX_SLAB_SMALL       3
16
17  #if (NGX_PTR_SIZE == 4)
18
19  #define NGX_SLAB_PAGE_FREE   0
20  #define NGX_SLAB_PAGE_BUSY   0xffffffff
21  #define NGX_SLAB_PAGE_START  0x80000000
22
23  #define NGX_SLAB_SHIFT_MASK  0x0000000f
24  #define NGX_SLAB_MAP_MASK    0xffff0000
25  #define NGX_SLAB_MAP_SHIFT   16
26
27  #define NGX_SLAB_BUSY        0xffffffff
28
29  #else /* (NGX_PTR_SIZE == 8) */
30
31  #define NGX_SLAB_PAGE_FREE   0
32  #define NGX_SLAB_PAGE_BUSY   0xffffffffffffffff
33  #define NGX_SLAB_PAGE_START  0x8000000000000000
34
35  #define NGX_SLAB_SHIFT_MASK  0x000000000000000f
36  #define NGX_SLAB_MAP_MASK    0xffffffff00000000
37  #define NGX_SLAB_MAP_SHIFT   32
38
39  #define NGX_SLAB_BUSY        0xffffffffffffffff
40
41  #endif
42
43
44  #if (NGX_DEBUG_MALLOC)
45
46  #define ngx_slab_junk(p, size)     ngx_memset(p, 0xA5, size)
47
48  #elif (NGX_HAVE_DEBUG_MALLOC)
49
50  #define ngx_slab_junk(p, size)                                               \
51      if (ngx_debug_malloc)          ngx_memset(p, 0xA5, size)
52
53  #else
54
```

```c
#define ngx_slab_junk(p, size)

#endif

static ngx_slab_page_t *ngx_slab_alloc_pages(ngx_slab_pool_t *pool,
    ngx_uint_t pages);
static void ngx_slab_free_pages(ngx_slab_pool_t *pool, ngx_slab_page_t *page,
    ngx_uint_t pages);
static void ngx_slab_error(ngx_slab_pool_t *pool, ngx_uint_t level,
    char *text);


static ngx_uint_t  ngx_slab_max_size;
static ngx_uint_t  ngx_slab_exact_size;
static ngx_uint_t  ngx_slab_exact_shift;


void
ngx_slab_init(ngx_slab_pool_t *pool)
{
    u_char           *p;
    size_t            size;
    ngx_int_t         m;
    ngx_uint_t        i, n, pages;
    ngx_slab_page_t  *slots;

    /* STUB */
    if (ngx_slab_max_size == 0) {
        ngx_slab_max_size = ngx_pagesize / 2;
        ngx_slab_exact_size = ngx_pagesize / (8 * sizeof(uintptr_t));
        for (n = ngx_slab_exact_size; n >>= 1; ngx_slab_exact_shift++) {
            /* void */
        }
    }
    /**/

    pool->min_size = 1 << pool->min_shift;

    p = (u_char *) pool + sizeof(ngx_slab_pool_t);
    size = pool->end - p;

    ngx_slab_junk(p, size);

    slots = (ngx_slab_page_t *) p;
    n = ngx_pagesize_shift - pool->min_shift;

     for (i = 0; i < n; i++) {
         slots[i].slab = 0;
         slots[i].next = &slots[i];
         slots[i].prev = 0;
     }

     p += n * sizeof(ngx_slab_page_t);

     pages = (ngx_uint_t) (size / (ngx_pagesize + sizeof(ngx_slab_page_t)));

     ngx_memzero(p, pages * sizeof(ngx_slab_page_t));

     pool->pages = (ngx_slab_page_t *) p;

     pool->free.prev = 0;
     pool->free.next = (ngx_slab_page_t *) p;

     pool->pages->slab = pages;
     pool->pages->next = &pool->free;
     pool->pages->prev = (uintptr_t) &pool->free;

     pool->start = (u_char *)
                   ngx_align_ptr((uintptr_t) p + pages * sizeof(ngx_slab_page_t),
                                 ngx_pagesize);

     m = pages - (pool->end - pool->start) / ngx_pagesize;
     if (m > 0) {
         pages -= m;
         pool->pages->slab = pages;
     }
```

```
131
132     pool->last = pool->pages + pages;
133
134     pool->log_nomem = 1;
135     pool->log_ctx = &pool->zero;
136     pool->zero = '\0';
137 }
138
139
140 void *
141 ngx_slab_alloc(ngx_slab_pool_t *pool, size_t size)
142 {
143     void  *p;
144
145     ngx_shmtx_lock(&pool->mutex);
146
147     p = ngx_slab_alloc_locked(pool, size);
148
149     ngx_shmtx_unlock(&pool->mutex);
150
151     return p;
152 }
153
154
155 void *
156 ngx_slab_alloc_locked(ngx_slab_pool_t *pool, size_t size)
157 {
158     size_t            s;
159     uintptr_t         p, n, m, mask, *bitmap;
160     ngx_uint_t        i, slot, shift, map;
161     ngx_slab_page_t  *page, *prev, *slots;
162
163     if (size > ngx_slab_max_size) {
164
165         ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, ngx_cycle->log, 0,
166                        "slab alloc: %uz", size);
167
168         page = ngx_slab_alloc_pages(pool, (size >> ngx_pagesize_shift)
169                                           + ((size % ngx_pagesize) ? 1 : 0));
170         if (page) {
171             p = (page - pool->pages) << ngx_pagesize_shift;
172             p += (uintptr_t) pool->start;
173
174         } else {
175             p = 0;
176         }
177
178         goto done;
179     }
180
181     if (size > pool->min_size) {
182         shift = 1;
183         for (s = size - 1; s >>= 1; shift++) { /* void */ }
184         slot = shift - pool->min_shift;
185
186     } else {
187         size = pool->min_size;
188         shift = pool->min_shift;
189         slot = 0;
190     }
191
192     ngx_log_debug2(NGX_LOG_DEBUG_ALLOC, ngx_cycle->log, 0,
193                    "slab alloc: %uz slot: %ui", size, slot);
194
195     slots = (ngx_slab_page_t *) ((u_char *) pool + sizeof(ngx_slab_pool_t));
196     page = slots[slot].next;
197
198     if (page->next != page) {
199
200         if (shift < ngx_slab_exact_shift) {
201
202             do {
203                 p = (page - pool->pages) << ngx_pagesize_shift;
204                 bitmap = (uintptr_t *) (pool->start + p);
205
206                 map = (1 << (ngx_pagesize_shift - shift))
```

```
                        / (sizeof(uintptr_t) * 8);

            for (n = 0; n < map; n++) {

                if (bitmap[n] != NGX_SLAB_BUSY) {

                    for (m = 1, i = 0; m; m <<= 1, i++) {
                        if ((bitmap[n] & m)) {
                            continue;
                        }

                        bitmap[n] |= m;

                        i = ((n * sizeof(uintptr_t) * 8) << shift)
                            + (i << shift);

                        if (bitmap[n] == NGX_SLAB_BUSY) {
                            for (n = n + 1; n < map; n++) {
                                 if (bitmap[n] != NGX_SLAB_BUSY) {
                                     p = (uintptr_t) bitmap + i;

                                     goto done;
                                 }
                            }

                            prev = (ngx_slab_page_t *)
                                        (page->prev & ~NGX_SLAB_PAGE_MASK);
                            prev->next = page->next;
                            page->next->prev = page->prev;

                            page->next = NULL;
                            page->prev = NGX_SLAB_SMALL;
                        }

                        p = (uintptr_t) bitmap + i;

                        goto done;
                    }
                }
            }

            page = page->next;

        } while (page);

    } else if (shift == ngx_slab_exact_shift) {

        do {
            if (page->slab != NGX_SLAB_BUSY) {

                for (m = 1, i = 0; m; m <<= 1, i++) {
                    if ((page->slab & m)) {
                        continue;
                    }

                    page->slab |= m;

                    if (page->slab == NGX_SLAB_BUSY) {
                        prev = (ngx_slab_page_t *)
                                        (page->prev & ~NGX_SLAB_PAGE_MASK);
                        prev->next = page->next;
                        page->next->prev = page->prev;

                        page->next = NULL;
                        page->prev = NGX_SLAB_EXACT;
                    }

                    p = (page - pool->pages) << ngx_pagesize_shift;
                    p += i << shift;
                    p += (uintptr_t) pool->start;

                    goto done;
                }
            }

            page = page->next;
```

```
283
284             } while (page);
285
286         } else { /* shift > ngx_slab_exact_shift */
287
288             n = ngx_pagesize_shift - (page->slab & NGX_SLAB_SHIFT_MASK);
289             n = 1 << n;
290             n = ((uintptr_t) 1 << n) - 1;
291             mask = n << NGX_SLAB_MAP_SHIFT;
292
293             do {
294                 if ((page->slab & NGX_SLAB_MAP_MASK) != mask) {
295
296                     for (m = (uintptr_t) 1 << NGX_SLAB_MAP_SHIFT, i = 0;
297                          m & mask;
298                          m <<= 1, i++)
299                     {
300                         if ((page->slab & m)) {
301                             continue;
302                         }
303
304                         page->slab |= m;
305
306                         if ((page->slab & NGX_SLAB_MAP_MASK) == mask) {
307                             prev = (ngx_slab_page_t *)
308                                             (page->prev & ~NGX_SLAB_PAGE_MASK);
309                             prev->next = page->next;
310                             page->next->prev = page->prev;
311
312                             page->next = NULL;
313                             page->prev = NGX_SLAB_BIG;
314                         }
315
316                         p = (page - pool->pages) << ngx_pagesize_shift;
317                         p += i << shift;
318                         p += (uintptr_t) pool->start;
319
320                         goto done;
321                     }
322                 }
323
324                 page = page->next;
325
326             } while (page);
327         }
328     }
329
330     page = ngx_slab_alloc_pages(pool, 1);
331
332     if (page) {
333         if (shift < ngx_slab_exact_shift) {
334             p = (page - pool->pages) << ngx_pagesize_shift;
335             bitmap = (uintptr_t *) (pool->start + p);
336
337             s = 1 << shift;
338             n = (1 << (ngx_pagesize_shift - shift)) / 8 / s;
339
340             if (n == 0) {
341                 n = 1;
342             }
343
344             bitmap[0] = (2 << n) - 1;
345
346             map = (1 << (ngx_pagesize_shift - shift)) / (sizeof(uintptr_t) * 8);
347
348             for (i = 1; i < map; i++) {
349                 bitmap[i] = 0;
350             }
351
352             page->slab = shift;
353             page->next = &slots[slot];
354             page->prev = (uintptr_t) &slots[slot] | NGX_SLAB_SMALL;
355
356             slots[slot].next = page;
357
358             p = ((page - pool->pages) << ngx_pagesize_shift) + s * n;
```

```
            p += (uintptr_t) pool->start;

            goto done;

        } else if (shift == ngx_slab_exact_shift) {

            page->slab = 1;
            page->next = &slots[slot];
            page->prev = (uintptr_t) &slots[slot] | NGX_SLAB_EXACT;

            slots[slot].next = page;

            p = (page - pool->pages) << ngx_pagesize_shift;
            p += (uintptr_t) pool->start;

            goto done;

        } else { /* shift > ngx_slab_exact_shift */

            page->slab = ((uintptr_t) 1 << NGX_SLAB_MAP_SHIFT) | shift;
            page->next = &slots[slot];
            page->prev = (uintptr_t) &slots[slot] | NGX_SLAB_BIG;

            slots[slot].next = page;

            p = (page - pool->pages) << ngx_pagesize_shift;
            p += (uintptr_t) pool->start;

            goto done;
        }
    }

    p = 0;

done:

    ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, ngx_cycle->log, 0, "slab alloc: %p", p);

    return (void *) p;
}


void *
ngx_slab_calloc(ngx_slab_pool_t *pool, size_t size)
{
    void  *p;

    ngx_shmtx_lock(&pool->mutex);

    p = ngx_slab_calloc_locked(pool, size);

    ngx_shmtx_unlock(&pool->mutex);

    return p;
}


void *
ngx_slab_calloc_locked(ngx_slab_pool_t *pool, size_t size)
{
    void  *p;

    p = ngx_slab_alloc_locked(pool, size);
    if (p) {
        ngx_memzero(p, size);
    }

    return p;
}


void
ngx_slab_free(ngx_slab_pool_t *pool, void *p)
{
    ngx_shmtx_lock(&pool->mutex);

```

```
    ngx_slab_free_locked(pool, p);

    ngx_shmtx_unlock(&pool->mutex);
}


void
ngx_slab_free_locked(ngx_slab_pool_t *pool, void *p)
{
    size_t            size;
    uintptr_t         slab, m, *bitmap;
    ngx_uint_t        n, type, slot, shift, map;
    ngx_slab_page_t  *slots, *page;

    ngx_log_debug1(NGX_LOG_DEBUG_ALLOC, ngx_cycle->log, 0, "slab free: %p", p);

    if ((u_char *) p < pool->start || (u_char *) p > pool->end) {
        ngx_slab_error(pool, NGX_LOG_ALERT, "ngx_slab_free(): outside of pool");
        goto fail;
    }

    n = ((u_char *) p - pool->start) >> ngx_pagesize_shift;
    page = &pool->pages[n];
    slab = page->slab;
    type = page->prev & NGX_SLAB_PAGE_MASK;

    switch (type) {

    case NGX_SLAB_SMALL:

        shift = slab & NGX_SLAB_SHIFT_MASK;
        size = 1 << shift;

        if ((uintptr_t) p & (size - 1)) {
            goto wrong_chunk;
        }

        n = ((uintptr_t) p & (ngx_pagesize - 1)) >> shift;
        m = (uintptr_t) 1 << (n & (sizeof(uintptr_t) * 8 - 1));
        n /= (sizeof(uintptr_t) * 8);
        bitmap = (uintptr_t *)
                             ((uintptr_t) p & ~((uintptr_t) ngx_pagesize - 1));

        if (bitmap[n] & m) {

            if (page->next == NULL) {
                slots = (ngx_slab_page_t *)
                                    ((u_char *) pool + sizeof(ngx_slab_pool_t));
                slot = shift - pool->min_shift;

                page->next = slots[slot].next;
                slots[slot].next = page;

                page->prev = (uintptr_t) &slots[slot] | NGX_SLAB_SMALL;
                page->next->prev = (uintptr_t) page | NGX_SLAB_SMALL;
            }

            bitmap[n] &= ~m;

            n = (1 << (ngx_pagesize_shift - shift)) / 8 / (1 << shift);

            if (n == 0) {
                n = 1;
            }

            if (bitmap[0] & ~(((uintptr_t) 1 << n) - 1)) {
                goto done;
            }

            map = (1 << (ngx_pagesize_shift - shift)) / (sizeof(uintptr_t) * 8);

            for (n = 1; n < map; n++) {
                if (bitmap[n]) {
                    goto done;
                }
            }
```

```
            ngx_slab_free_pages(pool, page, 1);

            goto done;
        }

        goto chunk_already_free;

    case NGX_SLAB_EXACT:

        m = (uintptr_t) 1 <<
                (((uintptr_t) p & (ngx_pagesize - 1)) >> ngx_slab_exact_shift);
        size = ngx_slab_exact_size;

        if ((uintptr_t) p & (size - 1)) {
            goto wrong_chunk;
        }

        if (slab & m) {
            if (slab == NGX_SLAB_BUSY) {
                slots = (ngx_slab_page_t *)
                                    ((u_char *) pool + sizeof(ngx_slab_pool_t));
                slot = ngx_slab_exact_shift - pool->min_shift;

                page->next = slots[slot].next;
                slots[slot].next = page;

                page->prev = (uintptr_t) &slots[slot] | NGX_SLAB_EXACT;
                page->next->prev = (uintptr_t) page | NGX_SLAB_EXACT;
            }

            page->slab &= ~m;

            if (page->slab) {
                goto done;
            }

            ngx_slab_free_pages(pool, page, 1);

            goto done;
        }

        goto chunk_already_free;

    case NGX_SLAB_BIG:

        shift = slab & NGX_SLAB_SHIFT_MASK;
        size = 1 << shift;

        if ((uintptr_t) p & (size - 1)) {
            goto wrong_chunk;
        }

        m = (uintptr_t) 1 << ((((uintptr_t) p & (ngx_pagesize - 1)) >> shift)
                              + NGX_SLAB_MAP_SHIFT);

        if (slab & m) {

            if (page->next == NULL) {
                slots = (ngx_slab_page_t *)
                                    ((u_char *) pool + sizeof(ngx_slab_pool_t));
                slot = shift - pool->min_shift;

                page->next = slots[slot].next;
                slots[slot].next = page;

                page->prev = (uintptr_t) &slots[slot] | NGX_SLAB_BIG;
                page->next->prev = (uintptr_t) page | NGX_SLAB_BIG;
            }

            page->slab &= ~m;

            if (page->slab & NGX_SLAB_MAP_MASK) {
                goto done;
            }
```

```
            ngx_slab_free_pages(pool, page, 1);

            goto done;
        }

        goto chunk_already_free;

    case NGX_SLAB_PAGE:

        if ((uintptr_t) p & (ngx_pagesize - 1)) {
            goto wrong_chunk;
        }

        if (slab == NGX_SLAB_PAGE_FREE) {
            ngx_slab_error(pool, NGX_LOG_ALERT,
                           "ngx_slab_free(): page is already free");
            goto fail;
        }

        if (slab == NGX_SLAB_PAGE_BUSY) {
            ngx_slab_error(pool, NGX_LOG_ALERT,
                           "ngx_slab_free(): pointer to wrong page");
            goto fail;
        }

        n = ((u_char *) p - pool->start) >> ngx_pagesize_shift;
        size = slab & ~NGX_SLAB_PAGE_START;

        ngx_slab_free_pages(pool, &pool->pages[n], size);

        ngx_slab_junk(p, size << ngx_pagesize_shift);

        return;
    }

    /* not reached */

    return;

done:

    ngx_slab_junk(p, size);

    return;

wrong_chunk:

    ngx_slab_error(pool, NGX_LOG_ALERT,
                   "ngx_slab_free(): pointer to wrong chunk");

    goto fail;

chunk_already_free:

    ngx_slab_error(pool, NGX_LOG_ALERT,
                   "ngx_slab_free(): chunk is already free");

fail:

    return;
}


static ngx_slab_page_t *
ngx_slab_alloc_pages(ngx_slab_pool_t *pool, ngx_uint_t pages)
{
    ngx_slab_page_t  *page, *p;

    for (page = pool->free.next; page != &pool->free; page = page->next) {

        if (page->slab >= pages) {

            if (page->slab > pages) {
                page[page->slab - 1].prev = (uintptr_t) &page[pages];

                page[pages].slab = page->slab - pages;
```

```
663                 page[pages].next = page->next;
664                 page[pages].prev = page->prev;
665
666                 p = (ngx_slab_page_t *) page->prev;
667                 p->next = &page[pages];
668                 page->next->prev = (uintptr_t) &page[pages];
669
670             } else {
671                 p = (ngx_slab_page_t *) page->prev;
672                 p->next = page->next;
673                 page->next->prev = page->prev;
674             }
675
676             page->slab = pages | NGX_SLAB_PAGE_START;
677             page->next = NULL;
678             page->prev = NGX_SLAB_PAGE;
679
680             if (--pages == 0) {
681                 return page;
682             }
683
684             for (p = page + 1; pages; pages--) {
685                 p->slab = NGX_SLAB_PAGE_BUSY;
686                 p->next = NULL;
687                 p->prev = NGX_SLAB_PAGE;
688                 p++;
689             }
690
691             return page;
692         }
693     }
694
695     if (pool->log_nomem) {
696         ngx_slab_error(pool, NGX_LOG_CRIT,
697                        "ngx_slab_alloc() failed: no memory");
698     }
699
700     return NULL;
701 }
702
703
704 static void
705 ngx_slab_free_pages(ngx_slab_pool_t *pool, ngx_slab_page_t *page,
706     ngx_uint_t pages)
707 {
708     ngx_uint_t        type;
709     ngx_slab_page_t  *prev, *join;
710
711     page->slab = pages--;
712
713     if (pages) {
714         ngx_memzero(&page[1], pages * sizeof(ngx_slab_page_t));
715     }
716
717     if (page->next) {
718         prev = (ngx_slab_page_t *) (page->prev & ~NGX_SLAB_PAGE_MASK);
719         prev->next = page->next;
720         page->next->prev = page->prev;
721     }
722
723     join = page + page->slab;
724
725     if (join < pool->last) {
726         type = join->prev & NGX_SLAB_PAGE_MASK;
727
728         if (type == NGX_SLAB_PAGE) {
729
730             if (join->next != NULL) {
731                 pages += join->slab;
732                 page->slab += join->slab;
733
734                 prev = (ngx_slab_page_t *) (join->prev & ~NGX_SLAB_PAGE_MASK);
735                 prev->next = join->next;
736                 join->next->prev = join->prev;
737
738                 join->slab = NGX_SLAB_PAGE_FREE;
```

```
                join->next = NULL;
                join->prev = NGX_SLAB_PAGE;
            }
        }
    }

    if (page > pool->pages) {
        join = page - 1;
        type = join->prev & NGX_SLAB_PAGE_MASK;

        if (type == NGX_SLAB_PAGE) {

            if (join->slab == NGX_SLAB_PAGE_FREE) {
                join = (ngx_slab_page_t *) (join->prev & ~NGX_SLAB_PAGE_MASK);
            }

            if (join->next != NULL) {
                pages += join->slab;
                join->slab += page->slab;

                prev = (ngx_slab_page_t *) (join->prev & ~NGX_SLAB_PAGE_MASK);
                prev->next = join->next;
                join->next->prev = join->prev;

                page->slab = NGX_SLAB_PAGE_FREE;
                page->next = NULL;
                page->prev = NGX_SLAB_PAGE;

                page = join;
            }
        }
    }

    if (pages) {
        page[pages].prev = (uintptr_t) page;
    }

    page->prev = (uintptr_t) &pool->free;
    page->next = pool->free.next;

    page->next->prev = (uintptr_t) page;

    pool->free.next = page;
}


static void
ngx_slab_error(ngx_slab_pool_t *pool, ngx_uint_t level, char *text)
{
    ngx_log_error(level, ngx_cycle->log, 0, "%s%s", text, pool->log_ctx);
}
```

One Level Up Top Level

src/os/unix/ngx_darwin_config.h - nginx-1.7.10

Macros defined

- IOV_MAX
- NGX_HAVE_CASELESS_FILESYSTEM
- NGX_HAVE_DEBUG_MALLOC
- NGX_HAVE_INHERITED_NONBLOCK
- NGX_HAVE_OS_SPECIFIC_INIT
- NGX_LISTEN_BACKLOG
- _NGX_DARWIN_CONFIG_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_DARWIN_CONFIG_H_INCLUDED_
9  #define _NGX_DARWIN_CONFIG_H_INCLUDED_
10
11
12  #include <sys/types.h>
13  #include <sys/time.h>
14  #include <unistd.h>
15  #include <inttypes.h>
16  #include <stdarg.h>
17  #include <stddef.h>             /* offsetof() */
18  #include <stdio.h>
19  #include <stdlib.h>
20  #include <ctype.h>
21  #include <errno.h>
22  #include <string.h>
23  #include <signal.h>
24  #include <pwd.h>
25  #include <grp.h>
26  #include <dirent.h>
27  #include <glob.h>
28  #include <sys/mount.h>          /* statfs() */
29
30  #include <sys/filio.h>          /* FIONBIO */
31  #include <sys/ioctl.h>
32  #include <sys/uio.h>
33  #include <sys/stat.h>
34  #include <fcntl.h>
35
36  #include <sys/wait.h>
37  #include <sys/mman.h>
38  #include <sys/resource.h>
39  #include <sched.h>
40
41  #include <sys/socket.h>
42  #include <netinet/in.h>
43  #include <netinet/tcp.h>        /* TCP_NODELAY */
44  #include <arpa/inet.h>
45  #include <netdb.h>
46  #include <sys/un.h>
47
48  #include <sys/sysctl.h>
49  #include <xlocale.h>
```

```
50
51
52  #ifndef IOV_MAX
53  #define IOV_MAX   64
54  #endif
55
56
57  #include <ngx_auto_config.h>
58
59
60  #if (NGX_HAVE_POSIX_SEM)
61  #include <semaphore.h>
62  #endif
63
64
65  #if (NGX_HAVE_POLL)
66  #include <poll.h>
67  #endif
68
69
70  #if (NGX_HAVE_KQUEUE)
71  #include <sys/event.h>
72  #endif
73
74
75  #define NGX_LISTEN_BACKLOG  -1
76
77
78  #ifndef NGX_HAVE_INHERITED_NONBLOCK
79  #define NGX_HAVE_INHERITED_NONBLOCK  1
80  #endif
81
82
83  #ifndef NGX_HAVE_CASELESS_FILESYSTEM
84  #define NGX_HAVE_CASELESS_FILESYSTEM  1
85  #endif
86
87
88  #define NGX_HAVE_OS_SPECIFIC_INIT    1
89  #define NGX_HAVE_DEBUG_MALLOC        1
90
91
92  extern char **environ;
93
94
95  #endif /* _NGX_DARWIN_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/event/ngx_event_posted.h - nginx-1.7.10

Macros defined

- _NGX_EVENT_POSTED_H_INCLUDED_
- ngx_delete_posted_event
- ngx_post_event

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_EVENT_POSTED_H_INCLUDED_
9  #define _NGX_EVENT_POSTED_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15
16
17  #define ngx_post_event(ev, q)                                                \
18                                                                               \
19      if (!ev->posted) {                                                       \
20          ev->posted = 1;                                                      \
21          ngx_queue_insert_tail(q, &ev->queue);                                \
22                                                                               \
23          ngx_log_debug1(NGX_LOG_DEBUG_CORE, ev->log, 0, "post event %p", ev);  \
24                                                                               \
25      } else  {                                                                \
26          ngx_log_debug1(NGX_LOG_DEBUG_CORE, ev->log, 0,                       \
27                         "update posted event %p", ev);                        \
28      }
29
30
31  #define ngx_delete_posted_event(ev)                                          \
32                                                                               \
33      ev->posted = 0;                                                          \
34      ngx_queue_remove(&ev->queue);                                            \
35                                                                               \
36      ngx_log_debug1(NGX_LOG_DEBUG_CORE, ev->log, 0,                           \
37                     "delete posted event %p", ev);
38
39
40
41  void ngx_event_process_posted(ngx_cycle_t *cycle, ngx_queue_t *posted);
42
43
44  extern ngx_queue_t  ngx_posted_accept_events;
45  extern ngx_queue_t  ngx_posted_events;
46
47
48  #endif /* _NGX_EVENT_POSTED_H_INCLUDED_ */
```

src/event/ngx_event_posted.c - nginx-1.7.10

Global variables defined

- ngx_posted_accept_events
- ngx_posted_events

Functions defined

- ngx_event_process_posted

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ngx_queue_t  ngx_posted_accept_events;
14  ngx_queue_t  ngx_posted_events;
15
16
17  void
18  ngx_event_process_posted(ngx_cycle_t *cycle, ngx_queue_t *posted)
19  {
20      ngx_queue_t  *q;
21      ngx_event_t  *ev;
22
23      while (!ngx_queue_empty(posted)) {
24
25          q = ngx_queue_head(posted);
26          ev = ngx_queue_data(q, ngx_event_t, queue);
27
28          ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
29                         "posted event %p", ev);
30
31          ngx_delete_posted_event(ev);
32
33          ev->handler(ev);
34      }
35  }
```

src/core/ngx_crc32.h - nginx-1.7.10

Functions defined

- ngx_crc32_long
- ngx_crc32_short
- ngx_crc32_update

Macros defined

- _NGX_CRC32_H_INCLUDED_
- ngx_crc32_final
- ngx_crc32_init

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_CRC32_H_INCLUDED_
9  #define _NGX_CRC32_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  extern uint32_t  *ngx_crc32_table_short;
17  extern uint32_t   ngx_crc32_table256[];
18
19
20  static ngx_inline uint32_t
21  ngx_crc32_short(u_char *p, size_t len)
22  {
23      u_char    c;
24      uint32_t  crc;
25
26      crc = 0xffffffff;
27
28      while (len--) {
29          c = *p++;
30          crc = ngx_crc32_table_short[(crc ^ (c & 0xf)) & 0xf] ^ (crc >> 4);
31          crc = ngx_crc32_table_short[(crc ^ (c >> 4)) & 0xf] ^ (crc >> 4);
32      }
33
34      return crc ^ 0xffffffff;
35  }
36
37
38  static ngx_inline uint32_t
39  ngx_crc32_long(u_char *p, size_t len)
40  {
41      uint32_t  crc;
42
43      crc = 0xffffffff;
44
45      while (len--) {
46          crc = ngx_crc32_table256[(crc ^ *p++) & 0xff] ^ (crc >> 8);
47      }
48
```

```
    return crc ^ 0xffffffff;
}


#define ngx_crc32_init(crc)                                                  \
    crc = 0xffffffff


static ngx_inline void
ngx_crc32_update(uint32_t *crc, u_char *p, size_t len)
{
    uint32_t  c;

    c = *crc;

    while (len--) {
        c = ngx_crc32_table256[(c ^ *p++) & 0xff] ^ (c >> 8);
    }

    *crc = c;
}


#define ngx_crc32_final(crc)                                                 \
    crc ^= 0xffffffff


ngx_int_t ngx_crc32_table_init(void);


#endif /* _NGX_CRC32_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_crc32.c - nginx-1.7.10

Global variables defined

- ngx_crc32_table16
- ngx_crc32_table256
- ngx_crc32_table_short

Functions defined

- ngx_crc32_table_init

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13   * The code and lookup tables are based on the algorithm
14   * described at http://www.w3.org/TR/PNG/
15   *
16   * The 256 element lookup table takes 1024 bytes, and it may be completely
17   * cached after processing about 30-60 bytes of data.  So for short data
18   * we use the 16 element lookup table that takes only 64 bytes and align it
19   * to CPU cache line size.  Of course, the small table adds code inside
20   * CRC32 loop, but the cache misses overhead is bigger than overhead of
21   * the additional code.  For example, ngx_crc32_short() of 16 bytes of data
22   * takes half as much CPU clocks than ngx_crc32_long().
23   */
24
25
26  static uint32_t  ngx_crc32_table16[] = {
27      0x00000000, 0x1db71064, 0x3b6e20c8, 0x26d930ac,
28      0x76dc4190, 0x6b6b51f4, 0x4db26158, 0x5005713c,
29      0xedb88320, 0xf00f9344, 0xd6d6a3e8, 0xcb61b38c,
30      0x9b64c2b0, 0x86d3d2d4, 0xa00ae278, 0xbdbdf21c
31  };
32
33
34  uint32_t  ngx_crc32_table256[] = {
35      0x00000000, 0x77073096, 0xee0e612c, 0x990951ba,
36      0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495a3,
37      0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
38      0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91,
39      0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de,
40      0x1adad47d, 0x6dde4eb, 0xf4d4b551, 0x83d385c7,
41      0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec,
42      0x14015c4f, 0x63066cd9, 0xfa0f3d63, 0x8d080df5,
43      0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
44      0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
45      0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940,
46      0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
47      0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116,
48      0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,
49      0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
50      0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d,
51      0x76dc4190, 0x01db7106, 0x98d220bc, 0xefd5102a,
52      0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
```

```
    0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818,
    0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
    0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
    0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457,
    0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea, 0xfcb9887c,
    0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
    0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2,
    0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb,
    0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
    0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9,
    0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086,
    0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4,
    0x59b33d17, 0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad,
    0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
    0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683,
    0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,
    0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
    0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe,
    0xf762575d, 0x806567cb, 0x196c3671, 0x6e6b06e7,
    0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
    0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
    0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252,
    0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
    0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60,
    0xdf60efc3, 0xa867df55, 0x316e8eef, 0x4669be79,
    0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
    0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f,
    0xc5ba3bbe, 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04,
    0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
    0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a,
    0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
    0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
    0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21,
    0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8, 0x1fda836e,
    0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
    0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c,
    0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45,
    0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
    0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db,
    0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0,
    0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
    0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6,
    0xbad03605, 0xcdd70693, 0x54de5729, 0x23d967bf,
    0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
    0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};


uint32_t *ngx_crc32_table_short = ngx_crc32_table16;


ngx_int_t
ngx_crc32_table_init(void)
{
    void  *p;

    if (((uintptr_t) ngx_crc32_table_short
          & ~((uintptr_t) ngx_cacheline_size - 1))
        == (uintptr_t) ngx_crc32_table_short)
    {
        return NGX_OK;
    }

    p = ngx_alloc(16 * sizeof(uint32_t) + ngx_cacheline_size, ngx_cycle->log);
    if (p == NULL) {
        return NGX_ERROR;
    }

    p = ngx_align_ptr(p, ngx_cacheline_size);

    ngx_memcpy(p, ngx_crc32_table16, 16 * sizeof(uint32_t));

    ngx_crc32_table_short = p;

    return NGX_OK;
```

```
129 }
```

One Level Up Top Level

src/event/ngx_event_openssl_stapling.c - nginx-1.7.10

Data types defined

- ngx_ssl_ocsp_ctx_s
- ngx_ssl_ocsp_ctx_t
- ngx_ssl_stapling_t

Functions defined

- ngx_ssl_certificate_status_callback
- ngx_ssl_ocsp_connect
- ngx_ssl_ocsp_create_request
- ngx_ssl_ocsp_done
- ngx_ssl_ocsp_dummy_handler
- ngx_ssl_ocsp_error
- ngx_ssl_ocsp_log_error
- ngx_ssl_ocsp_parse_header_line
- ngx_ssl_ocsp_parse_status_line
- ngx_ssl_ocsp_process_body
- ngx_ssl_ocsp_process_headers
- ngx_ssl_ocsp_process_status_line
- ngx_ssl_ocsp_read_handler
- ngx_ssl_ocsp_request
- ngx_ssl_ocsp_resolve_handler
- ngx_ssl_ocsp_start
- ngx_ssl_ocsp_write_handler
- ngx_ssl_stapling
- ngx_ssl_stapling
- ngx_ssl_stapling_cleanup
- ngx_ssl_stapling_file
- ngx_ssl_stapling_issuer
- ngx_ssl_stapling_ocsp_handler
- ngx_ssl_stapling_resolver
- ngx_ssl_stapling_resolver

- ngx_ssl_stapling_responder
- ngx_ssl_stapling_update

Source code

```
1
2  /*
3   * Copyright (C) Maxim Dounin
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_event_connect.h>
12
13
14  #if (!defined OPENSSL_NO_OCSP && defined SSL_CTRL_SET_TLSEXT_STATUS_REQ_CB)
15
16
17  typedef struct {
18      ngx_str_t                    staple;
19      ngx_msec_t                   timeout;
20
21      ngx_resolver_t              *resolver;
22      ngx_msec_t                   resolver_timeout;
23
24      ngx_addr_t                  *addrs;
25      ngx_str_t                    host;
26      ngx_str_t                    uri;
27      in_port_t                    port;
28
29      SSL_CTX                     *ssl_ctx;
30
31      X509                        *cert;
32      X509                        *issuer;
33
34      time_t                       valid;
35
36      unsigned                     verify:1;
37      unsigned                     loading:1;
38  } ngx_ssl_stapling_t;
39
40
41  typedef struct ngx_ssl_ocsp_ctx_s  ngx_ssl_ocsp_ctx_t;
42
43  struct ngx_ssl_ocsp_ctx_s {
44      X509                        *cert;
45      X509                        *issuer;
46
47      ngx_uint_t                   naddrs;
48
49      ngx_addr_t                  *addrs;
50      ngx_str_t                    host;
51      ngx_str_t                    uri;
52      in_port_t                    port;
53
54      ngx_resolver_t              *resolver;
55      ngx_msec_t                   resolver_timeout;
56
57      ngx_msec_t                   timeout;
58
59      void                        (*handler)(ngx_ssl_ocsp_ctx_t *r);
60      void                         *data;
61
62      ngx_buf_t                    *request;
63      ngx_buf_t                    *response;
64      ngx_peer_connection_t        peer;
65
66      ngx_int_t                   (*process)(ngx_ssl_ocsp_ctx_t *r);
67
68      ngx_uint_t                   state;
```

```
    ngx_uint_t                  code;
    ngx_uint_t                  count;

    ngx_uint_t                  done;

    u_char                     *header_name_start;
    u_char                     *header_name_end;
    u_char                     *header_start;
    u_char                     *header_end;

    ngx_pool_t                 *pool;
    ngx_log_t                  *log;
};


static ngx_int_t ngx_ssl_stapling_file(ngx_conf_t *cf, ngx_ssl_t *ssl,
    ngx_str_t *file);
static ngx_int_t ngx_ssl_stapling_issuer(ngx_conf_t *cf, ngx_ssl_t *ssl);
static ngx_int_t ngx_ssl_stapling_responder(ngx_conf_t *cf, ngx_ssl_t *ssl,
    ngx_str_t *responder);

static int ngx_ssl_certificate_status_callback(ngx_ssl_conn_t *ssl_conn,
    void *data);
static void ngx_ssl_stapling_update(ngx_ssl_stapling_t *staple);
static void ngx_ssl_stapling_ocsp_handler(ngx_ssl_ocsp_ctx_t *ctx);

static void ngx_ssl_stapling_cleanup(void *data);

static ngx_ssl_ocsp_ctx_t *ngx_ssl_ocsp_start(void);
static void ngx_ssl_ocsp_done(ngx_ssl_ocsp_ctx_t *ctx);
 static void ngx_ssl_ocsp_request(ngx_ssl_ocsp_ctx_t *ctx);
 static void ngx_ssl_ocsp_resolve_handler(ngx_resolver_ctx_t *resolve);
 static void ngx_ssl_ocsp_connect(ngx_ssl_ocsp_ctx_t *ctx);
 static void ngx_ssl_ocsp_write_handler(ngx_event_t *wev);
 static void ngx_ssl_ocsp_read_handler(ngx_event_t *rev);
 static void ngx_ssl_ocsp_dummy_handler(ngx_event_t *ev);

 static ngx_int_t ngx_ssl_ocsp_create_request(ngx_ssl_ocsp_ctx_t *ctx);
 static ngx_int_t ngx_ssl_ocsp_process_status_line(ngx_ssl_ocsp_ctx_t *ctx);
 static ngx_int_t ngx_ssl_ocsp_parse_status_line(ngx_ssl_ocsp_ctx_t *ctx);
 static ngx_int_t ngx_ssl_ocsp_process_headers(ngx_ssl_ocsp_ctx_t *ctx);
 static ngx_int_t ngx_ssl_ocsp_parse_header_line(ngx_ssl_ocsp_ctx_t *ctx);
 static ngx_int_t ngx_ssl_ocsp_process_body(ngx_ssl_ocsp_ctx_t *ctx);

 static u_char *ngx_ssl_ocsp_log_error(ngx_log_t *log, u_char *buf, size_t len);


 ngx_int_t
 ngx_ssl_stapling(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *file,
     ngx_str_t *responder, ngx_uint_t verify)
 {
     ngx_int_t                   rc;
     ngx_pool_cleanup_t         *cln;
     ngx_ssl_stapling_t         *staple;

     staple = ngx_pcalloc(cf->pool, sizeof(ngx_ssl_stapling_t));
     if (staple == NULL) {
         return NGX_ERROR;
     }

     cln = ngx_pool_cleanup_add(cf->pool, 0);
     if (cln == NULL) {
         return NGX_ERROR;
     }

     cln->handler = ngx_ssl_stapling_cleanup;
     cln->data = staple;

     if (SSL_CTX_set_ex_data(ssl->ctx, ngx_ssl_stapling_index, staple)
         == 0)
     {
         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                       "SSL_CTX_set_ex_data() failed");
         return NGX_ERROR;
     }
```

```
145
146     staple->ssl_ctx = ssl->ctx;
147     staple->timeout = 60000;
148     staple->verify = verify;
149
150     if (file->len) {
151         /* use OCSP response from the file */
152
153         if (ngx_ssl_stapling_file(cf, ssl, file) != NGX_OK) {
154             return NGX_ERROR;
155         }
156
157         goto done;
158     }
159
160     rc = ngx_ssl_stapling_issuer(cf, ssl);
161
162     if (rc == NGX_DECLINED) {
163         return NGX_OK;
164     }
165
166     if (rc != NGX_OK) {
167         return NGX_ERROR;
168     }
169
170     rc = ngx_ssl_stapling_responder(cf, ssl, responder);
171
172     if (rc == NGX_DECLINED) {
173         return NGX_OK;
174     }
175
176     if (rc != NGX_OK) {
177         return NGX_ERROR;
178     }
179
180 done:
181
182     SSL_CTX_set_tlsext_status_cb(ssl->ctx, ngx_ssl_certificate_status_callback);
183     SSL_CTX_set_tlsext_status_arg(ssl->ctx, staple);
184
185     return NGX_OK;
186 }
187
188
189 static ngx_int_t
190 ngx_ssl_stapling_file(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *file)
191 {
192     BIO                 *bio;
193     int                  len;
194     u_char              *p, *buf;
195     OCSP_RESPONSE       *response;
196     ngx_ssl_stapling_t  *staple;
197
198     staple = SSL_CTX_get_ex_data(ssl->ctx, ngx_ssl_stapling_index);
199
200     if (ngx_conf_full_name(cf->cycle, file, 1) != NGX_OK) {
201         return NGX_ERROR;
202     }
203
204     bio = BIO_new_file((char *) file->data, "r");
205     if (bio == NULL) {
206         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
207                       "BIO_new_file(\"%s\") failed", file->data);
208         return NGX_ERROR;
209     }
210
211     response = d2i_OCSP_RESPONSE_bio(bio, NULL);
212     if (response == NULL) {
213         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
214                       "d2i_OCSP_RESPONSE_bio(\"%s\") failed", file->data);
215         BIO_free(bio);
216         return NGX_ERROR;
217     }
218
219     len = i2d_OCSP_RESPONSE(response, NULL);
220     if (len <= 0) {
```

```
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "i2d_OCSP_RESPONSE(\"%s\") failed", file->data);
        goto failed;
    }

    buf = ngx_alloc(len, ssl->log);
    if (buf == NULL) {
        goto failed;
    }

    p = buf;
    len = i2d_OCSP_RESPONSE(response, &p);
    if (len <= 0) {
        ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
                      "i2d_OCSP_RESPONSE(\"%s\") failed", file->data);
        ngx_free(buf);
        goto failed;
    }

    OCSP_RESPONSE_free(response);
    BIO_free(bio);

    staple->staple.data = buf;
    staple->staple.len = len;

    return NGX_OK;

failed:

    OCSP_RESPONSE_free(response);
    BIO_free(bio);

    return NGX_ERROR;
}


static ngx_int_t
ngx_ssl_stapling_issuer(ngx_conf_t *cf, ngx_ssl_t *ssl)
{
    int                  i, n, rc;
    X509                *cert, *issuer;
    X509_STORE          *store;
    X509_STORE_CTX      *store_ctx;
    STACK_OF(X509)      *chain;
    ngx_ssl_stapling_t  *staple;

    staple = SSL_CTX_get_ex_data(ssl->ctx, ngx_ssl_stapling_index);
    cert = SSL_CTX_get_ex_data(ssl->ctx, ngx_ssl_certificate_index);

#if OPENSSL_VERSION_NUMBER >= 0x10001000L
    SSL_CTX_get_extra_chain_certs(ssl->ctx, &chain);
#else
    chain = ssl->ctx->extra_certs;
#endif

    n = sk_X509_num(chain);

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ssl->log, 0,
                   "SSL get issuer: %d extra certs", n);

    for (i = 0; i < n; i++) {
        issuer = sk_X509_value(chain, i);
        if (X509_check_issued(issuer, cert) == X509_V_OK) {
            CRYPTO_add(&issuer->references, 1, CRYPTO_LOCK_X509);

            ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ssl->log, 0,
                           "SSL get issuer: found %p in extra certs", issuer);

            staple->cert = cert;
            staple->issuer = issuer;

            return NGX_OK;
        }
    }

    store = SSL_CTX_get_cert_store(ssl->ctx);
```

```
297     if (store == NULL) {
298         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
299                       "SSL_CTX_get_cert_store() failed");
300         return NGX_ERROR;
301     }
302 
303     store_ctx = X509_STORE_CTX_new();
304     if (store_ctx == NULL) {
305         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
306                       "X509_STORE_CTX_new() failed");
307         return NGX_ERROR;
308     }
309 
310     if (X509_STORE_CTX_init(store_ctx, store, NULL, NULL) == 0) {
311         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
312                       "X509_STORE_CTX_init() failed");
313         return NGX_ERROR;
314     }
315 
316     rc = X509_STORE_CTX_get1_issuer(&issuer, store_ctx, cert);
317 
318     if (rc == -1) {
319         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
320                       "X509_STORE_CTX_get1_issuer() failed");
321         X509_STORE_CTX_free(store_ctx);
322         return NGX_ERROR;
323     }
324 
325     if (rc == 0) {
326         ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
327                       "\"ssl_stapling\" ignored, issuer certificate not found");
328         X509_STORE_CTX_free(store_ctx);
329         return NGX_DECLINED;
330     }
331 
332     X509_STORE_CTX_free(store_ctx);
333 
334     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ssl->log, 0,
335                    "SSL get issuer: found %p in cert store", issuer);
336 
337     staple->cert = cert;
338     staple->issuer = issuer;
339 
340     return NGX_OK;
341 }
342 
343 
344 static ngx_int_t
345 ngx_ssl_stapling_responder(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *responder)
346 {
347     ngx_url_t                  u;
348     char                      *s;
349     ngx_ssl_stapling_t        *staple;
350     STACK_OF(OPENSSL_STRING)  *aia;
351 
352     staple = SSL_CTX_get_ex_data(ssl->ctx, ngx_ssl_stapling_index);
353 
354     if (responder->len == 0) {
355 
356         /* extract OCSP responder URL from certificate */
357 
358         aia = X509_get1_ocsp(staple->cert);
359         if (aia == NULL) {
360             ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
361                           "\"ssl_stapling\" ignored, "
362                           "no OCSP responder URL in the certificate");
363             return NGX_DECLINED;
364         }
365 
366 #if OPENSSL_VERSION_NUMBER >= 0x10000000L
367         s = sk_OPENSSL_STRING_value(aia, 0);
368 #else
369         s = sk_value(aia, 0);
370 #endif
371         if (s == NULL) {
372             ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
```

```
                          "\"ssl_stapling\" ignored, "
                          "no OCSP responder URL in the certificate");
            X509_email_free(aia);
            return NGX_DECLINED;
        }

        responder->len = ngx_strlen(s);
        responder->data = ngx_palloc(cf->pool, responder->len);
        if (responder->data == NULL) {
            X509_email_free(aia);
            return NGX_ERROR;
        }

        ngx_memcpy(responder->data, s, responder->len);
        X509_email_free(aia);
    }

    ngx_memzero(&u, sizeof(ngx_url_t));

    u.url = *responder;
    u.default_port = 80;
    u.uri_part = 1;

    if (u.url.len > 7
        && ngx_strncasecmp(u.url.data, (u_char *) "http://", 7) == 0)
    {
        u.url.len -= 7;
        u.url.data += 7;

    } else {
        ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
                      "\"ssl_stapling\" ignored, "
                      "invalid URL prefix in OCSP responder \"%V\"", &u.url);
        return NGX_DECLINED;
    }

    if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
        if (u.err) {
            ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
                          "\"ssl_stapling\" ignored, "
                          "%s in OCSP responder \"%V\"", u.err, &u.url);
            return NGX_DECLINED;
        }

        return NGX_ERROR;
    }

    staple->addrs = u.addrs;
    staple->host = u.host;
    staple->uri = u.uri;
    staple->port = u.port;

    if (staple->uri.len == 0) {
        ngx_str_set(&staple->uri, "/");
    }

    return NGX_OK;
}


ngx_int_t
ngx_ssl_stapling_resolver(ngx_conf_t *cf, ngx_ssl_t *ssl,
    ngx_resolver_t *resolver, ngx_msec_t resolver_timeout)
{
    ngx_ssl_stapling_t  *staple;

    staple = SSL_CTX_get_ex_data(ssl->ctx, ngx_ssl_stapling_index);

    staple->resolver = resolver;
    staple->resolver_timeout = resolver_timeout;

    return NGX_OK;
}


static int
```

```c
ngx_ssl_certificate_status_callback(ngx_ssl_conn_t *ssl_conn, void *data)
{
    int                 rc;
    u_char             *p;
    ngx_connection_t   *c;
    ngx_ssl_stapling_t  *staple;

    c = ngx_ssl_get_connection(ssl_conn);

    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "SSL certificate status callback");

    staple = data;
    rc = SSL_TLSEXT_ERR_NOACK;

    if (staple->staple.len) {
        /* we have to copy ocsp response as OpenSSL will free it by itself */

        p = OPENSSL_malloc(staple->staple.len);
        if (p == NULL) {
            ngx_ssl_error(NGX_LOG_ALERT, c->log, 0, "OPENSSL_malloc() failed");
            return SSL_TLSEXT_ERR_NOACK;
        }

        ngx_memcpy(p, staple->staple.data, staple->staple.len);

        SSL_set_tlsext_status_ocsp_resp(ssl_conn, p, staple->staple.len);

        rc = SSL_TLSEXT_ERR_OK;
    }

    ngx_ssl_stapling_update(staple);

    return rc;
}


static void
ngx_ssl_stapling_update(ngx_ssl_stapling_t *staple)
{
    ngx_ssl_ocsp_ctx_t  *ctx;

    if (staple->host.len == 0
        || staple->loading || staple->valid >= ngx_time())
    {
        return;
    }

    staple->loading = 1;

    ctx = ngx_ssl_ocsp_start();
    if (ctx == NULL) {
        return;
    }

    ctx->cert = staple->cert;
    ctx->issuer = staple->issuer;

    ctx->addrs = staple->addrs;
    ctx->host = staple->host;
    ctx->uri = staple->uri;
    ctx->port = staple->port;
    ctx->timeout = staple->timeout;

    ctx->resolver = staple->resolver;
    ctx->resolver_timeout = staple->resolver_timeout;

    ctx->handler = ngx_ssl_stapling_ocsp_handler;
    ctx->data = staple;

    ngx_ssl_ocsp_request(ctx);

    return;
}
```

```c
static void
ngx_ssl_stapling_ocsp_handler(ngx_ssl_ocsp_ctx_t *ctx)
{
#if OPENSSL_VERSION_NUMBER >= 0x0090707fL
    const
#endif
    u_char                *p;
    int                    n;
    size_t                 len;
    ngx_str_t              response;
    X509_STORE            *store;
    STACK_OF(X509)        *chain;
    OCSP_CERTID           *id;
    OCSP_RESPONSE         *ocsp;
    OCSP_BASICRESP        *basic;
    ngx_ssl_stapling_t    *staple;
    ASN1_GENERALIZEDTIME  *thisupdate, *nextupdate;

    staple = ctx->data;
    ocsp = NULL;
    basic = NULL;
    id = NULL;

    if (ctx->code != 200) {
        goto error;
    }

    /* check the response */

    len = ctx->response->last - ctx->response->pos;
    p = ctx->response->pos;

    ocsp = d2i_OCSP_RESPONSE(NULL, &p, len);
    if (ocsp == NULL) {
        ngx_ssl_error(NGX_LOG_ERR, ctx->log, 0,
                      "d2i_OCSP_RESPONSE() failed");
        goto error;
    }

    n = OCSP_response_status(ocsp);

    if (n != OCSP_RESPONSE_STATUS_SUCCESSFUL) {
        ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
                      "OCSP response not successful (%d: %s)",
                      n, OCSP_response_status_str(n));
        goto error;
    }

    basic = OCSP_response_get1_basic(ocsp);
    if (basic == NULL) {
        ngx_ssl_error(NGX_LOG_ERR, ctx->log, 0,
                      "OCSP_response_get1_basic() failed");
        goto error;
    }

    store = SSL_CTX_get_cert_store(staple->ssl_ctx);
    if (store == NULL) {
        ngx_ssl_error(NGX_LOG_CRIT, ctx->log, 0,
                      "SSL_CTX_get_cert_store() failed");
        goto error;
    }

#if OPENSSL_VERSION_NUMBER >= 0x10001000L
    SSL_CTX_get_extra_chain_certs(staple->ssl_ctx, &chain);
#else
    chain = staple->ssl_ctx->extra_certs;
#endif

    if (OCSP_basic_verify(basic, chain, store,
                          staple->verify ? OCSP_TRUSTOTHER : OCSP_NOVERIFY)
        != 1)
    {
        ngx_ssl_error(NGX_LOG_ERR, ctx->log, 0,
                      "OCSP_basic_verify() failed");
        goto error;
    }
```

```
601
602     id = OCSP_cert_to_id(NULL, ctx->cert, ctx->issuer);
603     if (id == NULL) {
604         ngx_ssl_error(NGX_LOG_CRIT, ctx->log, 0,
605                       "OCSP_cert_to_id() failed");
606         goto error;
607     }
608
609     if (OCSP_resp_find_status(basic, id, &n, NULL, NULL,
610                               &thisupdate, &nextupdate)
611         != 1)
612     {
613         ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
614                       "certificate status not found in the OCSP response");
615         goto error;
616     }
617
618     if (n != V_OCSP_CERTSTATUS_GOOD) {
619         ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
620                       "certificate status \"%s\" in the OCSP response",
621                       OCSP_cert_status_str(n));
622         goto error;
623     }
624
625     if (OCSP_check_validity(thisupdate, nextupdate, 300, -1) != 1) {
626         ngx_ssl_error(NGX_LOG_ERR, ctx->log, 0,
627                       "OCSP_check_validity() failed");
628         goto error;
629     }
630
631     OCSP_CERTID_free(id);
632     OCSP_BASICRESP_free(basic);
633     OCSP_RESPONSE_free(ocsp);
634
635     /* copy the response to memory not in ctx->pool */
636
637     response.len = len;
638     response.data = ngx_alloc(response.len, ctx->log);
639
640     if (response.data == NULL) {
641         goto done;
642     }
643
644     ngx_memcpy(response.data, ctx->response->pos, response.len);
645
646     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
647                    "ssl ocsp response, %s, %uz",
648                    OCSP_cert_status_str(n), response.len);
649
650     if (staple->staple.data) {
651         ngx_free(staple->staple.data);
652     }
653
654     staple->staple = response;
655
656 done:
657
658     staple->loading = 0;
659     staple->valid = ngx_time() + 3600; /* ssl_stapling_valid */
660
661     ngx_ssl_ocsp_done(ctx);
662     return;
663
664 error:
665
666     staple->loading = 0;
667     staple->valid = ngx_time() + 300; /* ssl_stapling_err_valid */
668
669     if (id) {
670         OCSP_CERTID_free(id);
671     }
672
673     if (basic) {
674         OCSP_BASICRESP_free(basic);
675     }
676
```

```
    if (ocsp) {
        OCSP_RESPONSE_free(ocsp);
    }

    ngx_ssl_ocsp_done(ctx);
}


static void
ngx_ssl_stapling_cleanup(void *data)
{
    ngx_ssl_stapling_t  *staple = data;

    if (staple->issuer) {
        X509_free(staple->issuer);
    }

    if (staple->staple.data) {
        ngx_free(staple->staple.data);
    }
}


static ngx_ssl_ocsp_ctx_t *
ngx_ssl_ocsp_start(void)
{
    ngx_log_t           *log;
    ngx_pool_t          *pool;
    ngx_ssl_ocsp_ctx_t  *ctx;

    pool = ngx_create_pool(2048, ngx_cycle->log);
    if (pool == NULL) {
        return NULL;
    }

    ctx = ngx_pcalloc(pool, sizeof(ngx_ssl_ocsp_ctx_t));
    if (ctx == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    log = ngx_palloc(pool, sizeof(ngx_log_t));
    if (log == NULL) {
        ngx_destroy_pool(pool);
        return NULL;
    }

    ctx->pool = pool;

    *log = *ctx->pool->log;

    ctx->pool->log = log;
    ctx->log = log;

    log->handler = ngx_ssl_ocsp_log_error;
    log->data = ctx;
    log->action = "requesting certificate status";

    return ctx;
}


static void
ngx_ssl_ocsp_done(ngx_ssl_ocsp_ctx_t *ctx)
{
    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
                   "ssl ocsp done");

    if (ctx->peer.connection) {
        ngx_close_connection(ctx->peer.connection);
    }

    ngx_destroy_pool(ctx->pool);
}
```

```
753 static void
754 ngx_ssl_ocsp_error(ngx_ssl_ocsp_ctx_t *ctx)
755 {
756     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
757                    "ssl ocsp error");
758 
759     ctx->code = 0;
760     ctx->handler(ctx);
761 }
762 
763 
764 static void
765 ngx_ssl_ocsp_request(ngx_ssl_ocsp_ctx_t *ctx)
766 {
767     ngx_resolver_ctx_t  *resolve, temp;
768 
769     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
770                    "ssl ocsp request");
771 
772     if (ngx_ssl_ocsp_create_request(ctx) != NGX_OK) {
773         ngx_ssl_ocsp_error(ctx);
774         return;
775     }
776 
777     if (ctx->resolver) {
778         /* resolve OCSP responder hostname */
779 
780         temp.name = ctx->host;
781 
782         resolve = ngx_resolve_start(ctx->resolver, &temp);
783         if (resolve == NULL) {
784             ngx_ssl_ocsp_error(ctx);
785             return;
786         }
787 
788         if (resolve == NGX_NO_RESOLVER) {
789             ngx_log_error(NGX_LOG_WARN, ctx->log, 0,
790                           "no resolver defined to resolve %V", &ctx->host);
791             goto connect;
792         }
793 
794         resolve->name = ctx->host;
795         resolve->handler = ngx_ssl_ocsp_resolve_handler;
796         resolve->data = ctx;
797         resolve->timeout = ctx->resolver_timeout;
798 
799         if (ngx_resolve_name(resolve) != NGX_OK) {
800             ngx_ssl_ocsp_error(ctx);
801             return;
802         }
803 
804         return;
805     }
806 
807 connect:
808 
809     ngx_ssl_ocsp_connect(ctx);
810 }
811 
812 
813 static void
814 ngx_ssl_ocsp_resolve_handler(ngx_resolver_ctx_t *resolve)
815 {
816     ngx_ssl_ocsp_ctx_t *ctx = resolve->data;
817 
818     u_char           *p;
819     size_t            len;
820     in_port_t         port;
821     socklen_t         socklen;
822     ngx_uint_t        i;
823     struct sockaddr  *sockaddr;
824 
825     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
826                    "ssl ocsp resolve handler");
827 
828     if (resolve->state) {
```

```
        ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
                      "%V could not be resolved (%i: %s)",
                      &resolve->name, resolve->state,
                      ngx_resolver_strerror(resolve->state));
        goto failed;
    }

#if (NGX_DEBUG)
    {
    u_char     text[NGX_SOCKADDR_STRLEN];
    ngx_str_t  addr;

    addr.data = text;

    for (i = 0; i < resolve->naddrs; i++) {
        addr.len = ngx_sock_ntop(resolve->addrs[i].sockaddr,
                                 resolve->addrs[i].socklen,
                                 text, NGX_SOCKADDR_STRLEN, 0);

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
                       "name was resolved to %V", &addr);

    }
    }
#endif

    ctx->naddrs = resolve->naddrs;
    ctx->addrs = ngx_pcalloc(ctx->pool, ctx->naddrs * sizeof(ngx_addr_t));

    if (ctx->addrs == NULL) {
        goto failed;
    }

    port = htons(ctx->port);

    for (i = 0; i < resolve->naddrs; i++) {

        socklen = resolve->addrs[i].socklen;

        sockaddr = ngx_palloc(ctx->pool, socklen);
        if (sockaddr == NULL) {
            goto failed;
        }

        ngx_memcpy(sockaddr, resolve->addrs[i].sockaddr, socklen);

        switch (sockaddr->sa_family) {
#if (NGX_HAVE_INET6)
        case AF_INET6:
            ((struct sockaddr_in6 *) sockaddr)->sin6_port = port;
            break;
#endif
        default: /* AF_INET */
            ((struct sockaddr_in *) sockaddr)->sin_port = port;
        }

        ctx->addrs[i].sockaddr = sockaddr;
        ctx->addrs[i].socklen = socklen;

        p = ngx_pnalloc(ctx->pool, NGX_SOCKADDR_STRLEN);
        if (p == NULL) {
            goto failed;
        }

        len = ngx_sock_ntop(sockaddr, socklen, p, NGX_SOCKADDR_STRLEN, 1);

        ctx->addrs[i].name.len = len;
        ctx->addrs[i].name.data = p;
    }

    ngx_resolve_name_done(resolve);

    ngx_ssl_ocsp_connect(ctx);
    return;

failed:
```

```
905
906     ngx_resolve_name_done(resolve);
907     ngx_ssl_ocsp_error(ctx);
908 }
909
910
911 static void
912 ngx_ssl_ocsp_connect(ngx_ssl_ocsp_ctx_t *ctx)
913 {
914     ngx_int_t    rc;
915
916     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
917                    "ssl ocsp connect");
918
919     /* TODO: use all ip addresses */
920
921     ctx->peer.sockaddr = ctx->addrs[0].sockaddr;
922     ctx->peer.socklen = ctx->addrs[0].socklen;
923     ctx->peer.name = &ctx->addrs[0].name;
924     ctx->peer.get = ngx_event_get_peer;
925     ctx->peer.log = ctx->log;
926     ctx->peer.log_error = NGX_ERROR_ERR;
927
928     rc = ngx_event_connect_peer(&ctx->peer);
929
930     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
931                    "ssl ocsp connect peer done");
932
933     if (rc == NGX_ERROR || rc == NGX_BUSY || rc == NGX_DECLINED) {
934         ngx_ssl_ocsp_error(ctx);
935         return;
936     }
937
938     ctx->peer.connection->data = ctx;
939     ctx->peer.connection->pool = ctx->pool;
940
941     ctx->peer.connection->read->handler = ngx_ssl_ocsp_read_handler;
942     ctx->peer.connection->write->handler = ngx_ssl_ocsp_write_handler;
943
944     ctx->process = ngx_ssl_ocsp_process_status_line;
945
946     ngx_add_timer(ctx->peer.connection->read, ctx->timeout);
947     ngx_add_timer(ctx->peer.connection->write, ctx->timeout);
948
949     if (rc == NGX_OK) {
950         ngx_ssl_ocsp_write_handler(ctx->peer.connection->write);
951         return;
952     }
953 }
954
955
956 static void
957 ngx_ssl_ocsp_write_handler(ngx_event_t *wev)
958 {
959     ssize_t              n, size;
960     ngx_connection_t    *c;
961     ngx_ssl_ocsp_ctx_t  *ctx;
962
963     c = wev->data;
964     ctx = c->data;
965
966     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, wev->log, 0,
967                    "ssl ocsp write handler");
968
969     if (wev->timedout) {
970         ngx_log_error(NGX_LOG_ERR, wev->log, NGX_ETIMEDOUT,
971                       "OCSP responder timed out");
972         ngx_ssl_ocsp_error(ctx);
973         return;
974     }
975
976     size = ctx->request->last - ctx->request->pos;
977
978     n = ngx_send(c, ctx->request->pos, size);
979
980     if (n == NGX_ERROR) {
```

```
        ngx_ssl_ocsp_error(ctx);
        return;
    }

    if (n > 0) {
        ctx->request->pos += n;

        if (n == size) {
            wev->handler = ngx_ssl_ocsp_dummy_handler;

            if (wev->timer_set) {
                ngx_del_timer(wev);
            }

            if (ngx_handle_write_event(wev, 0) != NGX_OK) {
                ngx_ssl_ocsp_error(ctx);
            }

            return;
        }
    }

    if (!wev->timer_set) {
        ngx_add_timer(wev, ctx->timeout);
    }
}


static void
ngx_ssl_ocsp_read_handler(ngx_event_t *rev)
{
    ssize_t            n, size;
    ngx_int_t          rc;
    ngx_ssl_ocsp_ctx_t    *ctx;
    ngx_connection_t  *c;

    c = rev->data;
    ctx = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, rev->log, 0,
                   "ssl ocsp read handler");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_ERR, rev->log, NGX_ETIMEDOUT,
                      "OCSP responder timed out");
        ngx_ssl_ocsp_error(ctx);
        return;
    }

    if (ctx->response == NULL) {
        ctx->response = ngx_create_temp_buf(ctx->pool, 16384);
        if (ctx->response == NULL) {
            ngx_ssl_ocsp_error(ctx);
            return;
        }
    }

    for ( ;; ) {

        size = ctx->response->end - ctx->response->last;

        n = ngx_recv(c, ctx->response->last, size);

        if (n > 0) {
            ctx->response->last += n;

            rc = ctx->process(ctx);

            if (rc == NGX_ERROR) {
                ngx_ssl_ocsp_error(ctx);
                return;
            }

            continue;
        }
```

```
        if (n == NGX_AGAIN) {

            if (ngx_handle_read_event(rev, 0) != NGX_OK) {
                ngx_ssl_ocsp_error(ctx);
            }

            return;
        }

        break;
    }

    ctx->done = 1;

    rc = ctx->process(ctx);

    if (rc == NGX_DONE) {
        /* ctx->handler() was called */
        return;
    }

    ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
                  "OCSP responder prematurely closed connection");

    ngx_ssl_ocsp_error(ctx);
}


static void
ngx_ssl_ocsp_dummy_handler(ngx_event_t *ev)
{
    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "ssl ocsp dummy handler");
}


static ngx_int_t
ngx_ssl_ocsp_create_request(ngx_ssl_ocsp_ctx_t *ctx)
{
    int            len;
    u_char        *p;
    uintptr_t      escape;
    ngx_str_t      binary, base64;
    ngx_buf_t     *b;
    OCSP_CERTID   *id;
    OCSP_REQUEST  *ocsp;

    ocsp = OCSP_REQUEST_new();
    if (ocsp == NULL) {
        ngx_ssl_error(NGX_LOG_CRIT, ctx->log, 0,
                      "OCSP_REQUEST_new() failed");
        return NGX_ERROR;
    }

    id = OCSP_cert_to_id(NULL, ctx->cert, ctx->issuer);
    if (id == NULL) {
        ngx_ssl_error(NGX_LOG_CRIT, ctx->log, 0,
                      "OCSP_cert_to_id() failed");
        goto failed;
    }

    if (OCSP_request_add0_id(ocsp, id) == NULL) {
        ngx_ssl_error(NGX_LOG_CRIT, ctx->log, 0,
                      "OCSP_request_add0_id() failed");
        goto failed;
    }

    len = i2d_OCSP_REQUEST(ocsp, NULL);
    if (len <= 0) {
        ngx_ssl_error(NGX_LOG_CRIT, ctx->log, 0,
                      "i2d_OCSP_REQUEST() failed");
        goto failed;
    }

    binary.len = len;
    binary.data = ngx_palloc(ctx->pool, len);
```

```
    if (binary.data == NULL) {
        goto failed;
    }

    p = binary.data;
    len = i2d_OCSP_REQUEST(ocsp, &p);
    if (len <= 0) {
        ngx_ssl_error(NGX_LOG_EMERG, ctx->log, 0,
                      "i2d_OCSP_REQUEST() failed");
        goto failed;
    }

    base64.len = ngx_base64_encoded_length(binary.len);
    base64.data = ngx_palloc(ctx->pool, base64.len);
    if (base64.data == NULL) {
        goto failed;
    }

    ngx_encode_base64(&base64, &binary);

    escape = ngx_escape_uri(NULL, base64.data, base64.len,
                            NGX_ESCAPE_URI_COMPONENT);

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
                   "ssl ocsp request length %z, escape %d",
                   base64.len, escape);

    len = sizeof("GET ") - 1 + ctx->uri.len + sizeof("/") - 1
          + base64.len + 2 * escape + sizeof(" HTTP/1.0" CRLF) - 1
          + sizeof("Host: ") - 1 + ctx->host.len + sizeof(CRLF) - 1
          + sizeof(CRLF) - 1;

    b = ngx_create_temp_buf(ctx->pool, len);
    if (b == NULL) {
        goto failed;
    }

    p = b->last;

    p = ngx_cpymem(p, "GET ", sizeof("GET ") - 1);
    p = ngx_cpymem(p, ctx->uri.data, ctx->uri.len);

    if (ctx->uri.data[ctx->uri.len - 1] != '/') {
        *p++ = '/';
    }

    if (escape == 0) {
        p = ngx_cpymem(p, base64.data, base64.len);

    } else {
        p = (u_char *) ngx_escape_uri(p, base64.data, base64.len,
                                      NGX_ESCAPE_URI_COMPONENT);
    }

    p = ngx_cpymem(p, " HTTP/1.0" CRLF, sizeof(" HTTP/1.0" CRLF) - 1);
    p = ngx_cpymem(p, "Host: ", sizeof("Host: ") - 1);
    p = ngx_cpymem(p, ctx->host.data, ctx->host.len);
    *p++ = CR; *p++ = LF;

    /* add "\r\n" at the header end */
    *p++ = CR; *p++ = LF;

    b->last = p;
    ctx->request = b;

    OCSP_REQUEST_free(ocsp);

    return NGX_OK;

failed:

    OCSP_REQUEST_free(ocsp);

    return NGX_ERROR;
}
```

```
1209
1210 static ngx_int_t
1211 ngx_ssl_ocsp_process_status_line(ngx_ssl_ocsp_ctx_t *ctx)
1212 {
1213     ngx_int_t  rc;
1214
1215     rc = ngx_ssl_ocsp_parse_status_line(ctx);
1216
1217     if (rc == NGX_OK) {
1218 #if 0
1219         ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
1220                        "ssl ocsp status line \"%*s\"",
1221                        ctx->response->pos - ctx->response->start,
1222                        ctx->response->start);
1223 #endif
1224
1225         ctx->process = ngx_ssl_ocsp_process_headers;
1226         return ctx->process(ctx);
1227     }
1228
1229     if (rc == NGX_AGAIN) {
1230         return NGX_AGAIN;
1231     }
1232
1233     /* rc == NGX_ERROR */
1234
1235     ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
1236                   "OCSP responder sent invalid response");
1237
1238     return NGX_ERROR;
1239 }
1240
1241
1242 static ngx_int_t
1243 ngx_ssl_ocsp_parse_status_line(ngx_ssl_ocsp_ctx_t *ctx)
1244 {
1245     u_char      ch;
1246     u_char     *p;
1247     ngx_buf_t  *b;
1248     enum {
1249         sw_start = 0,
1250         sw_H,
1251         sw_HT,
1252         sw_HTT,
1253         sw_HTTP,
1254         sw_first_major_digit,
1255         sw_major_digit,
1256         sw_first_minor_digit,
1257         sw_minor_digit,
1258         sw_status,
1259         sw_space_after_status,
1260         sw_status_text,
1261         sw_almost_done
1262     } state;
1263
1264     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
1265                    "ssl ocsp process status line");
1266
1267     state = ctx->state;
1268     b = ctx->response;
1269
1270     for (p = b->pos; p < b->last; p++) {
1271         ch = *p;
1272
1273         switch (state) {
1274
1275         /* "HTTP/" */
1276         case sw_start:
1277             switch (ch) {
1278             case 'H':
1279                 state = sw_H;
1280                 break;
1281             default:
1282                 return NGX_ERROR;
1283             }
1284             break;
```

```
1285
1286        case sw_H:
1287            switch (ch) {
1288            case 'T':
1289                state = sw_HT;
1290                break;
1291            default:
1292                return NGX_ERROR;
1293            }
1294            break;
1295
1296        case sw_HT:
1297            switch (ch) {
1298            case 'T':
1299                state = sw_HTT;
1300                break;
1301            default:
1302                return NGX_ERROR;
1303            }
1304            break;
1305
1306        case sw_HTT:
1307            switch (ch) {
1308            case 'P':
1309                state = sw_HTTP;
1310                break;
1311            default:
1312                return NGX_ERROR;
1313            }
1314            break;
1315
1316        case sw_HTTP:
1317            switch (ch) {
1318            case '/':
1319                state = sw_first_major_digit;
1320                break;
1321            default:
1322                return NGX_ERROR;
1323            }
1324            break;
1325
1326        /* the first digit of major HTTP version */
1327        case sw_first_major_digit:
1328            if (ch < '1' || ch > '9') {
1329                return NGX_ERROR;
1330            }
1331
1332            state = sw_major_digit;
1333            break;
1334
1335        /* the major HTTP version or dot */
1336        case sw_major_digit:
1337            if (ch == '.') {
1338                state = sw_first_minor_digit;
1339                break;
1340            }
1341
1342            if (ch < '0' || ch > '9') {
1343                return NGX_ERROR;
1344            }
1345
1346            break;
1347
1348        /* the first digit of minor HTTP version */
1349        case sw_first_minor_digit:
1350            if (ch < '0' || ch > '9') {
1351                return NGX_ERROR;
1352            }
1353
1354            state = sw_minor_digit;
1355            break;
1356
1357        /* the minor HTTP version or the end of the request line */
1358        case sw_minor_digit:
1359            if (ch == ' ') {
1360                state = sw_status;
```

```
1361                break;
1362            }
1363
1364            if (ch < '0' || ch > '9') {
1365                return NGX_ERROR;
1366            }
1367
1368            break;
1369
1370        /* HTTP status code */
1371        case sw_status:
1372            if (ch == ' ') {
1373                break;
1374            }
1375
1376            if (ch < '0' || ch > '9') {
1377                return NGX_ERROR;
1378            }
1379
1380            ctx->code = ctx->code * 10 + ch - '0';
1381
1382            if (++ctx->count == 3) {
1383                state = sw_space_after_status;
1384            }
1385
1386            break;
1387
1388        /* space or end of line */
1389        case sw_space_after_status:
1390            switch (ch) {
1391            case ' ':
1392                state = sw_status_text;
1393                break;
1394            case '.':                    /* IIS may send 403.1, 403.2, etc */
1395                state = sw_status_text;
1396                break;
1397            case CR:
1398                state = sw_almost_done;
1399                break;
1400            case LF:
1401                goto done;
1402            default:
1403                return NGX_ERROR;
1404            }
1405            break;
1406
1407        /* any text until end of line */
1408        case sw_status_text:
1409            switch (ch) {
1410            case CR:
1411                state = sw_almost_done;
1412                break;
1413            case LF:
1414                goto done;
1415            }
1416            break;
1417
1418        /* end of status line */
1419        case sw_almost_done:
1420            switch (ch) {
1421            case LF:
1422                goto done;
1423            default:
1424                return NGX_ERROR;
1425            }
1426        }
1427    }
1428
1429    b->pos = p;
1430    ctx->state = state;
1431
1432    return NGX_AGAIN;
1433
1434done:
1435
1436    b->pos = p + 1;
```

```
    ctx->state = sw_start;

    return NGX_OK;
}


static ngx_int_t
ngx_ssl_ocsp_process_headers(ngx_ssl_ocsp_ctx_t *ctx)
{
    size_t     len;
    ngx_int_t  rc;

    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
                   "ssl ocsp process headers");

    for ( ;; ) {
        rc = ngx_ssl_ocsp_parse_header_line(ctx);

        if (rc == NGX_OK) {

            ngx_log_debug4(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
                           "ssl ocsp header \"%*s: %*s\"",
                           ctx->header_name_end - ctx->header_name_start,
                           ctx->header_name_start,
                           ctx->header_end - ctx->header_start,
                           ctx->header_start);

            len = ctx->header_name_end - ctx->header_name_start;

            if (len == sizeof("Content-Type") - 1
                && ngx_strncasecmp(ctx->header_name_start,
                                   (u_char *) "Content-Type",
                                   sizeof("Content-Type") - 1)
                   == 0)
            {
                len = ctx->header_end - ctx->header_start;

                if (len != sizeof("application/ocsp-response") - 1
                    || ngx_strncasecmp(ctx->header_start,
                                       (u_char *) "application/ocsp-response",
                                       sizeof("application/ocsp-response") - 1)
                       != 0)
                {
                    ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
                                  "OCSP responder sent invalid "
                                  "\"Content-Type\" header: \"%*s\"",
                                  ctx->header_end - ctx->header_start,
                                  ctx->header_start);
                    return NGX_ERROR;
                }

                continue;
            }

            /* TODO: honor Content-Length */

            continue;
        }

        if (rc == NGX_DONE) {
            break;
        }

        if (rc == NGX_AGAIN) {
            return NGX_AGAIN;
        }

        /* rc == NGX_ERROR */

        ngx_log_error(NGX_LOG_ERR, ctx->log, 0,
                      "OCSP responder sent invalid response");

        return NGX_ERROR;
    }

    ctx->process = ngx_ssl_ocsp_process_body;
```

```
1513      return ctx->process(ctx);
1514  }
1515
1516  static ngx_int_t
1517  ngx_ssl_ocsp_parse_header_line(ngx_ssl_ocsp_ctx_t *ctx)
1518  {
1519      u_char      c, ch, *p;
1520      enum {
1521          sw_start = 0,
1522          sw_name,
1523          sw_space_before_value,
1524          sw_value,
1525          sw_space_after_value,
1526          sw_almost_done,
1527          sw_header_almost_done
1528      } state;
1529
1530      state = ctx->state;
1531
1532      for (p = ctx->response->pos; p < ctx->response->last; p++) {
1533          ch = *p;
1534
1535  #if 0
1536          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
1537                         "s:%d in:'%02Xd:%c'", state, ch, ch);
1538  #endif
1539
1540          switch (state) {
1541
1542          /* first char */
1543          case sw_start:
1544
1545              switch (ch) {
1546              case CR:
1547                  ctx->header_end = p;
1548                  state = sw_header_almost_done;
1549                  break;
1550              case LF:
1551                  ctx->header_end = p;
1552                  goto header_done;
1553              default:
1554                  state = sw_name;
1555                  ctx->header_name_start = p;
1556
1557                  c = (u_char) (ch | 0x20);
1558                  if (c >= 'a' && c <= 'z') {
1559                      break;
1560                  }
1561
1562                  if (ch >= '0' && ch <= '9') {
1563                      break;
1564                  }
1565
1566                  return NGX_ERROR;
1567              }
1568              break;
1569
1570          /* header name */
1571          case sw_name:
1572              c = (u_char) (ch | 0x20);
1573              if (c >= 'a' && c <= 'z') {
1574                  break;
1575              }
1576
1577              if (ch == ':') {
1578                  ctx->header_name_end = p;
1579                  state = sw_space_before_value;
1580                  break;
1581              }
1582
1583              if (ch == '-') {
1584                  break;
1585              }
1586
1587              if (ch >= '0' && ch <= '9') {
1588                  break;
```

```
            }

            if (ch == CR) {
                ctx->header_name_end = p;
                ctx->header_start = p;
                ctx->header_end = p;
                state = sw_almost_done;
                break;
            }

            if (ch == LF) {
                ctx->header_name_end = p;
                ctx->header_start = p;
                ctx->header_end = p;
                goto done;
            }

            return NGX_ERROR;

        /* space* before header value */
        case sw_space_before_value:
            switch (ch) {
            case ' ':
                break;
            case CR:
                ctx->header_start = p;
                ctx->header_end = p;
                state = sw_almost_done;
                break;
            case LF:
                ctx->header_start = p;
                ctx->header_end = p;
                goto done;
            default:
                ctx->header_start = p;
                state = sw_value;
                break;
            }
            break;

        /* header value */
        case sw_value:
            switch (ch) {
            case ' ':
                ctx->header_end = p;
                state = sw_space_after_value;
                break;
            case CR:
                ctx->header_end = p;
                state = sw_almost_done;
                break;
            case LF:
                ctx->header_end = p;
                goto done;
            }
            break;

        /* space* before end of header line */
        case sw_space_after_value:
            switch (ch) {
            case ' ':
                break;
            case CR:
                state = sw_almost_done;
                break;
            case LF:
                goto done;
            default:
                state = sw_value;
                break;
            }
            break;

        /* end of header line */
        case sw_almost_done:
            switch (ch) {
```

```
            case LF:
                goto done;
            default:
                return NGX_ERROR;
            }

        /* end of header */
        case sw_header_almost_done:
            switch (ch) {
            case LF:
                goto header_done;
            default:
                return NGX_ERROR;
            }
        }
    }

    ctx->response->pos = p;
    ctx->state = state;

    return NGX_AGAIN;

done:

    ctx->response->pos = p + 1;
    ctx->state = sw_start;

    return NGX_OK;

header_done:

    ctx->response->pos = p + 1;
    ctx->state = sw_start;

    return NGX_DONE;
}


static ngx_int_t
ngx_ssl_ocsp_process_body(ngx_ssl_ocsp_ctx_t *ctx)
{
    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ctx->log, 0,
                   "ssl ocsp process body");

    if (ctx->done) {
        ctx->handler(ctx);
        return NGX_DONE;
    }

    return NGX_AGAIN;
}


static u_char *
ngx_ssl_ocsp_log_error(ngx_log_t *log, u_char *buf, size_t len)
{
    u_char              *p;
    ngx_ssl_ocsp_ctx_t  *ctx;

    p = buf;

    if (log->action) {
        p = ngx_snprintf(buf, len, " while %s", log->action);
        len -= p - buf;
    }

    ctx = log->data;

    if (ctx) {
        p = ngx_snprintf(p, len, ", responder: %V", &ctx->host);
    }

    return p;
}
```

```
1741 #else
1742
1743
1744 ngx_int_t
1745 ngx_ssl_stapling(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *file,
1746     ngx_str_t *responder, ngx_uint_t verify)
1747 {
1748     ngx_log_error(NGX_LOG_WARN, ssl->log, 0,
1749                   "\"ssl_stapling\" ignored, not supported");
1750
1751     return NGX_OK;
1752 }
1753
1754 ngx_int_t
1755 ngx_ssl_stapling_resolver(ngx_conf_t *cf, ngx_ssl_t *ssl,
1756     ngx_resolver_t *resolver, ngx_msec_t resolver_timeout)
1757 {
1758     return NGX_OK;
1759 }
1760
1761
1762 #endif
```

One Level Up Top Level

src/core/ngx_resolver.h - nginx-1.7.10

Data types defined

- ngx_resolver_ctx_s
- ngx_resolver_ctx_t
- ngx_resolver_handler_pt
- ngx_resolver_node_t
- ngx_resolver_t
- ngx_udp_connection_t

Macros defined

- NGX_NO_RESOLVER
- NGX_RESOLVER_MAX_RECURSION
- NGX_RESOLVE_A
- NGX_RESOLVE_AAAA
- NGX_RESOLVE_CNAME
- NGX_RESOLVE_DNAME
- NGX_RESOLVE_FORMERR
- NGX_RESOLVE_MX
- NGX_RESOLVE_NOTIMP
- NGX_RESOLVE_NXDOMAIN
- NGX_RESOLVE_PTR
- NGX_RESOLVE_REFUSED
- NGX_RESOLVE_SERVFAIL
- NGX_RESOLVE_TIMEDOUT
- NGX_RESOLVE_TXT
- _NGX_RESOLVER_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
```

```
#ifndef _NGX_RESOLVER_H_INCLUDED_
#define _NGX_RESOLVER_H_INCLUDED_


#define NGX_RESOLVE_A         1
#define NGX_RESOLVE_CNAME     5
#define NGX_RESOLVE_PTR       12
#define NGX_RESOLVE_MX        15
#define NGX_RESOLVE_TXT       16
#if (NGX_HAVE_INET6)
#define NGX_RESOLVE_AAAA      28
#endif
#define NGX_RESOLVE_DNAME     39

#define NGX_RESOLVE_FORMERR   1
#define NGX_RESOLVE_SERVFAIL  2
#define NGX_RESOLVE_NXDOMAIN  3
#define NGX_RESOLVE_NOTIMP    4
#define NGX_RESOLVE_REFUSED   5
#define NGX_RESOLVE_TIMEDOUT  NGX_ETIMEDOUT


#define NGX_NO_RESOLVER       (void *) -1

#define NGX_RESOLVER_MAX_RECURSION    50


typedef struct {
    ngx_connection_t         *connection;
    struct sockaddr          *sockaddr;
    socklen_t                 socklen;
    ngx_str_t                 server;
    ngx_log_t                 log;
} ngx_udp_connection_t;


typedef struct ngx_resolver_ctx_s  ngx_resolver_ctx_t;

typedef void (*ngx_resolver_handler_pt)(ngx_resolver_ctx_t *ctx);


typedef struct {
    /* PTR: resolved name, A: name to resolve */
    u_char                   *name;

    ngx_queue_t               queue;

    /* event ident must be after 3 pointers as in ngx_connection_t */
    ngx_int_t                 ident;

    ngx_rbtree_node_t         node;

#if (NGX_HAVE_INET6)
    /* PTR: IPv6 address to resolve (IPv4 address is in rbtree node key) */
    struct in6_addr           addr6;
#endif

    u_short                   nlen;
    u_short                   qlen;

    u_char                   *query;
#if (NGX_HAVE_INET6)
    u_char                   *query6;
#endif

    union {
        in_addr_t             addr;
        in_addr_t            *addrs;
        u_char               *cname;
    } u;

    u_char                    code;
    u_short                   naddrs;
    u_short                   cnlen;

#if (NGX_HAVE_INET6)
```

```
    union {
        struct in6_addr       addr6;
        struct in6_addr      *addrs6;
    } u6;

    u_short                   naddrs6;
#endif

    time_t                    expire;
    time_t                    valid;
    uint32_t                  ttl;

     ngx_resolver_ctx_t       *waiting;
} ngx_resolver_node_t;


typedef struct {
    /* has to be pointer because of "incomplete type" */
    ngx_event_t              *event;
    void                     *dummy;
    ngx_log_t                *log;

    /* event ident must be after 3 pointers as in ngx_connection_t */
    ngx_int_t                 ident;

    /* simple round robin DNS peers balancer */
    ngx_array_t               udp_connections;
    ngx_uint_t                last_connection;

    ngx_rbtree_t              name_rbtree;
    ngx_rbtree_node_t         name_sentinel;

    ngx_rbtree_t              addr_rbtree;
    ngx_rbtree_node_t         addr_sentinel;

    ngx_queue_t               name_resend_queue;
    ngx_queue_t               addr_resend_queue;

    ngx_queue_t               name_expire_queue;
    ngx_queue_t               addr_expire_queue;

#if (NGX_HAVE_INET6)
    ngx_uint_t                ipv6;                /* unsigned  ipv6:1; */
    ngx_rbtree_t              addr6_rbtree;
    ngx_rbtree_node_t         addr6_sentinel;
    ngx_queue_t               addr6_resend_queue;
    ngx_queue_t               addr6_expire_queue;
#endif

    time_t                    resend_timeout;
    time_t                    expire;
    time_t                    valid;

    ngx_uint_t                log_level;
} ngx_resolver_t;


struct ngx_resolver_ctx_s {
    ngx_resolver_ctx_t       *next;
    ngx_resolver_t           *resolver;
    ngx_udp_connection_t     *udp_connection;

    ngx_int_t                 state;
    ngx_str_t                 name;

    ngx_uint_t                naddrs;
    ngx_addr_t               *addrs;
    ngx_addr_t                addr;
    struct sockaddr_in        sin;

    ngx_resolver_handler_pt   handler;
    void                     *data;
    ngx_msec_t                timeout;

    ngx_uint_t                quick;  /* unsigned  quick:1; */
    ngx_uint_t                recursion;
```

```
    ngx_event_t              *event;
};


ngx_resolver_t *ngx_resolver_create(ngx_conf_t *cf, ngx_str_t *names,
    ngx_uint_t n);
ngx_resolver_ctx_t *ngx_resolve_start(ngx_resolver_t *r,
    ngx_resolver_ctx_t *temp);
ngx_int_t ngx_resolve_name(ngx_resolver_ctx_t *ctx);
void ngx_resolve_name_done(ngx_resolver_ctx_t *ctx);
ngx_int_t ngx_resolve_addr(ngx_resolver_ctx_t *ctx);
void ngx_resolve_addr_done(ngx_resolver_ctx_t *ctx);
char *ngx_resolver_strerror(ngx_int_t err);


#endif /* _NGX_RESOLVER_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_resolver.c - nginx-1.7.10

Data types defined

- ngx_resolver_an_t
- ngx_resolver_hdr_t
- ngx_resolver_qs_t

Functions defined

- ngx_resolve_addr
- ngx_resolve_addr_done
- ngx_resolve_name
- ngx_resolve_name_done
- ngx_resolve_name_locked
- ngx_resolve_start
- ngx_resolver_alloc
- ngx_resolver_calloc
- ngx_resolver_cleanup
- ngx_resolver_cleanup_tree
- ngx_resolver_copy
- ngx_resolver_create
- ngx_resolver_create_addr_query
- ngx_resolver_create_name_query
- ngx_resolver_dup
- ngx_resolver_expire
- ngx_resolver_export
- ngx_resolver_free
- ngx_resolver_free_locked
- ngx_resolver_free_node
- ngx_resolver_log_error
- ngx_resolver_lookup_addr
- ngx_resolver_lookup_addr6
- ngx_resolver_lookup_name
- ngx_resolver_process_a

- ngx_resolver_process_ptr
- ngx_resolver_process_response
- ngx_resolver_rbtree_insert_addr6_value
- ngx_resolver_rbtree_insert_value
- ngx_resolver_read_response
- ngx_resolver_resend
- ngx_resolver_resend_handler
- ngx_resolver_send_query
- ngx_resolver_strerror
- ngx_resolver_timeout_handler
- ngx_udp_connect

Macros defined

- NGX_RESOLVER_UDP_SIZE
- ngx_resolver_node

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #define NGX_RESOLVER_UDP_SIZE   4096
14
15
16  typedef struct {
17      u_char  ident_hi;
18      u_char  ident_lo;
19      u_char  flags_hi;
20      u_char  flags_lo;
21      u_char  nqs_hi;
22      u_char  nqs_lo;
23      u_char  nan_hi;
24      u_char  nan_lo;
25      u_char  nns_hi;
26      u_char  nns_lo;
27      u_char  nar_hi;
28      u_char  nar_lo;
29  } ngx_resolver_hdr_t;
30
31
32  typedef struct {
33      u_char  type_hi;
34      u_char  type_lo;
35      u_char  class_hi;
36      u_char  class_lo;
37  } ngx_resolver_qs_t;
38
39
```

```
typedef struct {
    u_char  type_hi;
    u_char  type_lo;
    u_char  class_hi;
    u_char  class_lo;
    u_char  ttl[4];
    u_char  len_hi;
    u_char  len_lo;
} ngx_resolver_an_t;


#define ngx_resolver_node(n)                                                 \
    (ngx_resolver_node_t *)                                                  \
        ((u_char *) (n) - offsetof(ngx_resolver_node_t, node))


ngx_int_t ngx_udp_connect(ngx_udp_connection_t *uc);


static void ngx_resolver_cleanup(void *data);
static void ngx_resolver_cleanup_tree(ngx_resolver_t *r, ngx_rbtree_t *tree);
static ngx_int_t ngx_resolve_name_locked(ngx_resolver_t *r,
    ngx_resolver_ctx_t *ctx);
static void ngx_resolver_expire(ngx_resolver_t *r, ngx_rbtree_t *tree,
    ngx_queue_t *queue);
static ngx_int_t ngx_resolver_send_query(ngx_resolver_t *r,
    ngx_resolver_node_t *rn);
static ngx_int_t ngx_resolver_create_name_query(ngx_resolver_node_t *rn,
    ngx_resolver_ctx_t *ctx);
static ngx_int_t ngx_resolver_create_addr_query(ngx_resolver_node_t *rn,
    ngx_resolver_ctx_t *ctx);
static void ngx_resolver_resend_handler(ngx_event_t *ev);
static time_t ngx_resolver_resend(ngx_resolver_t *r, ngx_rbtree_t *tree,
    ngx_queue_t *queue);
static void ngx_resolver_read_response(ngx_event_t *rev);
static void ngx_resolver_process_response(ngx_resolver_t *r, u_char *buf,
    size_t n);
static void ngx_resolver_process_a(ngx_resolver_t *r, u_char *buf, size_t n,
    ngx_uint_t ident, ngx_uint_t code, ngx_uint_t qtype,
    ngx_uint_t nan, ngx_uint_t ans);
static void ngx_resolver_process_ptr(ngx_resolver_t *r, u_char *buf, size_t n,
    ngx_uint_t ident, ngx_uint_t code, ngx_uint_t nan);
static ngx_resolver_node_t *ngx_resolver_lookup_name(ngx_resolver_t *r,
    ngx_str_t *name, uint32_t hash);
static ngx_resolver_node_t *ngx_resolver_lookup_addr(ngx_resolver_t *r,
    in_addr_t addr);
static void ngx_resolver_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
static ngx_int_t ngx_resolver_copy(ngx_resolver_t *r, ngx_str_t *name,
    u_char *buf, u_char *src, u_char *last);
static void ngx_resolver_timeout_handler(ngx_event_t *ev);
static void ngx_resolver_free_node(ngx_resolver_t *r, ngx_resolver_node_t *rn);
static void *ngx_resolver_alloc(ngx_resolver_t *r, size_t size);
static void *ngx_resolver_calloc(ngx_resolver_t *r, size_t size);
static void ngx_resolver_free(ngx_resolver_t *r, void *p);
static void ngx_resolver_free_locked(ngx_resolver_t *r, void *p);
static void *ngx_resolver_dup(ngx_resolver_t *r, void *src, size_t size);
static ngx_addr_t *ngx_resolver_export(ngx_resolver_t *r,
    ngx_resolver_node_t *rn, ngx_uint_t rotate);
static u_char *ngx_resolver_log_error(ngx_log_t *log, u_char *buf, size_t len);

#if (NGX_HAVE_INET6)
static void ngx_resolver_rbtree_insert_addr6_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
static ngx_resolver_node_t *ngx_resolver_lookup_addr6(ngx_resolver_t *r,
    struct in6_addr *addr, uint32_t hash);
#endif


ngx_resolver_t *
ngx_resolver_create(ngx_conf_t *cf, ngx_str_t *names, ngx_uint_t n)
{
    ngx_str_t              s;
    ngx_url_t              u;
    ngx_uint_t             i, j;
    ngx_resolver_t        *r;
```

```
    ngx_pool_cleanup_t    *cln;
    ngx_udp_connection_t  *uc;

    cln = ngx_pool_cleanup_add(cf->pool, 0);
    if (cln == NULL) {
        return NULL;
    }

    cln->handler = ngx_resolver_cleanup;

    r = ngx_calloc(sizeof(ngx_resolver_t), cf->log);
    if (r == NULL) {
        return NULL;
    }

    cln->data = r;

    r->event = ngx_calloc(sizeof(ngx_event_t), cf->log);
    if (r->event == NULL) {
        return NULL;
    }

    ngx_rbtree_init(&r->name_rbtree, &r->name_sentinel,
                    ngx_resolver_rbtree_insert_value);

    ngx_rbtree_init(&r->addr_rbtree, &r->addr_sentinel,
                    ngx_rbtree_insert_value);

    ngx_queue_init(&r->name_resend_queue);
    ngx_queue_init(&r->addr_resend_queue);

    ngx_queue_init(&r->name_expire_queue);
    ngx_queue_init(&r->addr_expire_queue);

#if (NGX_HAVE_INET6)
    r->ipv6 = 1;

    ngx_rbtree_init(&r->addr6_rbtree, &r->addr6_sentinel,
                    ngx_resolver_rbtree_insert_addr6_value);

    ngx_queue_init(&r->addr6_resend_queue);

    ngx_queue_init(&r->addr6_expire_queue);
#endif

    r->event->handler = ngx_resolver_resend_handler;
    r->event->data = r;
    r->event->log = &cf->cycle->new_log;
    r->ident = -1;

    r->resend_timeout = 5;
    r->expire = 30;
    r->valid = 0;

    r->log = &cf->cycle->new_log;
    r->log_level = NGX_LOG_ERR;

    if (n) {
        if (ngx_array_init(&r->udp_connections, cf->pool, n,
                           sizeof(ngx_udp_connection_t))
            != NGX_OK)
        {
            return NULL;
        }
    }

    for (i = 0; i < n; i++) {
        if (ngx_strncmp(names[i].data, "valid=", 6) == 0) {
            s.len = names[i].len - 6;
            s.data = names[i].data + 6;

            r->valid = ngx_parse_time(&s, 1);

            if (r->valid == (time_t) NGX_ERROR) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid parameter: %V", &names[i]);
```

```
                return NULL;
            }

            continue;
        }

#if (NGX_HAVE_INET6)
        if (ngx_strncmp(names[i].data, "ipv6=", 5) == 0) {

            if (ngx_strcmp(&names[i].data[5], "on") == 0) {
                r->ipv6 = 1;

            } else if (ngx_strcmp(&names[i].data[5], "off") == 0) {
                r->ipv6 = 0;

            } else {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid parameter: %V", &names[i]);
                return NULL;
            }

            continue;
        }
#endif

        ngx_memzero(&u, sizeof(ngx_url_t));

        u.url = names[i];
        u.default_port = 53;

        if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
            if (u.err) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "%s in resolver \"%V\"",
                                   u.err, &u.url);
            }

            return NULL;
        }

        uc = ngx_array_push_n(&r->udp_connections, u.naddrs);
        if (uc == NULL) {
            return NULL;
        }

        ngx_memzero(uc, u.naddrs * sizeof(ngx_udp_connection_t));

        for (j = 0; j < u.naddrs; j++) {
            uc[j].sockaddr = u.addrs[j].sockaddr;
            uc[j].socklen = u.addrs[j].socklen;
            uc[j].server = u.addrs[j].name;
        }
    }

    return r;
}


static void
ngx_resolver_cleanup(void *data)
{
    ngx_resolver_t  *r = data;

    ngx_uint_t             i;
    ngx_udp_connection_t  *uc;

    if (r) {
        ngx_log_debug0(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
                       "cleanup resolver");

        ngx_resolver_cleanup_tree(r, &r->name_rbtree);

        ngx_resolver_cleanup_tree(r, &r->addr_rbtree);

#if (NGX_HAVE_INET6)
        ngx_resolver_cleanup_tree(r, &r->addr6_rbtree);
```

```
#endif

        if (r->event) {
            ngx_free(r->event);
        }


        uc = r->udp_connections.elts;

        for (i = 0; i < r->udp_connections.nelts; i++) {
            if (uc[i].connection) {
                ngx_close_connection(uc[i].connection);
            }
        }

        ngx_free(r);
    }
}


static void
ngx_resolver_cleanup_tree(ngx_resolver_t *r, ngx_rbtree_t *tree)
{
    ngx_resolver_ctx_t   *ctx, *next;
    ngx_resolver_node_t  *rn;

    while (tree->root != tree->sentinel) {

        rn = ngx_resolver_node(ngx_rbtree_min(tree->root, tree->sentinel));

        ngx_queue_remove(&rn->queue);

        for (ctx = rn->waiting; ctx; ctx = next) {
            next = ctx->next;

            if (ctx->event) {
                ngx_resolver_free(r, ctx->event);
            }

            ngx_resolver_free(r, ctx);
        }

        ngx_rbtree_delete(tree, &rn->node);

        ngx_resolver_free_node(r, rn);
    }
}


ngx_resolver_ctx_t *
ngx_resolve_start(ngx_resolver_t *r, ngx_resolver_ctx_t *temp)
{
    in_addr_t            addr;
    ngx_resolver_ctx_t  *ctx;

    if (temp) {
        addr = ngx_inet_addr(temp->name.data, temp->name.len);

        if (addr != INADDR_NONE) {
            temp->resolver = r;
            temp->state = NGX_OK;
            temp->naddrs = 1;
            temp->addrs = &temp->addr;
            temp->addr.sockaddr = (struct sockaddr *) &temp->sin;
            temp->addr.socklen = sizeof(struct sockaddr_in);
            ngx_memzero(&temp->sin, sizeof(struct sockaddr_in));
            temp->sin.sin_family = AF_INET;
            temp->sin.sin_addr.s_addr = addr;
            temp->quick = 1;

            return temp;
        }
    }

    if (r->udp_connections.nelts == 0) {
        return NGX_NO_RESOLVER;
```

```
344     }
345
346     ctx = ngx_resolver_calloc(r, sizeof(ngx_resolver_ctx_t));
347
348     if (ctx) {
349         ctx->resolver = r;
350     }
351
352     return ctx;
353 }
354
355
356 ngx_int_t
357 ngx_resolve_name(ngx_resolver_ctx_t *ctx)
358 {
359     ngx_int_t        rc;
360     ngx_resolver_t  *r;
361
362     r = ctx->resolver;
363
364     if (ctx->name.len > 0 && ctx->name.data[ctx->name.len - 1] == '.') {
365         ctx->name.len--;
366     }
367
368     ngx_log_debug1(NGX_LOG_DEBUG_CORE, r->log, 0,
369                    "resolve: \"%V\"", &ctx->name);
370
371     if (ctx->quick) {
372         ctx->handler(ctx);
373         return NGX_OK;
374     }
375
376     /* lock name mutex */
377
378     rc = ngx_resolve_name_locked(r, ctx);
379
380     if (rc == NGX_OK) {
381         return NGX_OK;
382     }
383
384     /* unlock name mutex */
385
386     if (rc == NGX_AGAIN) {
387         return NGX_OK;
388     }
389
390     /* NGX_ERROR */
391
392     if (ctx->event) {
393         ngx_resolver_free(r, ctx->event);
394     }
395
396     ngx_resolver_free(r, ctx);
397
398     return NGX_ERROR;
399 }
400
401
402 void
403 ngx_resolve_name_done(ngx_resolver_ctx_t *ctx)
404 {
405     uint32_t              hash;
406     ngx_resolver_t       *r;
407     ngx_resolver_ctx_t   *w, **p;
408     ngx_resolver_node_t  *rn;
409
410     r = ctx->resolver;
411
412     ngx_log_debug1(NGX_LOG_DEBUG_CORE, r->log, 0,
413                    "resolve name done: %i", ctx->state);
414
415     if (ctx->quick) {
416         return;
417     }
418
419     if (ctx->event && ctx->event->timer_set) {
```

```
420         ngx_del_timer(ctx->event);
421     }
422
423     /* lock name mutex */
424
425     if (ctx->state == NGX_AGAIN) {
426
427         hash = ngx_crc32_short(ctx->name.data, ctx->name.len);
428
429         rn = ngx_resolver_lookup_name(r, &ctx->name, hash);
430
431         if (rn) {
432             p = &rn->waiting;
433             w = rn->waiting;
434
435             while (w) {
436                 if (w == ctx) {
437                     *p = w->next;
438
439                     goto done;
440                 }
441
442                 p = &w->next;
443                 w = w->next;
444             }
445         }
446
447         ngx_log_error(NGX_LOG_ALERT, r->log, 0,
448                       "could not cancel %V resolving", &ctx->name);
449     }
450
451 done:
452
453     ngx_resolver_expire(r, &r->name_rbtree, &r->name_expire_queue);
454
455     /* unlock name mutex */
456
457     /* lock alloc mutex */
458
459     if (ctx->event) {
460         ngx_resolver_free_locked(r, ctx->event);
461     }
462
463     ngx_resolver_free_locked(r, ctx);
464
465     /* unlock alloc mutex */
466 }
467
468
469 static ngx_int_t
470 ngx_resolve_name_locked(ngx_resolver_t *r, ngx_resolver_ctx_t *ctx)
471 {
472     uint32_t              hash;
473     ngx_int_t             rc;
474     ngx_uint_t            naddrs;
475     ngx_addr_t           *addrs;
476     ngx_resolver_ctx_t   *next;
477     ngx_resolver_node_t  *rn;
478
479     ngx_strlow(ctx->name.data, ctx->name.data, ctx->name.len);
480
481     hash = ngx_crc32_short(ctx->name.data, ctx->name.len);
482
483     rn = ngx_resolver_lookup_name(r, &ctx->name, hash);
484
485     if (rn) {
486
487         if (rn->valid >= ngx_time()) {
488
489             ngx_log_debug0(NGX_LOG_DEBUG_CORE, r->log, 0, "resolve cached");
490
491             ngx_queue_remove(&rn->queue);
492
493             rn->expire = ngx_time() + r->expire;
494
495             ngx_queue_insert_head(&r->name_expire_queue, &rn->queue);
```

```
            naddrs = (rn->naddrs == (u_short) -1) ? 0 : rn->naddrs;
#if (NGX_HAVE_INET6)
            naddrs += (rn->naddrs6 == (u_short) -1) ? 0 : rn->naddrs6;
#endif

            if (naddrs) {

                if (naddrs == 1 && rn->naddrs == 1) {
                    addrs = NULL;

                } else {
                    addrs = ngx_resolver_export(r, rn, 1);
                    if (addrs == NULL) {
                        return NGX_ERROR;
                    }
                }

                ctx->next = rn->waiting;
                rn->waiting = NULL;

                /* unlock name mutex */

                do {
                    ctx->state = NGX_OK;
                    ctx->naddrs = naddrs;

                    if (addrs == NULL) {
                        ctx->addrs = &ctx->addr;
                        ctx->addr.sockaddr = (struct sockaddr *) &ctx->sin;
                        ctx->addr.socklen = sizeof(struct sockaddr_in);
                        ngx_memzero(&ctx->sin, sizeof(struct sockaddr_in));
                        ctx->sin.sin_family = AF_INET;
                        ctx->sin.sin_addr.s_addr = rn->u.addr;

                    } else {
                        ctx->addrs = addrs;
                    }

                    next = ctx->next;

                    ctx->handler(ctx);

                    ctx = next;
                } while (ctx);

                if (addrs != NULL) {
                    ngx_resolver_free(r, addrs->sockaddr);
                    ngx_resolver_free(r, addrs);
                }

                return NGX_OK;
            }

            /* NGX_RESOLVE_CNAME */

            if (ctx->recursion++ < NGX_RESOLVER_MAX_RECURSION) {

                ctx->name.len = rn->cnlen;
                ctx->name.data = rn->u.cname;

                return ngx_resolve_name_locked(r, ctx);
            }

            ctx->next = rn->waiting;
            rn->waiting = NULL;

            /* unlock name mutex */

            do {
                ctx->state = NGX_RESOLVE_NXDOMAIN;
                next = ctx->next;

                ctx->handler(ctx);

                ctx = next;
```

```
572             } while (ctx);
573
574             return NGX_OK;
575         }
576
577         if (rn->waiting) {
578
579             ctx->next = rn->waiting;
580             rn->waiting = ctx;
581             ctx->state = NGX_AGAIN;
582
583             return NGX_AGAIN;
584         }
585
586         ngx_queue_remove(&rn->queue);
587
588         /* lock alloc mutex */
589
590         if (rn->query) {
591             ngx_resolver_free_locked(r, rn->query);
592             rn->query = NULL;
593 #if (NGX_HAVE_INET6)
594             rn->query6 = NULL;
595 #endif
596         }
597
598         if (rn->cnlen) {
599             ngx_resolver_free_locked(r, rn->u.cname);
600         }
601
602         if (rn->naddrs > 1 && rn->naddrs != (u_short) -1) {
603             ngx_resolver_free_locked(r, rn->u.addrs);
604         }
605
606 #if (NGX_HAVE_INET6)
607         if (rn->naddrs6 > 1 && rn->naddrs6 != (u_short) -1) {
608             ngx_resolver_free_locked(r, rn->u6.addrs6);
609         }
610 #endif
611
612         /* unlock alloc mutex */
613
614     } else {
615
616         rn = ngx_resolver_alloc(r, sizeof(ngx_resolver_node_t));
617         if (rn == NULL) {
618             return NGX_ERROR;
619         }
620
621         rn->name = ngx_resolver_dup(r, ctx->name.data, ctx->name.len);
622         if (rn->name == NULL) {
623             ngx_resolver_free(r, rn);
624             return NGX_ERROR;
625         }
626
627         rn->node.key = hash;
628         rn->nlen = (u_short) ctx->name.len;
629         rn->query = NULL;
630 #if (NGX_HAVE_INET6)
631         rn->query6 = NULL;
632 #endif
633
634         ngx_rbtree_insert(&r->name_rbtree, &rn->node);
635     }
636
637     rc = ngx_resolver_create_name_query(rn, ctx);
638
639     if (rc == NGX_ERROR) {
640         goto failed;
641     }
642
643     if (rc == NGX_DECLINED) {
644         ngx_rbtree_delete(&r->name_rbtree, &rn->node);
645
646         ngx_resolver_free(r, rn->query);
647         ngx_resolver_free(r, rn->name);
```

```
        ngx_resolver_free(r, rn);

        ctx->state = NGX_RESOLVE_NXDOMAIN;
        ctx->handler(ctx);

        return NGX_OK;
    }

    rn->naddrs = (u_short) -1;
#if (NGX_HAVE_INET6)
    rn->naddrs6 = r->ipv6 ? (u_short) -1 : 0;
#endif

    if (ngx_resolver_send_query(r, rn) != NGX_OK) {
        goto failed;
    }

    if (ctx->event == NULL) {
        ctx->event = ngx_resolver_calloc(r, sizeof(ngx_event_t));
        if (ctx->event == NULL) {
            goto failed;
        }

        ctx->event->handler = ngx_resolver_timeout_handler;
        ctx->event->data = rn;
        ctx->event->log = r->log;
        rn->ident = -1;

        ngx_add_timer(ctx->event, ctx->timeout);
    }

    if (ngx_queue_empty(&r->name_resend_queue)) {
        ngx_add_timer(r->event, (ngx_msec_t) (r->resend_timeout * 1000));
    }

    rn->expire = ngx_time() + r->resend_timeout;

    ngx_queue_insert_head(&r->name_resend_queue, &rn->queue);

    rn->code = 0;
    rn->cnlen = 0;
    rn->valid = 0;
    rn->ttl = NGX_MAX_UINT32_VALUE;
    rn->waiting = ctx;

    ctx->state = NGX_AGAIN;

    return NGX_AGAIN;

failed:

    ngx_rbtree_delete(&r->name_rbtree, &rn->node);

    if (rn->query) {
        ngx_resolver_free(r, rn->query);
    }

    ngx_resolver_free(r, rn->name);

    ngx_resolver_free(r, rn);

    return NGX_ERROR;
}


ngx_int_t
ngx_resolve_addr(ngx_resolver_ctx_t *ctx)
{
    u_char               *name;
    in_addr_t             addr;
    ngx_queue_t          *resend_queue, *expire_queue;
    ngx_rbtree_t         *tree;
    ngx_resolver_t       *r;
    struct sockaddr_in   *sin;
    ngx_resolver_node_t  *rn;
#if (NGX_HAVE_INET6)
```

```
    uint32_t              hash;
    struct sockaddr_in6  *sin6;
#endif

#if (NGX_SUPPRESS_WARN)
    addr = 0;
#if (NGX_HAVE_INET6)
    hash = 0;
    sin6 = NULL;
#endif
#endif

    r = ctx->resolver;

    switch (ctx->addr.sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) ctx->addr.sockaddr;
        hash = ngx_crc32_short(sin6->sin6_addr.s6_addr, 16);

        /* lock addr mutex */

        rn = ngx_resolver_lookup_addr6(r, &sin6->sin6_addr, hash);

        tree = &r->addr6_rbtree;
        resend_queue = &r->addr6_resend_queue;
        expire_queue = &r->addr6_expire_queue;

        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) ctx->addr.sockaddr;
        addr = ntohl(sin->sin_addr.s_addr);

        /* lock addr mutex */

        rn = ngx_resolver_lookup_addr(r, addr);

        tree = &r->addr_rbtree;
        resend_queue = &r->addr_resend_queue;
        expire_queue = &r->addr_expire_queue;
    }

    if (rn) {

        if (rn->valid >= ngx_time()) {

            ngx_log_debug0(NGX_LOG_DEBUG_CORE, r->log, 0, "resolve cached");

            ngx_queue_remove(&rn->queue);

            rn->expire = ngx_time() + r->expire;

            ngx_queue_insert_head(expire_queue, &rn->queue);

            name = ngx_resolver_dup(r, rn->name, rn->nlen);
            if (name == NULL) {
                goto failed;
            }

            ctx->name.len = rn->nlen;
            ctx->name.data = name;

            /* unlock addr mutex */

            ctx->state = NGX_OK;

            ctx->handler(ctx);

            ngx_resolver_free(r, name);

            return NGX_OK;
        }

```

```
        if (rn->waiting) {

            ctx->next = rn->waiting;
            rn->waiting = ctx;
            ctx->state = NGX_AGAIN;

            /* unlock addr mutex */

            return NGX_OK;
        }

        ngx_queue_remove(&rn->queue);

        ngx_resolver_free(r, rn->query);
        rn->query = NULL;
#if (NGX_HAVE_INET6)
        rn->query6 = NULL;
#endif

    } else {
        rn = ngx_resolver_alloc(r, sizeof(ngx_resolver_node_t));
        if (rn == NULL) {
            goto failed;
        }

        switch (ctx->addr.sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            rn->addr6 = sin6->sin6_addr;
            rn->node.key = hash;
            break;
#endif

        default: /* AF_INET */
            rn->node.key = addr;
        }

        rn->query = NULL;
#if (NGX_HAVE_INET6)
        rn->query6 = NULL;
#endif

        ngx_rbtree_insert(tree, &rn->node);
    }

    if (ngx_resolver_create_addr_query(rn, ctx) != NGX_OK) {
        goto failed;
    }

    rn->naddrs = (u_short) -1;
#if (NGX_HAVE_INET6)
    rn->naddrs6 = (u_short) -1;
#endif

    if (ngx_resolver_send_query(r, rn) != NGX_OK) {
        goto failed;
    }

    ctx->event = ngx_resolver_calloc(r, sizeof(ngx_event_t));
    if (ctx->event == NULL) {
        goto failed;
    }

    ctx->event->handler = ngx_resolver_timeout_handler;
    ctx->event->data = rn;
    ctx->event->log = r->log;
    rn->ident = -1;

    ngx_add_timer(ctx->event, ctx->timeout);

    if (ngx_queue_empty(resend_queue)) {
        ngx_add_timer(r->event, (ngx_msec_t) (r->resend_timeout * 1000));
    }

    rn->expire = ngx_time() + r->resend_timeout;
```

```
876
877     ngx_queue_insert_head(resend_queue, &rn->queue);
878
879     rn->code = 0;
880     rn->cnlen = 0;
881     rn->name = NULL;
882     rn->nlen = 0;
883     rn->valid = 0;
884     rn->ttl = NGX_MAX_UINT32_VALUE;
885     rn->waiting = ctx;
886
887     /* unlock addr mutex */
888
889     ctx->state = NGX_AGAIN;
890
891     return NGX_OK;
892
893 failed:
894
895     if (rn) {
896         ngx_rbtree_delete(tree, &rn->node);
897
898         if (rn->query) {
899             ngx_resolver_free(r, rn->query);
900         }
901
902         ngx_resolver_free(r, rn);
903     }
904
905     /* unlock addr mutex */
906
907     if (ctx->event) {
908         ngx_resolver_free(r, ctx->event);
909     }
910
911     ngx_resolver_free(r, ctx);
912
913     return NGX_ERROR;
914 }
915
916
917 void
918 ngx_resolve_addr_done(ngx_resolver_ctx_t *ctx)
919 {
920     in_addr_t             addr;
921     ngx_queue_t          *expire_queue;
922     ngx_rbtree_t         *tree;
923     ngx_resolver_t       *r;
924     ngx_resolver_ctx_t   *w, **p;
925     struct sockaddr_in   *sin;
926     ngx_resolver_node_t  *rn;
927 #if (NGX_HAVE_INET6)
928     uint32_t              hash;
929     struct sockaddr_in6  *sin6;
930 #endif
931
932     r = ctx->resolver;
933
934     switch (ctx->addr.sockaddr->sa_family) {
935
936 #if (NGX_HAVE_INET6)
937     case AF_INET6:
938         tree = &r->addr6_rbtree;
939         expire_queue = &r->addr6_expire_queue;
940         break;
941 #endif
942
943     default: /* AF_INET */
944         tree = &r->addr_rbtree;
945         expire_queue = &r->addr_expire_queue;
946     }
947
948     ngx_log_debug1(NGX_LOG_DEBUG_CORE, r->log, 0,
949                    "resolve addr done: %i", ctx->state);
950
951     if (ctx->event && ctx->event->timer_set) {
```

```
952         ngx_del_timer(ctx->event);
953     }
954
955     /* lock addr mutex */
956
957     if (ctx->state == NGX_AGAIN) {
958
959         switch (ctx->addr.sockaddr->sa_family) {
960
961 #if (NGX_HAVE_INET6)
962         case AF_INET6:
963             sin6 = (struct sockaddr_in6 *) ctx->addr.sockaddr;
964             hash = ngx_crc32_short(sin6->sin6_addr.s6_addr, 16);
965             rn = ngx_resolver_lookup_addr6(r, &sin6->sin6_addr, hash);
966             break;
967 #endif
968
969         default: /* AF_INET */
970             sin = (struct sockaddr_in *) ctx->addr.sockaddr;
971             addr = ntohl(sin->sin_addr.s_addr);
972             rn = ngx_resolver_lookup_addr(r, addr);
973         }
974
975         if (rn) {
976             p = &rn->waiting;
977             w = rn->waiting;
978
979             while (w) {
980                 if (w == ctx) {
981                     *p = w->next;
982
983                     goto done;
984                 }
985
986                 p = &w->next;
987                 w = w->next;
988             }
989         }
990
991         {
992             u_char     text[NGX_SOCKADDR_STRLEN];
993             ngx_str_t  addrtext;
994
995             addrtext.data = text;
996             addrtext.len = ngx_sock_ntop(ctx->addr.sockaddr, ctx->addr.socklen,
997                                          text, NGX_SOCKADDR_STRLEN, 0);
998
999             ngx_log_error(NGX_LOG_ALERT, r->log, 0,
1000                           "could not cancel %V resolving", &addrtext);
1001         }
1002     }
1003
1004 done:
1005
1006     ngx_resolver_expire(r, tree, expire_queue);
1007
1008     /* unlock addr mutex */
1009
1010     /* lock alloc mutex */
1011
1012     if (ctx->event) {
1013         ngx_resolver_free_locked(r, ctx->event);
1014     }
1015
1016     ngx_resolver_free_locked(r, ctx);
1017
1018     /* unlock alloc mutex */
1019 }
1020
1021
1022 static void
1023 ngx_resolver_expire(ngx_resolver_t *r, ngx_rbtree_t *tree, ngx_queue_t *queue)
1024 {
1025     time_t                now;
1026     ngx_uint_t            i;
1027     ngx_queue_t          *q;
```

```
    ngx_resolver_node_t  *rn;

    ngx_log_debug0(NGX_LOG_DEBUG_CORE, r->log, 0, "resolver expire");

    now = ngx_time();

    for (i = 0; i < 2; i++) {
        if (ngx_queue_empty(queue)) {
            return;
        }

        q = ngx_queue_last(queue);

        rn = ngx_queue_data(q, ngx_resolver_node_t, queue);

        if (now <= rn->expire) {
            return;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_CORE, r->log, 0,
                       "resolver expire \"%*s\"", (size_t) rn->nlen, rn->name);

        ngx_queue_remove(q);

        ngx_rbtree_delete(tree, &rn->node);

        ngx_resolver_free_node(r, rn);
    }
}


static ngx_int_t
ngx_resolver_send_query(ngx_resolver_t *r, ngx_resolver_node_t *rn)
{
    ssize_t                n;
    ngx_udp_connection_t  *uc;

    uc = r->udp_connections.elts;

    uc = &uc[r->last_connection++];
    if (r->last_connection == r->udp_connections.nelts) {
        r->last_connection = 0;
    }

    if (uc->connection == NULL) {

        uc->log = *r->log;
        uc->log.handler = ngx_resolver_log_error;
        uc->log.data = uc;
        uc->log.action = "resolving";

        if (ngx_udp_connect(uc) != NGX_OK) {
            return NGX_ERROR;
        }

        uc->connection->data = r;
        uc->connection->read->handler = ngx_resolver_read_response;
        uc->connection->read->resolver = 1;
    }

    if (rn->naddrs == (u_short) -1) {
        n = ngx_send(uc->connection, rn->query, rn->qlen);

        if (n == -1) {
            return NGX_ERROR;
        }

        if ((size_t) n != (size_t) rn->qlen) {
            ngx_log_error(NGX_LOG_CRIT, &uc->log, 0, "send() incomplete");
            return NGX_ERROR;
        }
    }

#if (NGX_HAVE_INET6)
    if (rn->query6 && rn->naddrs6 == (u_short) -1) {
        n = ngx_send(uc->connection, rn->query6, rn->qlen);
```

```
1104
1105            if (n == -1) {
1106                return NGX_ERROR;
1107            }
1108
1109            if ((size_t) n != (size_t) rn->qlen) {
1110                ngx_log_error(NGX_LOG_CRIT, &uc->log, 0, "send() incomplete");
1111                return NGX_ERROR;
1112            }
1113        }
1114    #endif
1115
1116        return NGX_OK;
1117    }
1118
1119
1120    static void
1121    ngx_resolver_resend_handler(ngx_event_t *ev)
1122    {
1123        time_t           timer, atimer, ntimer;
1124    #if (NGX_HAVE_INET6)
1125        time_t           a6timer;
1126    #endif
1127        ngx_resolver_t  *r;
1128
1129        r = ev->data;
1130
1131        ngx_log_debug0(NGX_LOG_DEBUG_CORE, r->log, 0,
1132                       "resolver resend handler");
1133
1134        /* lock name mutex */
1135
1136        ntimer = ngx_resolver_resend(r, &r->name_rbtree, &r->name_resend_queue);
1137
1138        /* unlock name mutex */
1139
1140        /* lock addr mutex */
1141
1142        atimer = ngx_resolver_resend(r, &r->addr_rbtree, &r->addr_resend_queue);
1143
1144        /* unlock addr mutex */
1145
1146    #if (NGX_HAVE_INET6)
1147
1148        /* lock addr6 mutex */
1149
1150        a6timer = ngx_resolver_resend(r, &r->addr6_rbtree, &r->addr6_resend_queue);
1151
1152        /* unlock addr6 mutex */
1153
1154    #endif
1155
1156        timer = ntimer;
1157
1158        if (timer == 0) {
1159            timer = atimer;
1160
1161        } else if (atimer) {
1162            timer = ngx_min(timer, atimer);
1163        }
1164
1165    #if (NGX_HAVE_INET6)
1166
1167        if (timer == 0) {
1168            timer = a6timer;
1169
1170        } else if (a6timer) {
1171            timer = ngx_min(timer, a6timer);
1172        }
1173
1174    #endif
1175
1176        if (timer) {
1177            ngx_add_timer(r->event, (ngx_msec_t) (timer * 1000));
1178        }
1179    }
```

```
1180
1181
1182  static time_t
1183  ngx_resolver_resend(ngx_resolver_t *r, ngx_rbtree_t *tree, ngx_queue_t *queue)
1184  {
1185      time_t                now;
1186      ngx_queue_t          *q;
1187      ngx_resolver_node_t  *rn;
1188
1189      now = ngx_time();
1190
1191      for ( ;; ) {
1192          if (ngx_queue_empty(queue)) {
1193              return 0;
1194          }
1195
1196          q = ngx_queue_last(queue);
1197
1198          rn = ngx_queue_data(q, ngx_resolver_node_t, queue);
1199
1200          if (now < rn->expire) {
1201              return rn->expire - now;
1202          }
1203
1204          ngx_log_debug3(NGX_LOG_DEBUG_CORE, r->log, 0,
1205                         "resolver resend \"%*s\" %p",
1206                         (size_t) rn->nlen, rn->name, rn->waiting);
1207
1208          ngx_queue_remove(q);
1209
1210          if (rn->waiting) {
1211
1212              (void) ngx_resolver_send_query(r, rn);
1213
1214              rn->expire = now + r->resend_timeout;
1215
1216              ngx_queue_insert_head(queue, q);
1217
1218              continue;
1219          }
1220
1221          ngx_rbtree_delete(tree, &rn->node);
1222
1223          ngx_resolver_free_node(r, rn);
1224      }
1225  }
1226
1227
1228  static void
1229  ngx_resolver_read_response(ngx_event_t *rev)
1230  {
1231      ssize_t            n;
1232      ngx_connection_t  *c;
1233      u_char             buf[NGX_RESOLVER_UDP_SIZE];
1234
1235      c = rev->data;
1236
1237      do {
1238          n = ngx_udp_recv(c, buf, NGX_RESOLVER_UDP_SIZE);
1239
1240          if (n < 0) {
1241              return;
1242          }
1243
1244          ngx_resolver_process_response(c->data, buf, n);
1245
1246      } while (rev->ready);
1247  }
1248
1249
1250  static void
1251  ngx_resolver_process_response(ngx_resolver_t *r, u_char *buf, size_t n)
1252  {
1253      char                *err;
1254      ngx_uint_t           i, times, ident, qident, flags, code, nqs, nan,
1255                           qtype, qclass;
```

```
#if (NGX_HAVE_INET6)
    ngx_uint_t            qident6;
#endif
    ngx_queue_t          *q;
    ngx_resolver_qs_t    *qs;
    ngx_resolver_hdr_t   *response;
    ngx_resolver_node_t  *rn;

    if (n < sizeof(ngx_resolver_hdr_t)) {
        goto short_response;
    }

    response = (ngx_resolver_hdr_t *) buf;

    ident = (response->ident_hi << 8) + response->ident_lo;
    flags = (response->flags_hi << 8) + response->flags_lo;
    nqs = (response->nqs_hi << 8) + response->nqs_lo;
    nan = (response->nan_hi << 8) + response->nan_lo;

    ngx_log_debug6(NGX_LOG_DEBUG_CORE, r->log, 0,
                   "resolver DNS response %ui fl:%04Xui %ui/%ui/%ud/%ud",
                   ident, flags, nqs, nan,
                   (response->nns_hi << 8) + response->nns_lo,
                   (response->nar_hi << 8) + response->nar_lo);

    /* response to a standard query */
    if ((flags & 0xf870) != 0x8000) {
        ngx_log_error(r->log_level, r->log, 0,
                      "invalid DNS response %ui fl:%04Xui", ident, flags);
        return;
    }

    code = flags & 0xf;

    if (code == NGX_RESOLVE_FORMERR) {

        times = 0;

        for (q = ngx_queue_head(&r->name_resend_queue);
             q != ngx_queue_sentinel(&r->name_resend_queue) || times++ < 100;
             q = ngx_queue_next(q))
        {
            rn = ngx_queue_data(q, ngx_resolver_node_t, queue);
            qident = (rn->query[0] << 8) + rn->query[1];

            if (qident == ident) {
                goto dns_error_name;
            }

#if (NGX_HAVE_INET6)
            if (rn->query6) {
                qident6 = (rn->query6[0] << 8) + rn->query6[1];

                if (qident6 == ident) {
                    goto dns_error_name;
                }
            }
#endif
        }

        goto dns_error;
    }

    if (code > NGX_RESOLVE_REFUSED) {
        goto dns_error;
    }

    if (nqs != 1) {
        err = "invalid number of questions in DNS response";
        goto done;
    }

    i = sizeof(ngx_resolver_hdr_t);

    while (i < (ngx_uint_t) n) {
        if (buf[i] == '\0') {
```

```
            goto found;
        }

        i += 1 + buf[i];
    }

    goto short_response;

found:

    if (i++ == sizeof(ngx_resolver_hdr_t)) {
        err = "zero-length domain name in DNS response";
        goto done;
    }

    if (i + sizeof(ngx_resolver_qs_t) + nan * (2 + sizeof(ngx_resolver_an_t))
        > (ngx_uint_t) n)
    {
        goto short_response;
    }

    qs = (ngx_resolver_qs_t *) &buf[i];

    qtype = (qs->type_hi << 8) + qs->type_lo;
    qclass = (qs->class_hi << 8) + qs->class_lo;

    ngx_log_debug2(NGX_LOG_DEBUG_CORE, r->log, 0,
                   "resolver DNS response qt:%ui cl:%ui", qtype, qclass);

    if (qclass != 1) {
        ngx_log_error(r->log_level, r->log, 0,
                      "unknown query class %ui in DNS response", qclass);
        return;
    }

    switch (qtype) {

    case NGX_RESOLVE_A:
#if (NGX_HAVE_INET6)
    case NGX_RESOLVE_AAAA:
#endif

        ngx_resolver_process_a(r, buf, n, ident, code, qtype, nan,
                               i + sizeof(ngx_resolver_qs_t));

        break;

    case NGX_RESOLVE_PTR:

        ngx_resolver_process_ptr(r, buf, n, ident, code, nan);

        break;

    default:
        ngx_log_error(r->log_level, r->log, 0,
                      "unknown query type %ui in DNS response", qtype);
        return;
    }

    return;

short_response:

    err = "short DNS response";

done:

    ngx_log_error(r->log_level, r->log, 0, err);

    return;

dns_error_name:

    ngx_log_error(r->log_level, r->log, 0,
                  "DNS error (%ui: %s), query id:%ui, name:\"%*s\"",
                  code, ngx_resolver_strerror(code), ident,
```

```
                  rn->nlen, rn->name);
    return;

dns_error:

    ngx_log_error(r->log_level, r->log, 0,
                  "DNS error (%ui: %s), query id:%ui",
                  code, ngx_resolver_strerror(code), ident);
    return;
}


static void
ngx_resolver_process_a(ngx_resolver_t *r, u_char *buf, size_t last,
    ngx_uint_t ident, ngx_uint_t code, ngx_uint_t qtype,
    ngx_uint_t nan, ngx_uint_t ans)
{
    char                *err;
    u_char              *cname;
    size_t               len;
    int32_t              ttl;
    uint32_t             hash;
    in_addr_t           *addr;
    ngx_str_t            name;
    ngx_addr_t          *addrs;
    ngx_uint_t           type, class, qident, naddrs, a, i, n, start;
#if (NGX_HAVE_INET6)
    struct in6_addr      *addr6;
#endif
    ngx_resolver_an_t    *an;
    ngx_resolver_ctx_t   *ctx, *next;
    ngx_resolver_node_t  *rn;

    if (ngx_resolver_copy(r, &name, buf,
                          buf + sizeof(ngx_resolver_hdr_t), buf + last)
        != NGX_OK)
    {
        return;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, r->log, 0, "resolver qs:%V", &name);

    hash = ngx_crc32_short(name.data, name.len);

    /* lock name mutex */

    rn = ngx_resolver_lookup_name(r, &name, hash);

    if (rn == NULL) {
        ngx_log_error(r->log_level, r->log, 0,
                      "unexpected response for %V", &name);
        ngx_resolver_free(r, name.data);
        goto failed;
    }

    switch (qtype) {

#if (NGX_HAVE_INET6)
    case NGX_RESOLVE_AAAA:

        if (rn->query6 == NULL || rn->naddrs6 != (u_short) -1) {
            ngx_log_error(r->log_level, r->log, 0,
                          "unexpected response for %V", &name);
            ngx_resolver_free(r, name.data);
            goto failed;
        }

        qident = (rn->query6[0] << 8) + rn->query6[1];

        break;
#endif

    default: /* NGX_RESOLVE_A */

        if (rn->query == NULL || rn->naddrs != (u_short) -1) {
            ngx_log_error(r->log_level, r->log, 0,
```

```
                          "unexpected response for %V", &name);
            ngx_resolver_free(r, name.data);
            goto failed;
        }

        qident = (rn->query[0] << 8) + rn->query[1];
    }

    if (ident != qident) {
        ngx_log_error(r->log_level, r->log, 0,
                      "wrong ident %ui response for %V, expect %ui",
                      ident, &name, qident);
        ngx_resolver_free(r, name.data);
        goto failed;
    }

    ngx_resolver_free(r, name.data);

    if (code == 0 && rn->code) {
        code = rn->code;
    }

    if (code == 0 && nan == 0) {

#if (NGX_HAVE_INET6)
        switch (qtype) {

        case NGX_RESOLVE_AAAA:

            rn->naddrs6 = 0;

            if (rn->naddrs == (u_short) -1) {
                goto next;
            }

            if (rn->naddrs) {
                goto export;
            }

            break;

        default: /* NGX_RESOLVE_A */

            rn->naddrs = 0;

            if (rn->naddrs6 == (u_short) -1) {
                goto next;
            }

            if (rn->naddrs6) {
                goto export;
            }
        }
#endif

        code = NGX_RESOLVE_NXDOMAIN;
    }

    if (code) {

#if (NGX_HAVE_INET6)
        switch (qtype) {

        case NGX_RESOLVE_AAAA:

            rn->naddrs6 = 0;

            if (rn->naddrs == (u_short) -1) {
                rn->code = (u_char) code;
                goto next;
            }

            break;

        default: /* NGX_RESOLVE_A */

```

```
            rn->naddrs = 0;

            if (rn->naddrs6 == (u_short) -1) {
                rn->code = (u_char) code;
                goto next;
            }
        }
#endif

        next = rn->waiting;
        rn->waiting = NULL;

        ngx_queue_remove(&rn->queue);

        ngx_rbtree_delete(&r->name_rbtree, &rn->node);

        /* unlock name mutex */

        while (next) {
            ctx = next;
            ctx->state = code;
            next = ctx->next;

            ctx->handler(ctx);
        }

        ngx_resolver_free_node(r, rn);

        return;
    }

    i = ans;
    naddrs = 0;
    cname = NULL;

    for (a = 0; a < nan; a++) {

        start = i;

        while (i < last) {

            if (buf[i] & 0xc0) {
                i += 2;
                goto found;
            }

            if (buf[i] == 0) {
                i++;
                goto test_length;
            }

            i += 1 + buf[i];
        }

        goto short_response;

    test_length:

        if (i - start < 2) {
            err = "invalid name in DNS response";
            goto invalid;
        }

    found:

        if (i + sizeof(ngx_resolver_an_t) >= last) {
            goto short_response;
        }

        an = (ngx_resolver_an_t *) &buf[i];

        type = (an->type_hi << 8) + an->type_lo;
        class = (an->class_hi << 8) + an->class_lo;
        len = (an->len_hi << 8) + an->len_lo;
        ttl = (an->ttl[0] << 24) + (an->ttl[1] << 16)
            + (an->ttl[2] << 8) + (an->ttl[3]);
```

```
1636
1637            if (class != 1) {
1638                ngx_log_error(r->log_level, r->log, 0,
1639                              "unexpected RR class %ui", class);
1640                goto failed;
1641            }
1642
1643            if (ttl < 0) {
1644                ttl = 0;
1645            }
1646
1647            rn->ttl = ngx_min(rn->ttl, (uint32_t) ttl);
1648
1649            i += sizeof(ngx_resolver_an_t);
1650
1651            switch (type) {
1652
1653            case NGX_RESOLVE_A:
1654
1655                if (qtype != NGX_RESOLVE_A) {
1656                    err = "unexpected A record in DNS response";
1657                    goto invalid;
1658                }
1659
1660                if (len != 4) {
1661                    err = "invalid A record in DNS response";
1662                    goto invalid;
1663                }
1664
1665                if (i + 4 > last) {
1666                    goto short_response;
1667                }
1668
1669                naddrs++;
1670
1671                break;
1672
1673 #if (NGX_HAVE_INET6)
1674            case NGX_RESOLVE_AAAA:
1675
1676                if (qtype != NGX_RESOLVE_AAAA) {
1677                    err = "unexpected AAAA record in DNS response";
1678                    goto invalid;
1679                }
1680
1681                if (len != 16) {
1682                    err = "invalid AAAA record in DNS response";
1683                    goto invalid;
1684                }
1685
1686                if (i + 16 > last) {
1687                    goto short_response;
1688                }
1689
1690                naddrs++;
1691
1692                break;
1693 #endif
1694
1695            case NGX_RESOLVE_CNAME:
1696
1697                cname = &buf[i];
1698
1699                break;
1700
1701            case NGX_RESOLVE_DNAME:
1702
1703                break;
1704
1705            default:
1706
1707                ngx_log_error(r->log_level, r->log, 0,
1708                              "unexpected RR type %ui", type);
1709            }
1710
1711            i += len;
```

```
1712      }
1713
1714      ngx_log_debug3(NGX_LOG_DEBUG_CORE, r->log, 0,
1715                     "resolver naddrs:%ui cname:%p ttl:%uD",
1716                     naddrs, cname, rn->ttl);
1717
1718      if (naddrs) {
1719
1720          switch (qtype) {
1721
1722  #if (NGX_HAVE_INET6)
1723          case NGX_RESOLVE_AAAA:
1724
1725              if (naddrs == 1) {
1726                  addr6 = &rn->u6.addr6;
1727                  rn->naddrs6 = 1;
1728
1729              } else {
1730                  addr6 = ngx_resolver_alloc(r, naddrs * sizeof(struct in6_addr));
1731                  if (addr6 == NULL) {
1732                      goto failed;
1733                  }
1734
1735                  rn->u6.addrs6 = addr6;
1736                  rn->naddrs6 = (u_short) naddrs;
1737              }
1738
1739  #if (NGX_SUPPRESS_WARN)
1740              addr = NULL;
1741  #endif
1742
1743              break;
1744  #endif
1745
1746          default: /* NGX_RESOLVE_A */
1747
1748              if (naddrs == 1) {
1749                  addr = &rn->u.addr;
1750                  rn->naddrs = 1;
1751
1752              } else {
1753                  addr = ngx_resolver_alloc(r, naddrs * sizeof(in_addr_t));
1754                  if (addr == NULL) {
1755                      goto failed;
1756                  }
1757
1758                  rn->u.addrs = addr;
1759                  rn->naddrs = (u_short) naddrs;
1760              }
1761
1762  #if (NGX_HAVE_INET6 && NGX_SUPPRESS_WARN)
1763              addr6 = NULL;
1764  #endif
1765          }
1766
1767          n = 0;
1768          i = ans;
1769
1770          for (a = 0; a < nan; a++) {
1771
1772              for ( ;; ) {
1773
1774                  if (buf[i] & 0xc0) {
1775                      i += 2;
1776                      break;
1777                  }
1778
1779                  if (buf[i] == 0) {
1780                      i++;
1781                      break;
1782                  }
1783
1784                  i += 1 + buf[i];
1785              }
1786
1787              an = (ngx_resolver_an_t *) &buf[i];
```

```
1788
1789            type = (an->type_hi << 8) + an->type_lo;
1790            len = (an->len_hi << 8) + an->len_lo;
1791
1792            i += sizeof(ngx_resolver_an_t);
1793
1794            if (type == NGX_RESOLVE_A) {
1795
1796                addr[n] = htonl((buf[i] << 24) + (buf[i + 1] << 16)
1797                                + (buf[i + 2] << 8) + (buf[i + 3]));
1798
1799                if (++n == naddrs) {
1800
1801 #if (NGX_HAVE_INET6)
1802                    if (rn->naddrs6 == (u_short) -1) {
1803                        goto next;
1804                    }
1805 #endif
1806
1807                    break;
1808                }
1809            }
1810
1811 #if (NGX_HAVE_INET6)
1812            else if (type == NGX_RESOLVE_AAAA) {
1813
1814                ngx_memcpy(addr6[n].s6_addr, &buf[i], 16);
1815
1816                if (++n == naddrs) {
1817
1818                    if (rn->naddrs == (u_short) -1) {
1819                        goto next;
1820                    }
1821
1822                    break;
1823                }
1824            }
1825 #endif
1826
1827            i += len;
1828        }
1829    }
1830
1831    switch (qtype) {
1832
1833 #if (NGX_HAVE_INET6)
1834    case NGX_RESOLVE_AAAA:
1835
1836        if (rn->naddrs6 == (u_short) -1) {
1837            rn->naddrs6 = 0;
1838        }
1839
1840        break;
1841 #endif
1842
1843    default: /* NGX_RESOLVE_A */
1844
1845        if (rn->naddrs == (u_short) -1) {
1846            rn->naddrs = 0;
1847        }
1848    }
1849
1850    if (rn->naddrs != (u_short) -1
1851 #if (NGX_HAVE_INET6)
1852        && rn->naddrs6 != (u_short) -1
1853 #endif
1854        && rn->naddrs
1855 #if (NGX_HAVE_INET6)
1856           + rn->naddrs6
1857 #endif
1858           > 0)
1859    {
1860
1861 #if (NGX_HAVE_INET6)
1862    export:
1863 #endif
```

```
1864
1865            naddrs = rn->naddrs;
1866    #if (NGX_HAVE_INET6)
1867            naddrs += rn->naddrs6;
1868    #endif
1869
1870            if (naddrs == 1 && rn->naddrs == 1) {
1871                addrs = NULL;
1872
1873            } else {
1874                addrs = ngx_resolver_export(r, rn, 0);
1875                if (addrs == NULL) {
1876                    goto failed;
1877                }
1878            }
1879
1880            ngx_queue_remove(&rn->queue);
1881
1882            rn->valid = ngx_time() + (r->valid ? r->valid : (time_t) rn->ttl);
1883            rn->expire = ngx_time() + r->expire;
1884
1885            ngx_queue_insert_head(&r->name_expire_queue, &rn->queue);
1886
1887            next = rn->waiting;
1888            rn->waiting = NULL;
1889
1890            /* unlock name mutex */
1891
1892            while (next) {
1893                ctx = next;
1894                ctx->state = NGX_OK;
1895                ctx->naddrs = naddrs;
1896
1897                if (addrs == NULL) {
1898                    ctx->addrs = &ctx->addr;
1899                    ctx->addr.sockaddr = (struct sockaddr *) &ctx->sin;
1900                    ctx->addr.socklen = sizeof(struct sockaddr_in);
1901                    ngx_memzero(&ctx->sin, sizeof(struct sockaddr_in));
1902                    ctx->sin.sin_family = AF_INET;
1903                    ctx->sin.sin_addr.s_addr = rn->u.addr;
1904
1905                } else {
1906                    ctx->addrs = addrs;
1907                }
1908
1909                next = ctx->next;
1910
1911                ctx->handler(ctx);
1912            }
1913
1914            if (addrs != NULL) {
1915                ngx_resolver_free(r, addrs->sockaddr);
1916                ngx_resolver_free(r, addrs);
1917            }
1918
1919            ngx_resolver_free(r, rn->query);
1920            rn->query = NULL;
1921    #if (NGX_HAVE_INET6)
1922            rn->query6 = NULL;
1923    #endif
1924
1925            return;
1926        }
1927
1928        if (cname) {
1929
1930            /* CNAME only */
1931
1932            if (rn->naddrs == (u_short) -1
1933    #if (NGX_HAVE_INET6)
1934                || rn->naddrs6 == (u_short) -1
1935    #endif
1936                )
1937            {
1938                goto next;
1939            }
```

```
1940
1941            if (ngx_resolver_copy(r, &name, buf, cname, buf + last) != NGX_OK) {
1942                goto failed;
1943            }
1944
1945            ngx_log_debug1(NGX_LOG_DEBUG_CORE, r->log, 0,
1946                           "resolver cname:\"%V\"", &name);
1947
1948            ngx_queue_remove(&rn->queue);
1949
1950            rn->cnlen = (u_short) name.len;
1951            rn->u.cname = name.data;
1952
1953            rn->valid = ngx_time() + (r->valid ? r->valid : (time_t) rn->ttl);
1954            rn->expire = ngx_time() + r->expire;
1955
1956            ngx_queue_insert_head(&r->name_expire_queue, &rn->queue);
1957
1958            ctx = rn->waiting;
1959            rn->waiting = NULL;
1960
1961            if (ctx) {
1962                ctx->name = name;
1963
1964                (void) ngx_resolve_name_locked(r, ctx);
1965            }
1966
1967            ngx_resolver_free(r, rn->query);
1968            rn->query = NULL;
1969    #if (NGX_HAVE_INET6)
1970            rn->query6 = NULL;
1971    #endif
1972
1973            /* unlock name mutex */
1974
1975            return;
1976        }
1977
1978        ngx_log_error(r->log_level, r->log, 0,
1979                      "no A or CNAME types in DNS response");
1980        return;
1981
1982    short_response:
1983
1984        err = "short DNS response";
1985
1986    invalid:
1987
1988        /* unlock name mutex */
1989
1990        ngx_log_error(r->log_level, r->log, 0, err);
1991
1992        return;
1993
1994    failed:
1995
1996    next:
1997
1998        /* unlock name mutex */
1999
2000        return;
2001    }
2002
2003
2004    static void
2005    ngx_resolver_process_ptr(ngx_resolver_t *r, u_char *buf, size_t n,
2006        ngx_uint_t ident, ngx_uint_t code, ngx_uint_t nan)
2007    {
2008        char               *err;
2009        size_t              len;
2010        u_char              text[NGX_SOCKADDR_STRLEN];
2011        in_addr_t           addr;
2012        int32_t             ttl;
2013        ngx_int_t           octet;
2014        ngx_str_t           name;
2015        ngx_uint_t          i, mask, qident, class;
```

```
    ngx_queue_t          *expire_queue;
    ngx_rbtree_t         *tree;
    ngx_resolver_an_t    *an;
    ngx_resolver_ctx_t   *ctx, *next;
    ngx_resolver_node_t  *rn;
#if (NGX_HAVE_INET6)
    uint32_t              hash;
    ngx_int_t             digit;
    struct in6_addr       addr6;
#endif

    if (ngx_resolver_copy(r, NULL, buf,
                          buf + sizeof(ngx_resolver_hdr_t), buf + n)
        != NGX_OK)
    {
        return;
    }

    /* AF_INET */

    addr = 0;
    i = sizeof(ngx_resolver_hdr_t);

    for (mask = 0; mask < 32; mask += 8) {
        len = buf[i++];

        octet = ngx_atoi(&buf[i], len);
        if (octet == NGX_ERROR || octet > 255) {
            goto invalid_in_addr_arpa;
        }

        addr += octet << mask;
        i += len;
    }

    if (ngx_strcasecmp(&buf[i], (u_char *) "\7in-addr\4arpa") == 0) {
        i += sizeof("\7in-addr\4arpa");

        /* lock addr mutex */

        rn = ngx_resolver_lookup_addr(r, addr);

        tree = &r->addr_rbtree;
        expire_queue = &r->addr_expire_queue;

        addr = htonl(addr);
        name.len = ngx_inet_ntop(AF_INET, &addr, text, NGX_SOCKADDR_STRLEN);
        name.data = text;

        goto valid;
    }

invalid_in_addr_arpa:

#if (NGX_HAVE_INET6)

    i = sizeof(ngx_resolver_hdr_t);

    for (octet = 15; octet >= 0; octet--) {
        if (buf[i++] != '\1') {
            goto invalid_ip6_arpa;
        }

        digit = ngx_hextoi(&buf[i++], 1);
        if (digit == NGX_ERROR) {
            goto invalid_ip6_arpa;
        }

        addr6.s6_addr[octet] = (u_char) digit;

        if (buf[i++] != '\1') {
            goto invalid_ip6_arpa;
        }

        digit = ngx_hextoi(&buf[i++], 1);
        if (digit == NGX_ERROR) {
```

```
            goto invalid_ip6_arpa;
        }

        addr6.s6_addr[octet] += (u_char) (digit * 16);
    }

    if (ngx_strcasecmp(&buf[i], (u_char *) "\3ip6\4arpa") == 0) {
        i += sizeof("\3ip6\4arpa");

        /* lock addr mutex */

        hash = ngx_crc32_short(addr6.s6_addr, 16);
        rn = ngx_resolver_lookup_addr6(r, &addr6, hash);

        tree = &r->addr6_rbtree;
        expire_queue = &r->addr6_expire_queue;

        name.len = ngx_inet6_ntop(addr6.s6_addr, text, NGX_SOCKADDR_STRLEN);
        name.data = text;

        goto valid;
    }

invalid_ip6_arpa:
#endif

    ngx_log_error(r->log_level, r->log, 0,
                  "invalid in-addr.arpa or ip6.arpa name in DNS response");
    return;

valid:

    if (rn == NULL || rn->query == NULL) {
        ngx_log_error(r->log_level, r->log, 0,
                      "unexpected response for %V", &name);
        goto failed;
    }

    qident = (rn->query[0] << 8) + rn->query[1];

    if (ident != qident) {
        ngx_log_error(r->log_level, r->log, 0,
                      "wrong ident %ui response for %V, expect %ui",
                      ident, &name, qident);
        goto failed;
    }

    if (code == 0 && nan == 0) {
        code = NGX_RESOLVE_NXDOMAIN;
    }

    if (code) {
        next = rn->waiting;
        rn->waiting = NULL;

        ngx_queue_remove(&rn->queue);

        ngx_rbtree_delete(tree, &rn->node);

        /* unlock addr mutex */

        while (next) {
            ctx = next;
            ctx->state = code;
            next = ctx->next;

            ctx->handler(ctx);
        }

        ngx_resolver_free_node(r, rn);

        return;
    }

    i += sizeof(ngx_resolver_qs_t);
```

```
    if (i + 2 + sizeof(ngx_resolver_an_t) >= n) {
        goto short_response;
    }

    /* compression pointer to *.arpa */

    if (buf[i] != 0xc0 || buf[i + 1] != sizeof(ngx_resolver_hdr_t)) {
        err = "invalid in-addr.arpa or ip6.arpa name in DNS response";
        goto invalid;
    }

    an = (ngx_resolver_an_t *) &buf[i + 2];

    class = (an->class_hi << 8) + an->class_lo;
    len = (an->len_hi << 8) + an->len_lo;
    ttl = (an->ttl[0] << 24) + (an->ttl[1] << 16)
        + (an->ttl[2] << 8) + (an->ttl[3]);

    if (class != 1) {
        ngx_log_error(r->log_level, r->log, 0,
                      "unexpected RR class %ui", class);
        goto failed;
    }

    if (ttl < 0) {
        ttl = 0;
    }

    ngx_log_debug3(NGX_LOG_DEBUG_CORE, r->log, 0,
                   "resolver qt:%ui cl:%ui len:%uz",
                   (an->type_hi << 8) + an->type_lo,
                   class, len);

    i += 2 + sizeof(ngx_resolver_an_t);

    if (i + len > n) {
        goto short_response;
    }

    if (ngx_resolver_copy(r, &name, buf, buf + i, buf + n) != NGX_OK) {
        goto failed;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_CORE, r->log, 0, "resolver an:%V", &name);

    if (name.len != (size_t) rn->nlen
        || ngx_strncmp(name.data, rn->name, name.len) != 0)
    {
        if (rn->nlen) {
            ngx_resolver_free(r, rn->name);
        }

        rn->nlen = (u_short) name.len;
        rn->name = name.data;

        name.data = ngx_resolver_dup(r, rn->name, name.len);
        if (name.data == NULL) {
            goto failed;
        }
    }

    ngx_queue_remove(&rn->queue);

    rn->valid = ngx_time() + (r->valid ? r->valid : ttl);
    rn->expire = ngx_time() + r->expire;

    ngx_queue_insert_head(expire_queue, &rn->queue);

    next = rn->waiting;
    rn->waiting = NULL;

    /* unlock addr mutex */

    while (next) {
        ctx = next;
        ctx->state = NGX_OK;
```

```
        ctx->name = name;
        next = ctx->next;

        ctx->handler(ctx);
    }

    ngx_resolver_free(r, name.data);

    return;

short_response:

    err = "short DNS response";

invalid:

    /* unlock addr mutex */

    ngx_log_error(r->log_level, r->log, 0, err);

    return;

failed:

    /* unlock addr mutex */

    return;
}


static ngx_resolver_node_t *
ngx_resolver_lookup_name(ngx_resolver_t *r, ngx_str_t *name, uint32_t hash)
{
    ngx_int_t             rc;
    ngx_rbtree_node_t    *node, *sentinel;
    ngx_resolver_node_t  *rn;

    node = r->name_rbtree.root;
    sentinel = r->name_rbtree.sentinel;

    while (node != sentinel) {

        if (hash < node->key) {
            node = node->left;
            continue;
        }

        if (hash > node->key) {
            node = node->right;
            continue;
        }

        /* hash == node->key */

        rn = ngx_resolver_node(node);

        rc = ngx_memn2cmp(name->data, rn->name, name->len, rn->nlen);

        if (rc == 0) {
            return rn;
        }

        node = (rc < 0) ? node->left : node->right;
    }

    /* not found */

    return NULL;
}


static ngx_resolver_node_t *
ngx_resolver_lookup_addr(ngx_resolver_t *r, in_addr_t addr)
{
    ngx_rbtree_node_t  *node, *sentinel;

```

```
    node = r->addr_rbtree.root;
    sentinel = r->addr_rbtree.sentinel;

    while (node != sentinel) {

        if (addr < node->key) {
            node = node->left;
            continue;
        }

        if (addr > node->key) {
            node = node->right;
            continue;
        }

        /* addr == node->key */

        return ngx_resolver_node(node);
    }

    /* not found */

    return NULL;
}


#if (NGX_HAVE_INET6)

static ngx_resolver_node_t *
ngx_resolver_lookup_addr6(ngx_resolver_t *r, struct in6_addr *addr,
    uint32_t hash)
{
    ngx_int_t             rc;
    ngx_rbtree_node_t    *node, *sentinel;
    ngx_resolver_node_t  *rn;

    node = r->addr6_rbtree.root;
    sentinel = r->addr6_rbtree.sentinel;

    while (node != sentinel) {

        if (hash < node->key) {
            node = node->left;
            continue;
        }

        if (hash > node->key) {
            node = node->right;
            continue;
        }

        /* hash == node->key */

        rn = ngx_resolver_node(node);

        rc = ngx_memcmp(addr, &rn->addr6, 16);

        if (rc == 0) {
            return rn;
        }

        node = (rc < 0) ? node->left : node->right;
    }

    /* not found */

    return NULL;
}

#endif


static void
ngx_resolver_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
```

```
    ngx_rbtree_node_t    **p;
    ngx_resolver_node_t   *rn, *rn_temp;

    for ( ;; ) {

        if (node->key < temp->key) {

            p = &temp->left;

        } else if (node->key > temp->key) {

            p = &temp->right;

        } else { /* node->key == temp->key */

            rn = ngx_resolver_node(node);
            rn_temp = ngx_resolver_node(temp);

            p = (ngx_memn2cmp(rn->name, rn_temp->name, rn->nlen, rn_temp->nlen)
                 < 0) ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
    ngx_rbt_red(node);
}


#if (NGX_HAVE_INET6)

static void
ngx_resolver_rbtree_insert_addr6_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
    ngx_rbtree_node_t    **p;
    ngx_resolver_node_t   *rn, *rn_temp;

    for ( ;; ) {

        if (node->key < temp->key) {

            p = &temp->left;

        } else if (node->key > temp->key) {

            p = &temp->right;

        } else { /* node->key == temp->key */

            rn = ngx_resolver_node(node);
            rn_temp = ngx_resolver_node(temp);

            p = (ngx_memcmp(&rn->addr6, &rn_temp->addr6, 16)
                 < 0) ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
```

```
2472      ngx_rbt_red(node);
2473  }
2474
2475  #endif
2476
2477
2478  static ngx_int_t
2479  ngx_resolver_create_name_query(ngx_resolver_node_t *rn, ngx_resolver_ctx_t *ctx)
2480  {
2481      u_char              *p, *s;
2482      size_t               len, nlen;
2483      ngx_uint_t           ident;
2484  #if (NGX_HAVE_INET6)
2485      ngx_resolver_t      *r;
2486  #endif
2487      ngx_resolver_qs_t   *qs;
2488      ngx_resolver_hdr_t  *query;
2489
2490      nlen = ctx->name.len ? (1 + ctx->name.len + 1) : 1;
2491
2492      len = sizeof(ngx_resolver_hdr_t) + nlen + sizeof(ngx_resolver_qs_t);
2493
2494  #if (NGX_HAVE_INET6)
2495      r = ctx->resolver;
2496
2497      p = ngx_resolver_alloc(ctx->resolver, r->ipv6 ? len * 2 : len);
2498  #else
2499      p = ngx_resolver_alloc(ctx->resolver, len);
2500  #endif
2501      if (p == NULL) {
2502          return NGX_ERROR;
2503      }
2504
2505      rn->qlen = (u_short) len;
2506      rn->query = p;
2507
2508  #if (NGX_HAVE_INET6)
2509      if (r->ipv6) {
2510          rn->query6 = p + len;
2511      }
2512  #endif
2513
2514      query = (ngx_resolver_hdr_t *) p;
2515
2516      ident = ngx_random();
2517
2518      ngx_log_debug2(NGX_LOG_DEBUG_CORE, ctx->resolver->log, 0,
2519                     "resolve: \"%V\" A %i", &ctx->name, ident & 0xffff);
2520
2521      query->ident_hi = (u_char) ((ident >> 8) & 0xff);
2522      query->ident_lo = (u_char) (ident & 0xff);
2523
2524      /* recursion query */
2525      query->flags_hi = 1; query->flags_lo = 0;
2526
2527      /* one question */
2528      query->nqs_hi = 0; query->nqs_lo = 1;
2529      query->nan_hi = 0; query->nan_lo = 0;
2530      query->nns_hi = 0; query->nns_lo = 0;
2531      query->nar_hi = 0; query->nar_lo = 0;
2532
2533      p += sizeof(ngx_resolver_hdr_t) + nlen;
2534
2535      qs = (ngx_resolver_qs_t *) p;
2536
2537      /* query type */
2538      qs->type_hi = 0; qs->type_lo = NGX_RESOLVE_A;
2539
2540      /* IN query class */
2541      qs->class_hi = 0; qs->class_lo = 1;
2542
2543      /* convert "www.example.com" to "\3www\7example\3com\0" */
2544
2545      len = 0;
2546      p--;
2547      *p-- = '\0';
```

```
2548
2549        if (ctx->name.len == 0)  {
2550            return NGX_DECLINED;
2551        }
2552
2553        for (s = ctx->name.data + ctx->name.len - 1; s >= ctx->name.data; s--) {
2554            if (*s != '.') {
2555                *p = *s;
2556                len++;
2557
2558            } else {
2559                if (len == 0 || len > 255) {
2560                    return NGX_DECLINED;
2561                }
2562
2563                *p = (u_char) len;
2564                len = 0;
2565            }
2566
2567            p--;
2568        }
2569
2570        if (len == 0 || len > 255) {
2571            return NGX_DECLINED;
2572        }
2573
2574        *p = (u_char) len;
2575
2576    #if (NGX_HAVE_INET6)
2577        if (!r->ipv6) {
2578            return NGX_OK;
2579        }
2580
2581        p = rn->query6;
2582
2583        ngx_memcpy(p, rn->query, rn->qlen);
2584
2585        query = (ngx_resolver_hdr_t *) p;
2586
2587        ident = ngx_random();
2588
2589        ngx_log_debug2(NGX_LOG_DEBUG_CORE, ctx->resolver->log, 0,
2590                       "resolve: \"%V\" AAAA %i", &ctx->name, ident & 0xffff);
2591
2592        query->ident_hi = (u_char) ((ident >> 8) & 0xff);
2593        query->ident_lo = (u_char) (ident & 0xff);
2594
2595        p += sizeof(ngx_resolver_hdr_t) + nlen;
2596
2597        qs = (ngx_resolver_qs_t *) p;
2598
2599        qs->type_lo = NGX_RESOLVE_AAAA;
2600    #endif
2601
2602        return NGX_OK;
2603    }
2604
2605
2606    static ngx_int_t
2607    ngx_resolver_create_addr_query(ngx_resolver_node_t *rn, ngx_resolver_ctx_t *ctx)
2608    {
2609        u_char               *p, *d;
2610        size_t                len;
2611        in_addr_t             addr;
2612        ngx_int_t             n;
2613        ngx_uint_t            ident;
2614        ngx_resolver_hdr_t   *query;
2615        struct sockaddr_in   *sin;
2616    #if (NGX_HAVE_INET6)
2617        struct sockaddr_in6  *sin6;
2618    #endif
2619
2620        switch (ctx->addr.sockaddr->sa_family) {
2621
2622    #if (NGX_HAVE_INET6)
2623        case AF_INET6:
```

```
        len = sizeof(ngx_resolver_hdr_t)
              + 64 + sizeof(".ip6.arpa.") - 1
              + sizeof(ngx_resolver_qs_t);

        break;
#endif

    default: /* AF_INET */
        len = sizeof(ngx_resolver_hdr_t)
              + sizeof(".255.255.255.255.in-addr.arpa.") - 1
              + sizeof(ngx_resolver_qs_t);
    }

    p = ngx_resolver_alloc(ctx->resolver, len);
    if (p == NULL) {
        return NGX_ERROR;
    }

    rn->query = p;
    query = (ngx_resolver_hdr_t *) p;

    ident = ngx_random();

    query->ident_hi = (u_char) ((ident >> 8) & 0xff);
    query->ident_lo = (u_char) (ident & 0xff);

    /* recursion query */
    query->flags_hi = 1; query->flags_lo = 0;

    /* one question */
    query->nqs_hi = 0; query->nqs_lo = 1;
    query->nan_hi = 0; query->nan_lo = 0;
    query->nns_hi = 0; query->nns_lo = 0;
    query->nar_hi = 0; query->nar_lo = 0;

    p += sizeof(ngx_resolver_hdr_t);

    switch (ctx->addr.sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) ctx->addr.sockaddr;

        for (n = 15; n >= 0; n--) {
            p = ngx_sprintf(p, "\1%xd\1%xd",
                            sin6->sin6_addr.s6_addr[n] & 0xf,
                            (sin6->sin6_addr.s6_addr[n] >> 4) & 0xf);
        }

        p = ngx_cpymem(p, "\3ip6\4arpa\0", 10);

        break;
#endif

    default: /* AF_INET */

        sin = (struct sockaddr_in *) ctx->addr.sockaddr;
        addr = ntohl(sin->sin_addr.s_addr);

        for (n = 0; n < 32; n += 8) {
            d = ngx_sprintf(&p[1], "%ud", (addr >> n) & 0xff);
            *p = (u_char) (d - &p[1]);
            p = d;
        }

        p = ngx_cpymem(p, "\7in-addr\4arpa\0", 14);
    }

    /* query type "PTR", IN query class */
    p = ngx_cpymem(p, "\0\14\0\1", 4);

    rn->qlen = (u_short) (p - rn->query);

    return NGX_OK;
}
```

```
2700
2701 static ngx_int_t
2702 ngx_resolver_copy(ngx_resolver_t *r, ngx_str_t *name, u_char *buf, u_char *src,
2703     u_char *last)
2704 {
2705     char        *err;
2706     u_char      *p, *dst;
2707     ssize_t      len;
2708     ngx_uint_t   i, n;
2709
2710     p = src;
2711     len = -1;
2712
2713     /*
2714      * compression pointers allow to create endless loop, so we set limit;
2715      * 128 pointers should be enough to store 255-byte name
2716      */
2717
2718     for (i = 0; i < 128; i++) {
2719         n = *p++;
2720
2721         if (n == 0) {
2722             goto done;
2723         }
2724
2725         if (n & 0xc0) {
2726             n = ((n & 0x3f) << 8) + *p;
2727             p = &buf[n];
2728
2729         } else {
2730             len += 1 + n;
2731             p = &p[n];
2732         }
2733
2734         if (p >= last) {
2735             err = "name is out of response";
2736             goto invalid;
2737         }
2738     }
2739
2740     err = "compression pointers loop";
2741
2742 invalid:
2743
2744     ngx_log_error(r->log_level, r->log, 0, err);
2745
2746     return NGX_ERROR;
2747
2748 done:
2749
2750     if (name == NULL) {
2751         return NGX_OK;
2752     }
2753
2754     if (len == -1) {
2755         ngx_str_null(name);
2756         return NGX_OK;
2757     }
2758
2759     dst = ngx_resolver_alloc(r, len);
2760     if (dst == NULL) {
2761         return NGX_ERROR;
2762     }
2763
2764     name->data = dst;
2765
2766     n = *src++;
2767
2768     for ( ;; ) {
2769         if (n & 0xc0) {
2770             n = ((n & 0x3f) << 8) + *src;
2771             src = &buf[n];
2772
2773             n = *src++;
2774
2775         } else {
```

```
            ngx_strlow(dst, src, n);
            dst += n;
            src += n;

            n = *src++;

            if (n != 0) {
                *dst++ = '.';
            }
        }

        if (n == 0) {
            name->len = dst - name->data;
            return NGX_OK;
        }
    }
}


static void
ngx_resolver_timeout_handler(ngx_event_t *ev)
{
    ngx_resolver_ctx_t   *ctx, *next;
    ngx_resolver_node_t  *rn;

    rn = ev->data;
    ctx = rn->waiting;
    rn->waiting = NULL;

    do {
        ctx->state = NGX_RESOLVE_TIMEDOUT;
        next = ctx->next;

        ctx->handler(ctx);

        ctx = next;
    } while (ctx);
}


static void
ngx_resolver_free_node(ngx_resolver_t *r, ngx_resolver_node_t *rn)
{
    /* lock alloc mutex */

    if (rn->query) {
        ngx_resolver_free_locked(r, rn->query);
    }

    if (rn->name) {
        ngx_resolver_free_locked(r, rn->name);
    }

    if (rn->cnlen) {
        ngx_resolver_free_locked(r, rn->u.cname);
    }

    if (rn->naddrs > 1 && rn->naddrs != (u_short) -1) {
        ngx_resolver_free_locked(r, rn->u.addrs);
    }

#if (NGX_HAVE_INET6)
    if (rn->naddrs6 > 1 && rn->naddrs6 != (u_short) -1) {
        ngx_resolver_free_locked(r, rn->u6.addrs6);
    }
#endif

    ngx_resolver_free_locked(r, rn);

    /* unlock alloc mutex */
}


static void *
ngx_resolver_alloc(ngx_resolver_t *r, size_t size)
{
```

```
2852      u_char  *p;
2853
2854      /* lock alloc mutex */
2855
2856      p = ngx_alloc(size, r->log);
2857
2858      /* unlock alloc mutex */
2859
2860      return p;
2861  }
2862
2863
2864  static void *
2865  ngx_resolver_calloc(ngx_resolver_t *r, size_t size)
2866  {
2867      u_char  *p;
2868
2869      p = ngx_resolver_alloc(r, size);
2870
2871      if (p) {
2872          ngx_memzero(p, size);
2873      }
2874
2875      return p;
2876  }
2877
2878
2879  static void
2880  ngx_resolver_free(ngx_resolver_t *r, void *p)
2881  {
2882      /* lock alloc mutex */
2883
2884      ngx_free(p);
2885
2886      /* unlock alloc mutex */
2887  }
2888
2889
2890  static void
2891  ngx_resolver_free_locked(ngx_resolver_t *r, void *p)
2892  {
2893      ngx_free(p);
2894  }
2895
2896
2897  static void *
2898  ngx_resolver_dup(ngx_resolver_t *r, void *src, size_t size)
2899  {
2900      void  *dst;
2901
2902      dst = ngx_resolver_alloc(r, size);
2903
2904      if (dst == NULL) {
2905          return dst;
2906      }
2907
2908      ngx_memcpy(dst, src, size);
2909
2910      return dst;
2911  }
2912
2913
2914  static ngx_addr_t *
2915  ngx_resolver_export(ngx_resolver_t *r, ngx_resolver_node_t *rn,
2916      ngx_uint_t rotate)
2917  {
2918      ngx_addr_t            *dst;
2919      ngx_uint_t             d, i, j, n;
2920      u_char               (*sockaddr)[NGX_SOCKADDRLEN];
2921      in_addr_t             *addr;
2922      struct sockaddr_in    *sin;
2923  #if (NGX_HAVE_INET6)
2924      struct in6_addr       *addr6;
2925      struct sockaddr_in6   *sin6;
2926  #endif
2927
```

```
    n = rn->naddrs;
#if (NGX_HAVE_INET6)
    n += rn->naddrs6;
#endif

    dst = ngx_resolver_calloc(r, n * sizeof(ngx_addr_t));
    if (dst == NULL) {
        return NULL;
    }

    sockaddr = ngx_resolver_calloc(r, n * NGX_SOCKADDRLEN);
    if (sockaddr == NULL) {
        ngx_resolver_free(r, dst);
        return NULL;
    }

    i = 0;
    d = rotate ? ngx_random() % n : 0;

    if (rn->naddrs) {
        j = rotate ? ngx_random() % rn->naddrs : 0;

        addr = (rn->naddrs == 1) ? &rn->u.addr : rn->u.addrs;

        do {
            sin = (struct sockaddr_in *) sockaddr[d];
            sin->sin_family = AF_INET;
            sin->sin_addr.s_addr = addr[j++];
            dst[d].sockaddr = (struct sockaddr *) sin;
            dst[d++].socklen = sizeof(struct sockaddr_in);

            if (d == n) {
                d = 0;
            }

            if (j == rn->naddrs) {
                j = 0;
            }
        } while (++i < rn->naddrs);
    }

#if (NGX_HAVE_INET6)
    if (rn->naddrs6) {
        j = rotate ? ngx_random() % rn->naddrs6 : 0;

        addr6 = (rn->naddrs6 == 1) ? &rn->u6.addr6 : rn->u6.addrs6;

        do {
            sin6 = (struct sockaddr_in6 *) sockaddr[d];
            sin6->sin6_family = AF_INET6;
            ngx_memcpy(sin6->sin6_addr.s6_addr, addr6[j++].s6_addr, 16);
            dst[d].sockaddr = (struct sockaddr *) sin6;
            dst[d++].socklen = sizeof(struct sockaddr_in6);

            if (d == n) {
                d = 0;
            }

            if (j == rn->naddrs6) {
                j = 0;
            }
        } while (++i < n);
    }
#endif

    return dst;
}


char *
ngx_resolver_strerror(ngx_int_t err)
{
    static char *errors[] = {
        "Format error",     /* FORMERR */
        "Server failure",   /* SERVFAIL */
        "Host not found",   /* NXDOMAIN */
```

```
3004         "Unimplemented",    /* NOTIMP */
3005         "Operation refused" /* REFUSED */
3006     };
3007
3008     if (err > 0 && err < 6) {
3009         return errors[err - 1];
3010     }
3011
3012     if (err == NGX_RESOLVE_TIMEDOUT) {
3013         return "Operation timed out";
3014     }
3015
3016     return "Unknown error";
3017 }
3018
3019
3020 static u_char *
3021 ngx_resolver_log_error(ngx_log_t *log, u_char *buf, size_t len)
3022 {
3023     u_char               *p;
3024     ngx_udp_connection_t  *uc;
3025
3026     p = buf;
3027
3028     if (log->action) {
3029         p = ngx_snprintf(buf, len, " while %s", log->action);
3030         len -= p - buf;
3031     }
3032
3033     uc = log->data;
3034
3035     if (uc) {
3036         p = ngx_snprintf(p, len, ", resolver: %V", &uc->server);
3037     }
3038
3039     return p;
3040 }
3041
3042
3043 ngx_int_t
3044 ngx_udp_connect(ngx_udp_connection_t *uc)
3045 {
3046     int                rc;
3047     ngx_int_t          event;
3048     ngx_event_t       *rev, *wev;
3049     ngx_socket_t       s;
3050     ngx_connection_t  *c;
3051
3052     s = ngx_socket(uc->sockaddr->sa_family, SOCK_DGRAM, 0);
3053
3054     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, &uc->log, 0, "UDP socket %d", s);
3055
3056     if (s == (ngx_socket_t) -1) {
3057         ngx_log_error(NGX_LOG_ALERT, &uc->log, ngx_socket_errno,
3058                       ngx_socket_n " failed");
3059         return NGX_ERROR;
3060     }
3061
3062     c = ngx_get_connection(s, &uc->log);
3063
3064     if (c == NULL) {
3065         if (ngx_close_socket(s) == -1) {
3066             ngx_log_error(NGX_LOG_ALERT, &uc->log, ngx_socket_errno,
3067                           ngx_close_socket_n "failed");
3068         }
3069
3070         return NGX_ERROR;
3071     }
3072
3073     if (ngx_nonblocking(s) == -1) {
3074         ngx_log_error(NGX_LOG_ALERT, &uc->log, ngx_socket_errno,
3075                       ngx_nonblocking_n " failed");
3076
3077         goto failed;
3078     }
3079
```

```
3080      rev = c->read;
3081      wev = c->write;
3082
3083      rev->log = &uc->log;
3084      wev->log = &uc->log;
3085
3086      uc->connection = c;
3087
3088      c->number = ngx_atomic_fetch_add(ngx_connection_counter, 1);
3089
3090      ngx_log_debug3(NGX_LOG_DEBUG_EVENT, &uc->log, 0,
3091                     "connect to %V, fd:%d #%uA", &uc->server, s, c->number);
3092
3093      rc = connect(s, uc->sockaddr, uc->socklen);
3094
3095      /* TODO: aio, iocp */
3096
3097      if (rc == -1) {
3098          ngx_log_error(NGX_LOG_CRIT, &uc->log, ngx_socket_errno,
3099                        "connect() failed");
3100
3101          goto failed;
3102      }
3103
3104      /* UDP sockets are always ready to write */
3105      wev->ready = 1;
3106
3107      if (ngx_add_event) {
3108
3109          event = (ngx_event_flags & NGX_USE_CLEAR_EVENT) ?
3110                      /* kqueue, epoll */                 NGX_CLEAR_EVENT:
3111                      /* select, poll, /dev/poll */      NGX_LEVEL_EVENT;
3112                      /* eventport event type has no meaning: oneshot only */
3113
3114          if (ngx_add_event(rev, NGX_READ_EVENT, event) != NGX_OK) {
3115              goto failed;
3116          }
3117
3118      } else {
3119          /* rtsig */
3120
3121          if (ngx_add_conn(c) == NGX_ERROR) {
3122              goto failed;
3123          }
3124      }
3125
3126      return NGX_OK;
3127
3128  failed:
3129
3130      ngx_close_connection(c);
3131      uc->connection = NULL;
3132
3133      return NGX_ERROR;
3134  }
```

One Level Up Top Level

src/core/ngx_parse.c - nginx-1.7.10

Functions defined

- ngx_parse_offset
- ngx_parse_size
- ngx_parse_time

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  ssize_t
13  ngx_parse_size(ngx_str_t *line)
14  {
15      u_char     unit;
16      size_t     len;
17      ssize_t    size;
18      ngx_int_t  scale;
19
20      len = line->len;
21      unit = line->data[len - 1];
22
23      switch (unit) {
24      case 'K':
25      case 'k':
26          len--;
27          scale = 1024;
28          break;
29
30      case 'M':
31      case 'm':
32          len--;
33          scale = 1024 * 1024;
34          break;
35
36      default:
37          scale = 1;
38      }
39
40      size = ngx_atosz(line->data, len);
41      if (size == NGX_ERROR) {
42          return NGX_ERROR;
43      }
44
45      size *= scale;
46
47      return size;
48  }
49
50
51  off_t
52  ngx_parse_offset(ngx_str_t *line)
53  {
54      u_char     unit;
55      off_t      offset;
56      size_t     len;
57      ngx_int_t  scale;
58
```

```
    len = line->len;
    unit = line->data[len - 1];

    switch (unit) {
    case 'K':
    case 'k':
        len--;
        scale = 1024;
        break;

    case 'M':
    case 'm':
        len--;
        scale = 1024 * 1024;
        break;

    case 'G':
    case 'g':
        len--;
        scale = 1024 * 1024 * 1024;
        break;

    default:
        scale = 1;
    }

    offset = ngx_atoof(line->data, len);
    if (offset == NGX_ERROR) {
        return NGX_ERROR;
    }

    offset *= scale;

    return offset;
}


ngx_int_t
ngx_parse_time(ngx_str_t *line, ngx_uint_t is_sec)
{
    u_char      *p, *last;
     ngx_int_t    value, total, scale;
     ngx_uint_t   max, valid;
     enum {
         st_start = 0,
         st_year,
         st_month,
         st_week,
         st_day,
         st_hour,
         st_min,
         st_sec,
         st_msec,
         st_last
     } step;

     valid = 0;
     value = 0;
     total = 0;
     step = is_sec ? st_start : st_month;
     scale = is_sec ? 1 : 1000;

     p = line->data;
     last = p + line->len;

     while (p < last) {

         if (*p >= '0' && *p <= '9') {
             value = value * 10 + (*p++ - '0');
             valid = 1;
             continue;
         }

         switch (*p++) {

         case 'y':
```

```
            if (step > st_start) {
                return NGX_ERROR;
            }
            step = st_year;
            max = NGX_MAX_INT32_VALUE / (60 * 60 * 24 * 365);
            scale = 60 * 60 * 24 * 365;
            break;

        case 'M':
            if (step >= st_month) {
                return NGX_ERROR;
            }
            step = st_month;
            max = NGX_MAX_INT32_VALUE / (60 * 60 * 24 * 30);
            scale = 60 * 60 * 24 * 30;
            break;

        case 'w':
            if (step >= st_week) {
                return NGX_ERROR;
            }
            step = st_week;
            max = NGX_MAX_INT32_VALUE / (60 * 60 * 24 * 7);
            scale = 60 * 60 * 24 * 7;
            break;

        case 'd':
            if (step >= st_day) {
                return NGX_ERROR;
            }
            step = st_day;
            max = NGX_MAX_INT32_VALUE / (60 * 60 * 24);
            scale = 60 * 60 * 24;
            break;

        case 'h':
            if (step >= st_hour) {
                return NGX_ERROR;
            }
            step = st_hour;
            max = NGX_MAX_INT32_VALUE / (60 * 60);
            scale = 60 * 60;
            break;

        case 'm':
            if (*p == 's') {
                if (is_sec || step >= st_msec) {
                    return NGX_ERROR;
                }
                p++;
                step = st_msec;
                max = NGX_MAX_INT32_VALUE;
                scale = 1;
                break;
            }

            if (step >= st_min) {
                return NGX_ERROR;
            }
            step = st_min;
            max = NGX_MAX_INT32_VALUE / 60;
            scale = 60;
            break;

        case 's':
            if (step >= st_sec) {
                return NGX_ERROR;
            }
            step = st_sec;
            max = NGX_MAX_INT32_VALUE;
            scale = 1;
            break;

        case ' ':
            if (step >= st_sec) {
                return NGX_ERROR;
```

```
            }
            step = st_last;
            max = NGX_MAX_INT32_VALUE;
            scale = 1;
            break;

        default:
            return NGX_ERROR;
        }

        if (step != st_msec && !is_sec) {
            scale *= 1000;
            max /= 1000;
        }

        if ((ngx_uint_t) value > max) {
            return NGX_ERROR;
        }

        total += value * scale;

        if ((ngx_uint_t) total > NGX_MAX_INT32_VALUE) {
            return NGX_ERROR;
        }

        value = 0;
        scale = is_sec ? 1 : 1000;

        while (p < last && *p == ' ') {
            p++;
        }
    }

    if (valid) {
        return total + value * scale;
    }

    return NGX_ERROR;
}
```

One Level Up Top Level

src/event/ngx_event_connect.h - nginx-1.7.10

Data types defined

- ngx_event_free_peer_pt
- ngx_event_get_peer_pt
- ngx_event_save_peer_session_pt
- ngx_event_set_peer_session_pt
- ngx_peer_connection_s
- ngx_peer_connection_t

Macros defined

- NGX_PEER_FAILED
- NGX_PEER_KEEPALIVE
- NGX_PEER_NEXT
- _NGX_EVENT_CONNECT_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_EVENT_CONNECT_H_INCLUDED_
9  #define _NGX_EVENT_CONNECT_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15
16
17  #define NGX_PEER_KEEPALIVE           1
18  #define NGX_PEER_NEXT                2
19  #define NGX_PEER_FAILED              4
20
21
22  typedef struct ngx_peer_connection_s  ngx_peer_connection_t;
23
24  typedef ngx_int_t (*ngx_event_get_peer_pt)(ngx_peer_connection_t *pc,
25      void *data);
26  typedef void (*ngx_event_free_peer_pt)(ngx_peer_connection_t *pc, void *data,
27      ngx_uint_t state);
28  #if (NGX_SSL)
29
30  typedef ngx_int_t (*ngx_event_set_peer_session_pt)(ngx_peer_connection_t *pc,
31      void *data);
32  typedef void (*ngx_event_save_peer_session_pt)(ngx_peer_connection_t *pc,
33      void *data);
34  #endif
35
36
37  struct ngx_peer_connection_s {
38      ngx_connection_t                *connection;
```

```
39
40      struct sockaddr                 *sockaddr;
41      socklen_t                        socklen;
42      ngx_str_t                       *name;
43
44      ngx_uint_t                       tries;
45      ngx_msec_t                       start_time;
46
47      ngx_event_get_peer_pt            get;
48      ngx_event_free_peer_pt           free;
49      void                            *data;
50
51  #if (NGX_SSL)
52      ngx_event_set_peer_session_pt    set_session;
53      ngx_event_save_peer_session_pt   save_session;
54  #endif
55
56  #if (NGX_THREADS)
57      ngx_atomic_t                    *lock;
58  #endif
59
60      ngx_addr_t                      *local;
61
62      int                              rcvbuf;
63
64      ngx_log_t                       *log;
65
66      unsigned                         cached:1;
67
68                                       /* ngx_connection_log_error_e */
69      unsigned                         log_error:2;
70  };
71
72
73  ngx_int_t ngx_event_connect_peer(ngx_peer_connection_t *pc);
74  ngx_int_t ngx_event_get_peer(ngx_peer_connection_t *pc, void *data);
75
76
77  #endif /* _NGX_EVENT_CONNECT_H_INCLUDED_ */
```

One Level Up Top Level

src/event/ngx_event_connect.c - nginx-1.7.10

Functions defined

- ngx_event_connect_peer
- ngx_event_get_peer

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_event_connect.h>
12
13
14  ngx_int_t
15  ngx_event_connect_peer(ngx_peer_connection_t *pc)
16  {
17      int                rc;
18      ngx_int_t          event;
19      ngx_err_t          err;
20      ngx_uint_t         level;
21      ngx_socket_t       s;
22      ngx_event_t       *rev, *wev;
23      ngx_connection_t  *c;
24
25      rc = pc->get(pc, pc->data);
26      if (rc != NGX_OK) {
27          return rc;
28      }
29
30      s = ngx_socket(pc->sockaddr->sa_family, SOCK_STREAM, 0);
31
32      ngx_log_debug1(NGX_LOG_DEBUG_EVENT, pc->log, 0, "socket %d", s);
33
34      if (s == (ngx_socket_t) -1) {
35          ngx_log_error(NGX_LOG_ALERT, pc->log, ngx_socket_errno,
36                        ngx_socket_n " failed");
37          return NGX_ERROR;
38      }
39
40
41      c = ngx_get_connection(s, pc->log);
42
43      if (c == NULL) {
44          if (ngx_close_socket(s) == -1) {
45              ngx_log_error(NGX_LOG_ALERT, pc->log, ngx_socket_errno,
46                            ngx_close_socket_n "failed");
47          }
48
49          return NGX_ERROR;
50      }
51
52      if (pc->rcvbuf) {
53          if (setsockopt(s, SOL_SOCKET, SO_RCVBUF,
54                         (const void *) &pc->rcvbuf, sizeof(int)) == -1)
55          {
56              ngx_log_error(NGX_LOG_ALERT, pc->log, ngx_socket_errno,
57                            "setsockopt(SO_RCVBUF) failed");
58              goto failed;
59          }
60      }
```

```
61
62      if (ngx_nonblocking(s) == -1) {
63          ngx_log_error(NGX_LOG_ALERT, pc->log, ngx_socket_errno,
64                        ngx_nonblocking_n " failed");
65
66          goto failed;
67      }
68
69      if (pc->local) {
70          if (bind(s, pc->local->sockaddr, pc->local->socklen) == -1) {
71              ngx_log_error(NGX_LOG_CRIT, pc->log, ngx_socket_errno,
72                            "bind(%V) failed", &pc->local->name);
73
74              goto failed;
75          }
76      }
77
78      c->recv = ngx_recv;
79      c->send = ngx_send;
80      c->recv_chain = ngx_recv_chain;
81      c->send_chain = ngx_send_chain;
82
83      c->sendfile = 1;
84
85      c->log_error = pc->log_error;
86
87      if (pc->sockaddr->sa_family == AF_UNIX) {
88          c->tcp_nopush = NGX_TCP_NOPUSH_DISABLED;
89          c->tcp_nodelay = NGX_TCP_NODELAY_DISABLED;
90
91  #if (NGX_SOLARIS)
92          /* Solaris's sendfilev() supports AF_NCA, AF_INET, and AF_INET6 */
93          c->sendfile = 0;
94  #endif
95      }
96
97      rev = c->read;
98      wev = c->write;
99
100      rev->log = pc->log;
101      wev->log = pc->log;
102
103      pc->connection = c;
104
105      c->number = ngx_atomic_fetch_add(ngx_connection_counter, 1);
106
107      if (ngx_add_conn) {
108          if (ngx_add_conn(c) == NGX_ERROR) {
109              goto failed;
110          }
111      }
112
113      ngx_log_debug3(NGX_LOG_DEBUG_EVENT, pc->log, 0,
114                     "connect to %V, fd:%d #%uA", pc->name, s, c->number);
115
116      rc = connect(s, pc->sockaddr, pc->socklen);
117
118      if (rc == -1) {
119          err = ngx_socket_errno;
120
121
122          if (err != NGX_EINPROGRESS
123  #if (NGX_WIN32)
124              /* Winsock returns WSAEWOULDBLOCK (NGX_EAGAIN) */
125              && err != NGX_EAGAIN
126  #endif
127              )
128          {
129              if (err == NGX_ECONNREFUSED
130  #if (NGX_LINUX)
131                  /*
132                   * Linux returns EAGAIN instead of ECONNREFUSED
133                   * for unix sockets if listen queue is full
134                   */
135                  || err == NGX_EAGAIN
136  #endif
```

```
                || err == NGX_ECONNRESET
                || err == NGX_ENETDOWN
                || err == NGX_ENETUNREACH
                || err == NGX_EHOSTDOWN
                || err == NGX_EHOSTUNREACH)
            {
                level = NGX_LOG_ERR;

            } else {
                level = NGX_LOG_CRIT;
            }

            ngx_log_error(level, c->log, err, "connect() to %V failed",
                          pc->name);

            ngx_close_connection(c);
            pc->connection = NULL;

            return NGX_DECLINED;
        }
    }

    if (ngx_add_conn) {
        if (rc == -1) {

            /* NGX_EINPROGRESS */

            return NGX_AGAIN;
        }

        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, pc->log, 0, "connected");

        wev->ready = 1;

        return NGX_OK;
    }

    if (ngx_event_flags & NGX_USE_AIO_EVENT) {

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, pc->log, ngx_socket_errno,
                       "connect(): %d", rc);

        /* aio, iocp */

        if (ngx_blocking(s) == -1) {
            ngx_log_error(NGX_LOG_ALERT, pc->log, ngx_socket_errno,
                          ngx_blocking_n " failed");
            goto failed;
        }

        /*
         * FreeBSD's aio allows to post an operation on non-connected socket.
         * NT does not support it.
         *
         * TODO: check in Win32, etc. As workaround we can use NGX_ONESHOT_EVENT
         */

        rev->ready = 1;
        wev->ready = 1;

        return NGX_OK;
    }

    if (ngx_event_flags & NGX_USE_CLEAR_EVENT) {

        /* kqueue */

        event = NGX_CLEAR_EVENT;

    } else {

        /* select, poll, /dev/poll */

        event = NGX_LEVEL_EVENT;
    }

```

```
213     if (ngx_add_event(rev, NGX_READ_EVENT, event) != NGX_OK) {
214         goto failed;
215     }
216 
217     if (rc == -1) {
218 
219         /* NGX_EINPROGRESS */
220 
221         if (ngx_add_event(wev, NGX_WRITE_EVENT, event) != NGX_OK) {
222             goto failed;
223         }
224 
225         return NGX_AGAIN;
226     }
227 
228     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, pc->log, 0, "connected");
229 
230     wev->ready = 1;
231 
232     return NGX_OK;
233 
234 failed:
235 
236     ngx_close_connection(c);
237     pc->connection = NULL;
238 
239     return NGX_ERROR;
240 }
241 
242 
243 ngx_int_t
244 ngx_event_get_peer(ngx_peer_connection_t *pc, void *data)
245 {
246     return NGX_OK;
247 }
```

One Level Up Top Level

src/os/unix/ngx_solaris_sendfilev_chain.c - nginx-1.7.10

Data types defined

- sendfilevec
- sendfilevec_t

Functions defined

- ngx_solaris_sendfilev_chain
- sendfilev

Macros defined

- NGX_SENDFILEVECS
- SFV_FD_SELF

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #if (NGX_TEST_BUILD_SOLARIS_SENDFILEV)
14
15  /* Solaris declarations */
16
17  typedef struct sendfilevec {
18      int     sfv_fd;
19      u_int   sfv_flag;
20      off_t   sfv_off;
21      size_t  sfv_len;
22  } sendfilevec_t;
23
24  #define SFV_FD_SELF  -2
25
26  static ssize_t sendfilev(int fd, const struct sendfilevec *vec,
27      int sfvcnt, size_t *xferred)
28  {
29      return -1;
30  }
31
32  ngx_chain_t *ngx_solaris_sendfilev_chain(ngx_connection_t *c, ngx_chain_t *in,
33      off_t limit);
34
35  #endif
36
37
38  #define NGX_SENDFILEVECS  NGX_IOVS_PREALLOCATE
39
40
41  ngx_chain_t *
42  ngx_solaris_sendfilev_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
43  {
44      int             fd;
```

```
    u_char          *prev;
    off_t            size, send, prev_send, aligned, fprev;
    size_t           sent;
    ssize_t          n;
    ngx_int_t        eintr;
    ngx_err_t        err;
    ngx_uint_t       nsfv;
    sendfilevec_t   *sfv, sfvs[NGX_SENDFILEVECS];
    ngx_event_t     *wev;
    ngx_chain_t     *cl;

    wev = c->write;

    if (!wev->ready) {
        return in;
    }

    if (!c->sendfile) {
        return ngx_writev_chain(c, in, limit);
    }


    /* the maximum limit size is the maximum size_t value - the page size */

    if (limit == 0 || limit > (off_t) (NGX_MAX_SIZE_T_VALUE - ngx_pagesize)) {
        limit = NGX_MAX_SIZE_T_VALUE - ngx_pagesize;
    }


    send = 0;

    for ( ;; ) {
        fd = SFV_FD_SELF;
        prev = NULL;
        fprev = 0;
        sfv = NULL;
        eintr = 0;
        sent = 0;
        prev_send = send;

        nsfv = 0;

        /* create the sendfilevec and coalesce the neighbouring bufs */

        for (cl = in; cl && send < limit; cl = cl->next) {

            if (ngx_buf_special(cl->buf)) {
                continue;
            }

            if (ngx_buf_in_memory_only(cl->buf)) {
                fd = SFV_FD_SELF;

                size = cl->buf->last - cl->buf->pos;

                 if (send + size > limit) {
                     size = limit - send;
                 }

                 if (prev == cl->buf->pos) {
                     sfv->sfv_len += (size_t) size;

                 } else {
                     if (nsfv == NGX_SENDFILEVECS) {
                         break;
                     }

                     sfv = &sfvs[nsfv++];

                     sfv->sfv_fd = SFV_FD_SELF;
                     sfv->sfv_flag = 0;
                     sfv->sfv_off = (off_t) (uintptr_t) cl->buf->pos;
                     sfv->sfv_len = (size_t) size;
                 }

                 prev = cl->buf->pos + (size_t) size;
```

```
            send += size;

        } else {
            prev = NULL;

            size = cl->buf->file_last - cl->buf->file_pos;

            if (send + size > limit) {
                size = limit - send;

                aligned = (cl->buf->file_pos + size + ngx_pagesize - 1)
                          & ~((off_t) ngx_pagesize - 1);

                if (aligned <= cl->buf->file_last) {
                    size = aligned - cl->buf->file_pos;
                }
            }

            if (fd == cl->buf->file->fd && fprev == cl->buf->file_pos) {
                sfv->sfv_len += (size_t) size;

            } else {
                if (nsfv == NGX_SENDFILEVECS) {
                    break;
                }

                sfv = &sfvs[nsfv++];

                fd = cl->buf->file->fd;
                sfv->sfv_fd = fd;
                sfv->sfv_flag = 0;
                sfv->sfv_off = cl->buf->file_pos;
                sfv->sfv_len = (size_t) size;
            }

            fprev = cl->buf->file_pos + size;
            send += size;
        }
    }

    n = sendfilev(c->fd, sfvs, nsfv, &sent);

    if (n == -1) {
        err = ngx_errno;

        switch (err) {
        case NGX_EAGAIN:
            break;

        case NGX_EINTR:
            eintr = 1;
            break;

        default:
            wev->error = 1;
            ngx_connection_error(c, err, "sendfilev() failed");
            return NGX_CHAIN_ERROR;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, err,
                       "sendfilev() sent only %uz bytes", sent);
    }

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "sendfilev: %z %z", n, sent);

    c->sent += sent;

    in = ngx_chain_update_sent(in, sent);

    if (eintr) {
        send = prev_send + sent;
        continue;
    }

    if (send - prev_send != (off_t) sent) {
```

```
197             wev->ready = 0;
198             return in;
199         }
200
201         if (send >= limit || in == NULL) {
202             return in;
203         }
204     }
205 }
```

One Level Up Top Level

src/os/unix/ngx_writev_chain.c - nginx-1.7.10

Functions defined

- ngx_output_chain_to_iovec
- ngx_writev
- ngx_writev_chain

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10 #include <ngx_event.h>
11
12
13 ngx_chain_t *
14 ngx_writev_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
15 {
16     ssize_t        n, sent;
17     off_t          send, prev_send;
18     ngx_chain_t   *cl;
19     ngx_event_t   *wev;
20     ngx_iovec_t    vec;
21     struct iovec   iovs[NGX_IOVS_PREALLOCATE];
22
23     wev = c->write;
24
25     if (!wev->ready) {
26         return in;
27     }
28
29 #if (NGX_HAVE_KQUEUE)
30
31     if ((ngx_event_flags & NGX_USE_KQUEUE_EVENT) && wev->pending_eof) {
32         (void) ngx_connection_error(c, wev->kq_errno,
33                                     "kevent() reported about an closed connection");
34         wev->error = 1;
35         return NGX_CHAIN_ERROR;
36     }
37
38 #endif
39
40     /* the maximum limit size is the maximum size_t value - the page size */
41
42     if (limit == 0 || limit > (off_t) (NGX_MAX_SIZE_T_VALUE - ngx_pagesize)) {
43         limit = NGX_MAX_SIZE_T_VALUE - ngx_pagesize;
44     }
45
46     send = 0;
47
48     vec.iovs = iovs;
49     vec.nalloc = NGX_IOVS_PREALLOCATE;
50
51     for ( ;; ) {
52         prev_send = send;
53
54         /* create the iovec and coalesce the neighbouring bufs */
55
56         cl = ngx_output_chain_to_iovec(&vec, in, limit - send, c->log);
57
58         if (cl == NGX_CHAIN_ERROR) {
```

```
            return NGX_CHAIN_ERROR;
        }

        if (cl && cl->buf->in_file) {
            ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                          "file buf in writev "
                          "t:%d r:%d f:%d %p %p-%p %p %O-%O",
                          cl->buf->temporary,
                          cl->buf->recycled,
                          cl->buf->in_file,
                          cl->buf->start,
                          cl->buf->pos,
                          cl->buf->last,
                          cl->buf->file,
                          cl->buf->file_pos,
                          cl->buf->file_last);

            ngx_debug_point();

            return NGX_CHAIN_ERROR;
        }

        send += vec.size;

        n = ngx_writev(c, &vec);

        if (n == NGX_ERROR) {
            return NGX_CHAIN_ERROR;
        }

        sent = (n == NGX_AGAIN) ? 0 : n;

        c->sent += sent;

        in = ngx_chain_update_sent(in, sent);

        if (send - prev_send != sent) {
            wev->ready = 0;
            return in;
        }

         if (send >= limit || in == NULL) {
             return in;
         }
    }
}


ngx_chain_t *
ngx_output_chain_to_iovec(ngx_iovec_t *vec, ngx_chain_t *in, size_t limit,
    ngx_log_t *log)
{
    size_t         total, size;
    u_char        *prev;
    ngx_uint_t     n;
    struct iovec  *iov;

    iov = NULL;
    prev = NULL;
    total = 0;
    n = 0;

    for ( /* void */ ; in && total < limit; in = in->next) {

         if (ngx_buf_special(in->buf)) {
             continue;
         }

         if (in->buf->in_file) {
             break;
         }

         if (!ngx_buf_in_memory(in->buf)) {
             ngx_log_error(NGX_LOG_ALERT, log, 0,
                           "bad buf in output chain "
                           "t:%d r:%d f:%d %p %p-%p %p %O-%O",
```

```
                          in->buf->temporary,
                          in->buf->recycled,
                          in->buf->in_file,
                          in->buf->start,
                          in->buf->pos,
                          in->buf->last,
                          in->buf->file,
                          in->buf->file_pos,
                          in->buf->file_last);

            ngx_debug_point();

            return NGX_CHAIN_ERROR;
        }

        size = in->buf->last - in->buf->pos;

        if (size > limit - total) {
            size = limit - total;
        }

        if (prev == in->buf->pos) {
            iov->iov_len += size;

        } else {
            if (n == vec->nalloc) {
                break;
            }

            iov = &vec->iovs[n++];

            iov->iov_base = (void *) in->buf->pos;
            iov->iov_len = size;
        }

        prev = in->buf->pos + size;
        total += size;
    }

    vec->count = n;
    vec->size = total;

    return in;
}


ssize_t
ngx_writev(ngx_connection_t *c, ngx_iovec_t *vec)
{
    ssize_t    n;
    ngx_err_t  err;

eintr:

    n = writev(c->fd, vec->iovs, vec->count);

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "writev: %z of %uz", n, vec->size);

    if (n == -1) {
        err = ngx_errno;

        switch (err) {
        case NGX_EAGAIN:
            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
                           "writev() not ready");
            return NGX_AGAIN;

        case NGX_EINTR:
            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
                           "writev() was interrupted");
            goto eintr;

        default:
            c->write->error = 1;
            ngx_connection_error(c, err, "writev() failed");
```

```
211             return NGX_ERROR;
212         }
213     }
214
215     return n;
216 }
```

One Level Up Top Level

src/http/modules/ngx_http_ssl_module.c - nginx-1.7.10

Global variables defined

- ngx_http_ssl_commands
- ngx_http_ssl_module
- ngx_http_ssl_module_ctx
- ngx_http_ssl_protocols
- ngx_http_ssl_sess_id_ctx
- ngx_http_ssl_vars
- ngx_http_ssl_verify

Data types defined

- ngx_ssl_variable_handler_pt

Functions defined

- ngx_http_ssl_add_variables
- ngx_http_ssl_alpn_select
- ngx_http_ssl_create_srv_conf
- ngx_http_ssl_enable
- ngx_http_ssl_init
- ngx_http_ssl_merge_srv_conf
- ngx_http_ssl_npn_advertised
- ngx_http_ssl_password_file
- ngx_http_ssl_session_cache
- ngx_http_ssl_static_variable
- ngx_http_ssl_variable

Macros defined

- NGX_DEFAULT_CIPHERS
- NGX_DEFAULT_ECDH_CURVE
- NGX_HTTP_NPN_ADVERTISE

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
```

```
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_http.h>


 typedef ngx_int_t (*ngx_ssl_variable_handler_pt)(ngx_connection_t *c,
     ngx_pool_t *pool, ngx_str_t *s);


 #define NGX_DEFAULT_CIPHERS     "HIGH:!aNULL:!MD5"
 #define NGX_DEFAULT_ECDH_CURVE  "prime256v1"

 #define NGX_HTTP_NPN_ADVERTISE  "\x08http/1.1"


 #ifdef TLSEXT_TYPE_application_layer_protocol_negotiation
 static int ngx_http_ssl_alpn_select(ngx_ssl_conn_t *ssl_conn,
     const unsigned char **out, unsigned char *outlen,
     const unsigned char *in, unsigned int inlen, void *arg);
 #endif

 #ifdef TLSEXT_TYPE_next_proto_neg
 static int ngx_http_ssl_npn_advertised(ngx_ssl_conn_t *ssl_conn,
     const unsigned char **out, unsigned int *outlen, void *arg);
 #endif

 static ngx_int_t ngx_http_ssl_static_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
 static ngx_int_t ngx_http_ssl_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);

 static ngx_int_t ngx_http_ssl_add_variables(ngx_conf_t *cf);
 static void *ngx_http_ssl_create_srv_conf(ngx_conf_t *cf);
 static char *ngx_http_ssl_merge_srv_conf(ngx_conf_t *cf,
     void *parent, void *child);

 static char *ngx_http_ssl_enable(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static char *ngx_http_ssl_password_file(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static char *ngx_http_ssl_session_cache(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);

 static ngx_int_t ngx_http_ssl_init(ngx_conf_t *cf);


 static ngx_conf_bitmask_t  ngx_http_ssl_protocols[] = {
     { ngx_string("SSLv2"), NGX_SSL_SSLv2 },
     { ngx_string("SSLv3"), NGX_SSL_SSLv3 },
     { ngx_string("TLSv1"), NGX_SSL_TLSv1 },
     { ngx_string("TLSv1.1"), NGX_SSL_TLSv1_1 },
     { ngx_string("TLSv1.2"), NGX_SSL_TLSv1_2 },
     { ngx_null_string, 0 }
 };


 static ngx_conf_enum_t  ngx_http_ssl_verify[] = {
     { ngx_string("off"), 0 },
     { ngx_string("on"), 1 },
     { ngx_string("optional"), 2 },
     { ngx_string("optional_no_ca"), 3 },
     { ngx_null_string, 0 }
 };


 static ngx_command_t  ngx_http_ssl_commands[] = {

     { ngx_string("ssl"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
       ngx_http_ssl_enable,
       NGX_HTTP_SRV_CONF_OFFSET,
       offsetof(ngx_http_ssl_srv_conf_t, enable),
       NULL },
```

```
81
82      { ngx_string("ssl_certificate"),
83        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
84        ngx_conf_set_str_slot,
85        NGX_HTTP_SRV_CONF_OFFSET,
86        offsetof(ngx_http_ssl_srv_conf_t, certificate),
87        NULL },
88
89      { ngx_string("ssl_certificate_key"),
90        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
91        ngx_conf_set_str_slot,
92        NGX_HTTP_SRV_CONF_OFFSET,
93        offsetof(ngx_http_ssl_srv_conf_t, certificate_key),
94        NULL },
95
96      { ngx_string("ssl_password_file"),
97        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
98        ngx_http_ssl_password_file,
99        NGX_HTTP_SRV_CONF_OFFSET,
100         0,
101         NULL },
102
103      { ngx_string("ssl_dhparam"),
104        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
105        ngx_conf_set_str_slot,
106        NGX_HTTP_SRV_CONF_OFFSET,
107        offsetof(ngx_http_ssl_srv_conf_t, dhparam),
108        NULL },
109
110      { ngx_string("ssl_ecdh_curve"),
111        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
112        ngx_conf_set_str_slot,
113        NGX_HTTP_SRV_CONF_OFFSET,
114        offsetof(ngx_http_ssl_srv_conf_t, ecdh_curve),
115        NULL },
116
117      { ngx_string("ssl_protocols"),
118        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_1MORE,
119        ngx_conf_set_bitmask_slot,
120        NGX_HTTP_SRV_CONF_OFFSET,
121        offsetof(ngx_http_ssl_srv_conf_t, protocols),
122        &ngx_http_ssl_protocols },
123
124      { ngx_string("ssl_ciphers"),
125        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
126        ngx_conf_set_str_slot,
127        NGX_HTTP_SRV_CONF_OFFSET,
128        offsetof(ngx_http_ssl_srv_conf_t, ciphers),
129        NULL },
130
131      { ngx_string("ssl_buffer_size"),
132        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
133        ngx_conf_set_size_slot,
134        NGX_HTTP_SRV_CONF_OFFSET,
135        offsetof(ngx_http_ssl_srv_conf_t, buffer_size),
136        NULL },
137
138      { ngx_string("ssl_verify_client"),
139        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
140        ngx_conf_set_enum_slot,
141        NGX_HTTP_SRV_CONF_OFFSET,
142        offsetof(ngx_http_ssl_srv_conf_t, verify),
143        &ngx_http_ssl_verify },
144
145      { ngx_string("ssl_verify_depth"),
146        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
147        ngx_conf_set_num_slot,
148        NGX_HTTP_SRV_CONF_OFFSET,
149        offsetof(ngx_http_ssl_srv_conf_t, verify_depth),
150        NULL },
151
152      { ngx_string("ssl_client_certificate"),
153        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
154        ngx_conf_set_str_slot,
155        NGX_HTTP_SRV_CONF_OFFSET,
156        offsetof(ngx_http_ssl_srv_conf_t, client_certificate),
```

```
      NULL },

    { ngx_string("ssl_trusted_certificate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, trusted_certificate),
      NULL },

    { ngx_string("ssl_prefer_server_ciphers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, prefer_server_ciphers),
      NULL },

    { ngx_string("ssl_session_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE12,
      ngx_http_ssl_session_cache,
      NGX_HTTP_SRV_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("ssl_session_tickets"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, session_tickets),
      NULL },

    { ngx_string("ssl_session_ticket_key"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_array_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, session_ticket_keys),
      NULL },

    { ngx_string("ssl_session_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_sec_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, session_timeout),
      NULL },

    { ngx_string("ssl_crl"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, crl),
      NULL },

    { ngx_string("ssl_stapling"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, stapling),
      NULL },

    { ngx_string("ssl_stapling_file"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, stapling_file),
      NULL },

    { ngx_string("ssl_stapling_responder"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
      offsetof(ngx_http_ssl_srv_conf_t, stapling_responder),
      NULL },

    { ngx_string("ssl_stapling_verify"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_SRV_CONF_OFFSET,
```

```
      offsetof(ngx_http_ssl_srv_conf_t, stapling_verify),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_ssl_module_ctx = {
    ngx_http_ssl_add_variables,            /* preconfiguration */
    ngx_http_ssl_init,                     /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_http_ssl_create_srv_conf,          /* create server configuration */
    ngx_http_ssl_merge_srv_conf,           /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_ssl_module = {
    NGX_MODULE_V1,
    &ngx_http_ssl_module_ctx,              /* module context */
    ngx_http_ssl_commands,                 /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_variable_t  ngx_http_ssl_vars[] = {

    { ngx_string("ssl_protocol"), NULL, ngx_http_ssl_static_variable,
      (uintptr_t) ngx_ssl_get_protocol, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_cipher"), NULL, ngx_http_ssl_static_variable,
      (uintptr_t) ngx_ssl_get_cipher_name, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_session_id"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_session_id, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_session_reused"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_session_reused, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_server_name"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_server_name, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_cert"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_certificate, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_raw_cert"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_raw_certificate,
      NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_s_dn"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_subject_dn, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_i_dn"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_issuer_dn, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_serial"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_serial_number, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_fingerprint"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_fingerprint, NGX_HTTP_VAR_CHANGEABLE, 0 },

    { ngx_string("ssl_client_verify"), NULL, ngx_http_ssl_variable,
      (uintptr_t) ngx_ssl_get_client_verify, NGX_HTTP_VAR_CHANGEABLE, 0 },
```

```
309
310     { ngx_null_string, NULL, NULL, 0, 0, 0 }
311 };
312
313
314 static ngx_str_t ngx_http_ssl_sess_id_ctx = ngx_string("HTTP");
315
316
317 #ifdef TLSEXT_TYPE_application_layer_protocol_negotiation
318
319 static int
320 ngx_http_ssl_alpn_select(ngx_ssl_conn_t *ssl_conn, const unsigned char **out,
321     unsigned char *outlen, const unsigned char *in, unsigned int inlen,
322     void *arg)
323 {
324     unsigned int            srvlen;
325     unsigned char          *srv;
326 #if (NGX_DEBUG)
327     unsigned int            i;
328 #endif
329 #if (NGX_HTTP_SPDY)
330     ngx_http_connection_t  *hc;
331 #endif
332 #if (NGX_HTTP_SPDY || NGX_DEBUG)
333     ngx_connection_t       *c;
334
335     c = ngx_ssl_get_connection(ssl_conn);
336 #endif
337
338 #if (NGX_DEBUG)
339     for (i = 0; i < inlen; i += in[i] + 1) {
340         ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
341                        "SSL ALPN supported by client: %*s", in[i], &in[i + 1]);
342     }
343 #endif
344
345 #if (NGX_HTTP_SPDY)
346     hc = c->data;
347
348     if (hc->addr_conf->spdy) {
349         srv = (unsigned char *) NGX_SPDY_NPN_ADVERTISE NGX_HTTP_NPN_ADVERTISE;
350         srvlen = sizeof(NGX_SPDY_NPN_ADVERTISE NGX_HTTP_NPN_ADVERTISE) - 1;
351
352     } else
353 #endif
354     {
355         srv = (unsigned char *) NGX_HTTP_NPN_ADVERTISE;
356         srvlen = sizeof(NGX_HTTP_NPN_ADVERTISE) - 1;
357     }
358
359     if (SSL_select_next_proto((unsigned char **) out, outlen, srv, srvlen,
360                               in, inlen)
361         != OPENSSL_NPN_NEGOTIATED)
362     {
363         return SSL_TLSEXT_ERR_NOACK;
364     }
365
366     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
367                    "SSL ALPN selected: %*s", *outlen, *out);
368
369     return SSL_TLSEXT_ERR_OK;
370 }
371
372 #endif
373
374
375 #ifdef TLSEXT_TYPE_next_proto_neg
376
377 static int
378 ngx_http_ssl_npn_advertised(ngx_ssl_conn_t *ssl_conn,
379     const unsigned char **out, unsigned int *outlen, void *arg)
380 {
381 #if (NGX_HTTP_SPDY || NGX_DEBUG)
382     ngx_connection_t  *c;
383
384     c = ngx_ssl_get_connection(ssl_conn);
```

```
385     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "SSL NPN advertised");
386 #endif
387
388 #if (NGX_HTTP_SPDY)
389     {
390     ngx_http_connection_t  *hc;
391
392     hc = c->data;
393
394     if (hc->addr_conf->spdy) {
395         *out = (unsigned char *) NGX_SPDY_NPN_ADVERTISE NGX_HTTP_NPN_ADVERTISE;
396         *outlen = sizeof(NGX_SPDY_NPN_ADVERTISE NGX_HTTP_NPN_ADVERTISE) - 1;
397
398         return SSL_TLSEXT_ERR_OK;
399     }
400     }
401 #endif
402
403     *out = (unsigned char *) NGX_HTTP_NPN_ADVERTISE;
404     *outlen = sizeof(NGX_HTTP_NPN_ADVERTISE) - 1;
405
406     return SSL_TLSEXT_ERR_OK;
407 }
408
409 #endif
410
411
412 static ngx_int_t
413 ngx_http_ssl_static_variable(ngx_http_request_t *r,
414     ngx_http_variable_value_t *v, uintptr_t data)
415 {
416     ngx_ssl_variable_handler_pt  handler = (ngx_ssl_variable_handler_pt) data;
417
418     size_t     len;
419     ngx_str_t  s;
420
421     if (r->connection->ssl) {
422
423         (void) handler(r->connection, NULL, &s);
424
425         v->data = s.data;
426
427         for (len = 0; v->data[len]; len++) { /* void */ }
428
429         v->len = len;
430         v->valid = 1;
431         v->no_cacheable = 0;
432         v->not_found = 0;
433
434         return NGX_OK;
435     }
436
437     v->not_found = 1;
438
439     return NGX_OK;
440 }
441
442
443 static ngx_int_t
444 ngx_http_ssl_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
445     uintptr_t data)
446 {
447     ngx_ssl_variable_handler_pt  handler = (ngx_ssl_variable_handler_pt) data;
448
449     ngx_str_t  s;
450
451     if (r->connection->ssl) {
452
453         if (handler(r->connection, r->pool, &s) != NGX_OK) {
454             return NGX_ERROR;
455         }
456
457         v->len = s.len;
458         v->data = s.data;
459
460         if (v->len) {
```

```
            v->valid = 1;
            v->no_cacheable = 0;
            v->not_found = 0;

            return NGX_OK;
        }
    }

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_ssl_add_variables(ngx_conf_t *cf)
{
    ngx_http_variable_t  *var, *v;

    for (v = ngx_http_ssl_vars; v->name.len; v++) {
        var = ngx_http_add_variable(cf, &v->name, v->flags);
        if (var == NULL) {
            return NGX_ERROR;
        }

        var->get_handler = v->get_handler;
        var->data = v->data;
    }

    return NGX_OK;
}


static void *
ngx_http_ssl_create_srv_conf(ngx_conf_t *cf)
{
    ngx_http_ssl_srv_conf_t  *sscf;

    sscf = ngx_pcalloc(cf->pool, sizeof(ngx_http_ssl_srv_conf_t));
    if (sscf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     sscf->protocols = 0;
     *     sscf->certificate = { 0, NULL };
     *     sscf->certificate_key = { 0, NULL };
     *     sscf->dhparam = { 0, NULL };
     *     sscf->ecdh_curve = { 0, NULL };
     *     sscf->client_certificate = { 0, NULL };
     *     sscf->trusted_certificate = { 0, NULL };
     *     sscf->crl = { 0, NULL };
     *     sscf->ciphers = { 0, NULL };
     *     sscf->shm_zone = NULL;
     *     sscf->stapling_file = { 0, NULL };
     *     sscf->stapling_responder = { 0, NULL };
     */

    sscf->enable = NGX_CONF_UNSET;
    sscf->prefer_server_ciphers = NGX_CONF_UNSET;
    sscf->buffer_size = NGX_CONF_UNSET_SIZE;
    sscf->verify = NGX_CONF_UNSET_UINT;
    sscf->verify_depth = NGX_CONF_UNSET_UINT;
    sscf->passwords = NGX_CONF_UNSET_PTR;
    sscf->builtin_session_cache = NGX_CONF_UNSET;
    sscf->session_timeout = NGX_CONF_UNSET;
    sscf->session_tickets = NGX_CONF_UNSET;
    sscf->session_ticket_keys = NGX_CONF_UNSET_PTR;
    sscf->stapling = NGX_CONF_UNSET;
    sscf->stapling_verify = NGX_CONF_UNSET;

    return sscf;
}

```

```
537
538 static char *
539 ngx_http_ssl_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
540 {
541     ngx_http_ssl_srv_conf_t *prev = parent;
542     ngx_http_ssl_srv_conf_t *conf = child;
543
544     ngx_pool_cleanup_t  *cln;
545
546     if (conf->enable == NGX_CONF_UNSET) {
547         if (prev->enable == NGX_CONF_UNSET) {
548             conf->enable = 0;
549
550         } else {
551             conf->enable = prev->enable;
552             conf->file = prev->file;
553             conf->line = prev->line;
554         }
555     }
556
557     ngx_conf_merge_value(conf->session_timeout,
558                          prev->session_timeout, 300);
559
560     ngx_conf_merge_value(conf->prefer_server_ciphers,
561                          prev->prefer_server_ciphers, 0);
562
563     ngx_conf_merge_bitmask_value(conf->protocols, prev->protocols,
564                          (NGX_CONF_BITMASK_SET|NGX_SSL_SSLv3|NGX_SSL_TLSv1
565                           |NGX_SSL_TLSv1_1|NGX_SSL_TLSv1_2));
566
567     ngx_conf_merge_size_value(conf->buffer_size, prev->buffer_size,
568                          NGX_SSL_BUFSIZE);
569
570     ngx_conf_merge_uint_value(conf->verify, prev->verify, 0);
571     ngx_conf_merge_uint_value(conf->verify_depth, prev->verify_depth, 1);
572
573     ngx_conf_merge_str_value(conf->certificate, prev->certificate, "");
574     ngx_conf_merge_str_value(conf->certificate_key, prev->certificate_key, "");
575
576     ngx_conf_merge_ptr_value(conf->passwords, prev->passwords, NULL);
577
578     ngx_conf_merge_str_value(conf->dhparam, prev->dhparam, "");
579
580     ngx_conf_merge_str_value(conf->client_certificate, prev->client_certificate,
581                          "");
582     ngx_conf_merge_str_value(conf->trusted_certificate,
583                          prev->trusted_certificate, "");
584     ngx_conf_merge_str_value(conf->crl, prev->crl, "");
585
586     ngx_conf_merge_str_value(conf->ecdh_curve, prev->ecdh_curve,
587                          NGX_DEFAULT_ECDH_CURVE);
588
589     ngx_conf_merge_str_value(conf->ciphers, prev->ciphers, NGX_DEFAULT_CIPHERS);
590
591     ngx_conf_merge_value(conf->stapling, prev->stapling, 0);
592     ngx_conf_merge_value(conf->stapling_verify, prev->stapling_verify, 0);
593     ngx_conf_merge_str_value(conf->stapling_file, prev->stapling_file, "");
594     ngx_conf_merge_str_value(conf->stapling_responder,
595                          prev->stapling_responder, "");
596
597     conf->ssl.log = cf->log;
598
599     if (conf->enable) {
600
601         if (conf->certificate.len == 0) {
602             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
603                           "no \"ssl_certificate\" is defined for "
604                           "the \"ssl\" directive in %s:%ui",
605                           conf->file, conf->line);
606             return NGX_CONF_ERROR;
607         }
608
609         if (conf->certificate_key.len == 0) {
610             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
611                           "no \"ssl_certificate_key\" is defined for "
612                           "the \"ssl\" directive in %s:%ui",
```

```
                          conf->file, conf->line);
            return NGX_CONF_ERROR;
        }

    } else {

        if (conf->certificate.len == 0) {
            return NGX_CONF_OK;
        }

        if (conf->certificate_key.len == 0) {
            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                          "no \"ssl_certificate_key\" is defined "
                          "for certificate \"%V\"", &conf->certificate);
            return NGX_CONF_ERROR;
        }
    }

    if (ngx_ssl_create(&conf->ssl, conf->protocols, conf) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

#ifdef SSL_CTRL_SET_TLSEXT_HOSTNAME

    if (SSL_CTX_set_tlsext_servername_callback(conf->ssl.ctx,
                                               ngx_http_ssl_servername)
        == 0)
    {
        ngx_log_error(NGX_LOG_WARN, cf->log, 0,
            "nginx was built with SNI support, however, now it is linked "
            "dynamically to an OpenSSL library which has no tlsext support, "
            "therefore SNI is not available");
    }

#endif

#ifdef TLSEXT_TYPE_application_layer_protocol_negotiation
    SSL_CTX_set_alpn_select_cb(conf->ssl.ctx, ngx_http_ssl_alpn_select, NULL);
#endif

#ifdef TLSEXT_TYPE_next_proto_neg
    SSL_CTX_set_next_protos_advertised_cb(conf->ssl.ctx,
                                          ngx_http_ssl_npn_advertised, NULL);
#endif

    cln = ngx_pool_cleanup_add(cf->pool, 0);
    if (cln == NULL) {
        return NGX_CONF_ERROR;
    }

    cln->handler = ngx_ssl_cleanup_ctx;
    cln->data = &conf->ssl;

    if (ngx_ssl_certificate(cf, &conf->ssl, &conf->certificate,
                            &conf->certificate_key, conf->passwords)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    if (SSL_CTX_set_cipher_list(conf->ssl.ctx,
                                (const char *) conf->ciphers.data)
        == 0)
    {
        ngx_ssl_error(NGX_LOG_EMERG, cf->log, 0,
                      "SSL_CTX_set_cipher_list(\"%V\") failed",
                      &conf->ciphers);
        return NGX_CONF_ERROR;
    }

    conf->ssl.buffer_size = conf->buffer_size;

    if (conf->verify) {

        if (conf->client_certificate.len == 0 && conf->verify != 3) {
            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
```

```
                          "no ssl_client_certificate for ssl_client_verify");
            return NGX_CONF_ERROR;
        }

        if (ngx_ssl_client_certificate(cf, &conf->ssl,
                                       &conf->client_certificate,
                                       conf->verify_depth)
            != NGX_OK)
        {
            return NGX_CONF_ERROR;
        }
    }

    if (ngx_ssl_trusted_certificate(cf, &conf->ssl,
                                    &conf->trusted_certificate,
                                    conf->verify_depth)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    if (ngx_ssl_crl(cf, &conf->ssl, &conf->crl) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (conf->prefer_server_ciphers) {
        SSL_CTX_set_options(conf->ssl.ctx, SSL_OP_CIPHER_SERVER_PREFERENCE);
    }

    /* a temporary 512-bit RSA key is required for export versions of MSIE */
    SSL_CTX_set_tmp_rsa_callback(conf->ssl.ctx, ngx_ssl_rsa512_key_callback);

    if (ngx_ssl_dhparam(cf, &conf->ssl, &conf->dhparam) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (ngx_ssl_ecdh_curve(cf, &conf->ssl, &conf->ecdh_curve) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_value(conf->builtin_session_cache,
                         prev->builtin_session_cache, NGX_SSL_NONE_SCACHE);

    if (conf->shm_zone == NULL) {
        conf->shm_zone = prev->shm_zone;
    }

    if (ngx_ssl_session_cache(&conf->ssl, &ngx_http_ssl_sess_id_ctx,
                              conf->builtin_session_cache,
                              conf->shm_zone, conf->session_timeout)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_value(conf->session_tickets, prev->session_tickets, 1);

#ifdef SSL_OP_NO_TICKET
    if (!conf->session_tickets) {
        SSL_CTX_set_options(conf->ssl.ctx, SSL_OP_NO_TICKET);
    }
#endif

    ngx_conf_merge_ptr_value(conf->session_ticket_keys,
                         prev->session_ticket_keys, NULL);

    if (ngx_ssl_session_ticket_keys(cf, &conf->ssl, conf->session_ticket_keys)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    if (conf->stapling) {

        if (ngx_ssl_stapling(cf, &conf->ssl, &conf->stapling_file,
                             &conf->stapling_responder, conf->stapling_verify)
```

```
765             != NGX_OK)
766         {
767             return NGX_CONF_ERROR;
768         }
769
770     }
771
772     return NGX_CONF_OK;
773 }
774
775
776 static char *
777 ngx_http_ssl_enable(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
778 {
779     ngx_http_ssl_srv_conf_t *sscf = conf;
780
781     char  *rv;
782
783     rv = ngx_conf_set_flag_slot(cf, cmd, conf);
784
785     if (rv != NGX_CONF_OK) {
786         return rv;
787     }
788
789     sscf->file = cf->conf_file->file.name.data;
790     sscf->line = cf->conf_file->line;
791
792     return NGX_CONF_OK;
793 }
794
795
796 static char *
797 ngx_http_ssl_password_file(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
798 {
799     ngx_http_ssl_srv_conf_t *sscf = conf;
800
801     ngx_str_t  *value;
802
803     if (sscf->passwords != NGX_CONF_UNSET_PTR) {
804         return "is duplicate";
805     }
806
807     value = cf->args->elts;
808
809     sscf->passwords = ngx_ssl_read_password_file(cf, &value[1]);
810
811     if (sscf->passwords == NULL) {
812         return NGX_CONF_ERROR;
813     }
814
815     return NGX_CONF_OK;
816 }
817
818
819 static char *
820 ngx_http_ssl_session_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
821 {
822     ngx_http_ssl_srv_conf_t *sscf = conf;
823
824     size_t       len;
825     ngx_str_t   *value, name, size;
826     ngx_int_t    n;
827     ngx_uint_t   i, j;
828
829     value = cf->args->elts;
830
831     for (i = 1; i < cf->args->nelts; i++) {
832
833         if (ngx_strcmp(value[i].data, "off") == 0) {
834             sscf->builtin_session_cache = NGX_SSL_NO_SCACHE;
835             continue;
836         }
837
838         if (ngx_strcmp(value[i].data, "none") == 0) {
839             sscf->builtin_session_cache = NGX_SSL_NONE_SCACHE;
840             continue;
```

```
        }

        if (ngx_strcmp(value[i].data, "builtin") == 0) {
            sscf->builtin_session_cache = NGX_SSL_DFLT_BUILTIN_SCACHE;
            continue;
        }

        if (value[i].len > sizeof("builtin:") - 1
            && ngx_strncmp(value[i].data, "builtin:", sizeof("builtin:") - 1)
               == 0)
        {
            n = ngx_atoi(value[i].data + sizeof("builtin:") - 1,
                         value[i].len - (sizeof("builtin:") - 1));

            if (n == NGX_ERROR) {
                goto invalid;
            }

            sscf->builtin_session_cache = n;

            continue;
        }

        if (value[i].len > sizeof("shared:") - 1
            && ngx_strncmp(value[i].data, "shared:", sizeof("shared:") - 1)
               == 0)
        {
            len = 0;

            for (j = sizeof("shared:") - 1; j < value[i].len; j++) {
                if (value[i].data[j] == ':') {
                    break;
                }

                len++;
            }

            if (len == 0) {
                goto invalid;
            }

            name.len = len;
            name.data = value[i].data + sizeof("shared:") - 1;

            size.len = value[i].len - j - 1;
            size.data = name.data + len + 1;

            n = ngx_parse_size(&size);

            if (n == NGX_ERROR) {
                goto invalid;
            }

            if (n < (ngx_int_t) (8 * ngx_pagesize)) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "session cache \"%V\" is too small",
                                   &value[i]);

                return NGX_CONF_ERROR;
            }

            sscf->shm_zone = ngx_shared_memory_add(cf, &name, n,
                                                   &ngx_http_ssl_module);
            if (sscf->shm_zone == NULL) {
                return NGX_CONF_ERROR;
            }

            sscf->shm_zone->init = ngx_ssl_session_cache_init;

            continue;
        }

        goto invalid;
    }

    if (sscf->shm_zone && sscf->builtin_session_cache == NGX_CONF_UNSET) {
```

```
        sscf->builtin_session_cache = NGX_SSL_NO_BUILTIN_SCACHE;
    }

    return NGX_CONF_OK;

invalid:

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "invalid session cache \"%V\"", &value[i]);

    return NGX_CONF_ERROR;
}


static ngx_int_t
ngx_http_ssl_init(ngx_conf_t *cf)
{
    ngx_uint_t                  s;
    ngx_http_ssl_srv_conf_t    *sscf;
    ngx_http_core_loc_conf_t   *clcf;
    ngx_http_core_srv_conf_t  **cscfp;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
    cscfp = cmcf->servers.elts;

    for (s = 0; s < cmcf->servers.nelts; s++) {

        sscf = cscfp[s]->ctx->srv_conf[ngx_http_ssl_module.ctx_index];

        if (sscf->ssl.ctx == NULL || !sscf->stapling) {
            continue;
        }

        clcf = cscfp[s]->ctx->loc_conf[ngx_http_core_module.ctx_index];

        if (ngx_ssl_stapling_resolver(cf, &sscf->ssl, clcf->resolver,
                                      clcf->resolver_timeout)
            != NGX_OK)
        {
            return NGX_ERROR;
        }
    }

    return NGX_OK;
}
```

One Level Up Top Level

src/mail/ngx_mail_ssl_module.c - nginx-1.7.10

Global variables defined

- ngx_mail_ssl_commands
- ngx_mail_ssl_module
- ngx_mail_ssl_module_ctx
- ngx_mail_ssl_protocols
- ngx_mail_ssl_sess_id_ctx
- ngx_mail_starttls_state

Functions defined

- ngx_mail_ssl_create_conf
- ngx_mail_ssl_enable
- ngx_mail_ssl_merge_conf
- ngx_mail_ssl_password_file
- ngx_mail_ssl_session_cache
- ngx_mail_ssl_starttls

Macros defined

- NGX_DEFAULT_CIPHERS
- NGX_DEFAULT_ECDH_CURVE

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_mail.h>
11
12
13  #define NGX_DEFAULT_CIPHERS     "HIGH:!aNULL:!MD5"
14  #define NGX_DEFAULT_ECDH_CURVE  "prime256v1"
15
16
17  static void *ngx_mail_ssl_create_conf(ngx_conf_t *cf);
18  static char *ngx_mail_ssl_merge_conf(ngx_conf_t *cf, void *parent, void *child);
19
20  static char *ngx_mail_ssl_enable(ngx_conf_t *cf, ngx_command_t *cmd,
21      void *conf);
22  static char *ngx_mail_ssl_starttls(ngx_conf_t *cf, ngx_command_t *cmd,
23      void *conf);
24  static char *ngx_mail_ssl_password_file(ngx_conf_t *cf, ngx_command_t *cmd,
25      void *conf);
26  static char *ngx_mail_ssl_session_cache(ngx_conf_t *cf, ngx_command_t *cmd,
```

```
    void *conf);


static ngx_conf_enum_t  ngx_mail_starttls_state[] = {
    { ngx_string("off"), NGX_MAIL_STARTTLS_OFF },
    { ngx_string("on"), NGX_MAIL_STARTTLS_ON },
    { ngx_string("only"), NGX_MAIL_STARTTLS_ONLY },
    { ngx_null_string, 0 }
};



static ngx_conf_bitmask_t  ngx_mail_ssl_protocols[] = {
    { ngx_string("SSLv2"), NGX_SSL_SSLv2 },
    { ngx_string("SSLv3"), NGX_SSL_SSLv3 },
    { ngx_string("TLSv1"), NGX_SSL_TLSv1 },
    { ngx_string("TLSv1.1"), NGX_SSL_TLSv1_1 },
    { ngx_string("TLSv1.2"), NGX_SSL_TLSv1_2 },
    { ngx_null_string, 0 }
};


static ngx_command_t  ngx_mail_ssl_commands[] = {

    { ngx_string("ssl"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_mail_ssl_enable,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, enable),
      NULL },

    { ngx_string("starttls"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_mail_ssl_starttls,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, starttls),
      ngx_mail_starttls_state },

    { ngx_string("ssl_certificate"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, certificate),
      NULL },

    { ngx_string("ssl_certificate_key"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, certificate_key),
      NULL },

    { ngx_string("ssl_password_file"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_mail_ssl_password_file,
      NGX_MAIL_SRV_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("ssl_dhparam"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, dhparam),
      NULL },

    { ngx_string("ssl_ecdh_curve"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, ecdh_curve),
      NULL },

     { ngx_string("ssl_protocols"),
       NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
       ngx_conf_set_bitmask_slot,
```

```
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, protocols),
      &ngx_mail_ssl_protocols },

    { ngx_string("ssl_ciphers"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, ciphers),
      NULL },

    { ngx_string("ssl_prefer_server_ciphers"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, prefer_server_ciphers),
      NULL },

    { ngx_string("ssl_session_cache"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE12,
      ngx_mail_ssl_session_cache,
      NGX_MAIL_SRV_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("ssl_session_tickets"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, session_tickets),
      NULL },

    { ngx_string("ssl_session_ticket_key"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_array_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, session_ticket_keys),
      NULL },

    { ngx_string("ssl_session_timeout"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_sec_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_ssl_conf_t, session_timeout),
      NULL },

      ngx_null_command
};


static ngx_mail_module_t  ngx_mail_ssl_module_ctx = {
    NULL,                                  /* protocol */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_mail_ssl_create_conf,              /* create server configuration */
    ngx_mail_ssl_merge_conf                /* merge server configuration */
};


ngx_module_t  ngx_mail_ssl_module = {
    NGX_MODULE_V1,
    &ngx_mail_ssl_module_ctx,              /* module context */
    ngx_mail_ssl_commands,                 /* module directives */
    NGX_MAIL_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};
```

```
179
180 static ngx_str_t ngx_mail_ssl_sess_id_ctx = ngx_string("MAIL");
181
182
183 static void *
184 ngx_mail_ssl_create_conf(ngx_conf_t *cf)
185 {
186     ngx_mail_ssl_conf_t  *scf;
187
188     scf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_ssl_conf_t));
189     if (scf == NULL) {
190         return NULL;
191     }
192
193     /*
194      * set by ngx_pcalloc():
195      *
196      *     scf->protocols = 0;
197      *     scf->certificate = { 0, NULL };
198      *     scf->certificate_key = { 0, NULL };
199      *     scf->dhparam = { 0, NULL };
200      *     scf->ecdh_curve = { 0, NULL };
201      *     scf->ciphers = { 0, NULL };
202      *     scf->shm_zone = NULL;
203      */
204
205     scf->enable = NGX_CONF_UNSET;
206     scf->starttls = NGX_CONF_UNSET_UINT;
207     scf->passwords = NGX_CONF_UNSET_PTR;
208     scf->prefer_server_ciphers = NGX_CONF_UNSET;
209     scf->builtin_session_cache = NGX_CONF_UNSET;
210     scf->session_timeout = NGX_CONF_UNSET;
211     scf->session_tickets = NGX_CONF_UNSET;
212     scf->session_ticket_keys = NGX_CONF_UNSET_PTR;
213
214     return scf;
215 }
216
217
218 static char *
219 ngx_mail_ssl_merge_conf(ngx_conf_t *cf, void *parent, void *child)
220 {
221     ngx_mail_ssl_conf_t *prev = parent;
222     ngx_mail_ssl_conf_t *conf = child;
223
224     char                *mode;
225     ngx_pool_cleanup_t  *cln;
226
227     ngx_conf_merge_value(conf->enable, prev->enable, 0);
228     ngx_conf_merge_uint_value(conf->starttls, prev->starttls,
229                               NGX_MAIL_STARTTLS_OFF);
230
231     ngx_conf_merge_value(conf->session_timeout,
232                               prev->session_timeout, 300);
233
234     ngx_conf_merge_value(conf->prefer_server_ciphers,
235                               prev->prefer_server_ciphers, 0);
236
237     ngx_conf_merge_bitmask_value(conf->protocols, prev->protocols,
238                               (NGX_CONF_BITMASK_SET|NGX_SSL_SSLv3|NGX_SSL_TLSv1
239                                |NGX_SSL_TLSv1_1|NGX_SSL_TLSv1_2));
240
241     ngx_conf_merge_str_value(conf->certificate, prev->certificate, "");
242     ngx_conf_merge_str_value(conf->certificate_key, prev->certificate_key, "");
243
244     ngx_conf_merge_ptr_value(conf->passwords, prev->passwords, NULL);
245
246     ngx_conf_merge_str_value(conf->dhparam, prev->dhparam, "");
247
248     ngx_conf_merge_str_value(conf->ecdh_curve, prev->ecdh_curve,
249                               NGX_DEFAULT_ECDH_CURVE);
250
251     ngx_conf_merge_str_value(conf->ciphers, prev->ciphers, NGX_DEFAULT_CIPHERS);
252
253
254     conf->ssl.log = cf->log;
```

```
255
256     if (conf->enable) {
257        mode = "ssl";
258
259     } else if (conf->starttls != NGX_MAIL_STARTTLS_OFF) {
260        mode = "starttls";
261
262     } else {
263        mode = "";
264     }
265
266     if (conf->file == NULL) {
267         conf->file = prev->file;
268         conf->line = prev->line;
269     }
270
271     if (*mode) {
272
273         if (conf->certificate.len == 0) {
274             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
275                           "no \"ssl_certificate\" is defined for "
276                           "the \"%s\" directive in %s:%ui",
277                           mode, conf->file, conf->line);
278             return NGX_CONF_ERROR;
279         }
280
281         if (conf->certificate_key.len == 0) {
282             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
283                           "no \"ssl_certificate_key\" is defined for "
284                           "the \"%s\" directive in %s:%ui",
285                           mode, conf->file, conf->line);
286             return NGX_CONF_ERROR;
287         }
288
289     } else {
290
291         if (conf->certificate.len == 0) {
292             return NGX_CONF_OK;
293         }
294
295         if (conf->certificate_key.len == 0) {
296             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
297                           "no \"ssl_certificate_key\" is defined "
298                           "for certificate \"%V\"",
299                           &conf->certificate);
300             return NGX_CONF_ERROR;
301         }
302     }
303
304     if (ngx_ssl_create(&conf->ssl, conf->protocols, NULL) != NGX_OK) {
305         return NGX_CONF_ERROR;
306     }
307
308     cln = ngx_pool_cleanup_add(cf->pool, 0);
309     if (cln == NULL) {
310         return NGX_CONF_ERROR;
311     }
312
313     cln->handler = ngx_ssl_cleanup_ctx;
314     cln->data = &conf->ssl;
315
316     if (ngx_ssl_certificate(cf, &conf->ssl, &conf->certificate,
317                             &conf->certificate_key, conf->passwords)
318         != NGX_OK)
319     {
320         return NGX_CONF_ERROR;
321     }
322
323     if (SSL_CTX_set_cipher_list(conf->ssl.ctx,
324                                 (const char *) conf->ciphers.data)
325         == 0)
326     {
327         ngx_ssl_error(NGX_LOG_EMERG, cf->log, 0,
328                       "SSL_CTX_set_cipher_list(\"%V\") failed",
329                       &conf->ciphers);
330         return NGX_CONF_ERROR;
```

```
    }

    if (conf->prefer_server_ciphers) {
        SSL_CTX_set_options(conf->ssl.ctx, SSL_OP_CIPHER_SERVER_PREFERENCE);
    }

    SSL_CTX_set_tmp_rsa_callback(conf->ssl.ctx, ngx_ssl_rsa512_key_callback);

    if (ngx_ssl_dhparam(cf, &conf->ssl, &conf->dhparam) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (ngx_ssl_ecdh_curve(cf, &conf->ssl, &conf->ecdh_curve) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_value(conf->builtin_session_cache,
                         prev->builtin_session_cache, NGX_SSL_NONE_SCACHE);

    if (conf->shm_zone == NULL) {
        conf->shm_zone = prev->shm_zone;
    }

    if (ngx_ssl_session_cache(&conf->ssl, &ngx_mail_ssl_sess_id_ctx,
                              conf->builtin_session_cache,
                              conf->shm_zone, conf->session_timeout)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_value(conf->session_tickets,
                         prev->session_tickets, 1);

#ifdef SSL_OP_NO_TICKET
    if (!conf->session_tickets) {
        SSL_CTX_set_options(conf->ssl.ctx, SSL_OP_NO_TICKET);
    }
#endif

    ngx_conf_merge_ptr_value(conf->session_ticket_keys,
                         prev->session_ticket_keys, NULL);

    if (ngx_ssl_session_ticket_keys(cf, &conf->ssl, conf->session_ticket_keys)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static char *
ngx_mail_ssl_enable(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_ssl_conf_t  *scf = conf;

    char  *rv;

    rv = ngx_conf_set_flag_slot(cf, cmd, conf);

    if (rv != NGX_CONF_OK) {
        return rv;
    }

    if (scf->enable && (ngx_int_t) scf->starttls > NGX_MAIL_STARTTLS_OFF) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "\"starttls\" directive conflicts with \"ssl on\"");
        return NGX_CONF_ERROR;
    }

    scf->file = cf->conf_file->file.name.data;
    scf->line = cf->conf_file->line;

    return NGX_CONF_OK;
```

```
}


static char *
ngx_mail_ssl_starttls(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_ssl_conf_t  *scf = conf;

    char  *rv;

    rv = ngx_conf_set_enum_slot(cf, cmd, conf);

    if (rv != NGX_CONF_OK) {
        return rv;
    }

    if (scf->enable == 1 && (ngx_int_t) scf->starttls > NGX_MAIL_STARTTLS_OFF) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "\"ssl\" directive conflicts with \"starttls\"");
        return NGX_CONF_ERROR;
    }

    scf->file = cf->conf_file->file.name.data;
    scf->line = cf->conf_file->line;

    return NGX_CONF_OK;
}


static char *
ngx_mail_ssl_password_file(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_ssl_conf_t  *scf = conf;

    ngx_str_t  *value;

    if (scf->passwords != NGX_CONF_UNSET_PTR) {
        return "is duplicate";
    }

    value = cf->args->elts;

    scf->passwords = ngx_ssl_read_password_file(cf, &value[1]);

    if (scf->passwords == NULL) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static char *
ngx_mail_ssl_session_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_ssl_conf_t  *scf = conf;

    size_t       len;
    ngx_str_t   *value, name, size;
    ngx_int_t    n;
    ngx_uint_t   i, j;

    value = cf->args->elts;

    for (i = 1; i < cf->args->nelts; i++) {

        if (ngx_strcmp(value[i].data, "off") == 0) {
            scf->builtin_session_cache = NGX_SSL_NO_SCACHE;
            continue;
        }

        if (ngx_strcmp(value[i].data, "none") == 0) {
            scf->builtin_session_cache = NGX_SSL_NONE_SCACHE;
            continue;
        }

```

```
        if (ngx_strcmp(value[i].data, "builtin") == 0) {
            scf->builtin_session_cache = NGX_SSL_DFLT_BUILTIN_SCACHE;
            continue;
        }

        if (value[i].len > sizeof("builtin:") - 1
            && ngx_strncmp(value[i].data, "builtin:", sizeof("builtin:") - 1)
               == 0)
        {
            n = ngx_atoi(value[i].data + sizeof("builtin:") - 1,
                         value[i].len - (sizeof("builtin:") - 1));

            if (n == NGX_ERROR) {
                goto invalid;
            }

            scf->builtin_session_cache = n;

            continue;
        }

        if (value[i].len > sizeof("shared:") - 1
            && ngx_strncmp(value[i].data, "shared:", sizeof("shared:") - 1)
               == 0)
        {
            len = 0;

            for (j = sizeof("shared:") - 1; j < value[i].len; j++) {
                if (value[i].data[j] == ':') {
                    break;
                }

                len++;
            }

            if (len == 0) {
                goto invalid;
            }

            name.len = len;
            name.data = value[i].data + sizeof("shared:") - 1;

            size.len = value[i].len - j - 1;
            size.data = name.data + len + 1;

            n = ngx_parse_size(&size);

            if (n == NGX_ERROR) {
                goto invalid;
            }

            if (n < (ngx_int_t) (8 * ngx_pagesize)) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "session cache \"%V\" is too small",
                                   &value[i]);

                return NGX_CONF_ERROR;
            }

            scf->shm_zone = ngx_shared_memory_add(cf, &name, n,
                                                  &ngx_mail_ssl_module);
            if (scf->shm_zone == NULL) {
                return NGX_CONF_ERROR;
            }

            scf->shm_zone->init = ngx_ssl_session_cache_init;

            continue;
        }

        goto invalid;
    }

    if (scf->shm_zone && scf->builtin_session_cache == NGX_CONF_UNSET) {
        scf->builtin_session_cache = NGX_SSL_NO_BUILTIN_SCACHE;
    }
```

```
559
560     return NGX_CONF_OK;
561
562 invalid:
563
564     ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
565                        "invalid session cache \"%V\"", &value[i]);
566
567     return NGX_CONF_ERROR;
568 }
```

One Level Up Top Level

src/mail/ - nginx-1.7.10

- ngx_mail.c
- ngx_mail.h
- ngx_mail_auth_http_module.c
- ngx_mail_core_module.c
- ngx_mail_handler.c
- ngx_mail_imap_handler.c
- ngx_mail_imap_module.c
- ngx_mail_imap_module.h
- ngx_mail_parse.c
- ngx_mail_pop3_handler.c
- ngx_mail_pop3_module.c
- ngx_mail_pop3_module.h
- ngx_mail_proxy_module.c
- ngx_mail_smtp_handler.c
- ngx_mail_smtp_module.c
- ngx_mail_smtp_module.h
- ngx_mail_ssl_module.c
- ngx_mail_ssl_module.h

src/mail/ngx_mail.c - nginx-1.7.10

Global variables defined

- ngx_mail_commands
- ngx_mail_max_module
- ngx_mail_module
- ngx_mail_module_ctx

Functions defined

- ngx_mail_add_addrs
- ngx_mail_add_addrs6
- ngx_mail_add_ports
- ngx_mail_block
- ngx_mail_cmp_conf_addrs
- ngx_mail_optimize_servers

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12
13
14  static char *ngx_mail_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
15  static ngx_int_t ngx_mail_add_ports(ngx_conf_t *cf, ngx_array_t *ports,
16      ngx_mail_listen_t *listen);
17  static char *ngx_mail_optimize_servers(ngx_conf_t *cf, ngx_array_t *ports);
18  static ngx_int_t ngx_mail_add_addrs(ngx_conf_t *cf, ngx_mail_port_t *mport,
19      ngx_mail_conf_addr_t *addr);
20  #if (NGX_HAVE_INET6)
21  static ngx_int_t ngx_mail_add_addrs6(ngx_conf_t *cf, ngx_mail_port_t *mport,
22      ngx_mail_conf_addr_t *addr);
23  #endif
24  static ngx_int_t ngx_mail_cmp_conf_addrs(const void *one, const void *two);
25
26
27  ngx_uint_t  ngx_mail_max_module;
28
29
30  static ngx_command_t  ngx_mail_commands[] = {
31
32      { ngx_string("mail"),
33        NGX_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
34        ngx_mail_block,
35        0,
36        0,
37        NULL },
38
```

```
    { ngx_string("imap"),
      NGX_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
      ngx_mail_block,
      0,
      0,
      NULL },

      ngx_null_command
};


static ngx_core_module_t  ngx_mail_module_ctx = {
    ngx_string("mail"),
    NULL,
    NULL
};


ngx_module_t  ngx_mail_module = {
    NGX_MODULE_V1,
    &ngx_mail_module_ctx,                  /* module context */
    ngx_mail_commands,                     /* module directives */
    NGX_CORE_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static char *
ngx_mail_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char                        *rv;
    ngx_uint_t                   i, m, mi, s;
    ngx_conf_t                   pcf;
    ngx_array_t                  ports;
    ngx_mail_listen_t           *listen;
    ngx_mail_module_t           *module;
    ngx_mail_conf_ctx_t         *ctx;
    ngx_mail_core_srv_conf_t   **cscfp;
    ngx_mail_core_main_conf_t   *cmcf;

    if (cmd->name.data[0] == 'i') {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "the \"imap\" directive is deprecated, "
                           "use the \"mail\" directive instead");
    }

    /* the main mail context */

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_mail_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    *(ngx_mail_conf_ctx_t **) conf = ctx;

     /* count the number of the http modules and set up their indices */

     ngx_mail_max_module = 0;
     for (m = 0; ngx_modules[m]; m++) {
         if (ngx_modules[m]->type != NGX_MAIL_MODULE) {
             continue;
         }

         ngx_modules[m]->ctx_index = ngx_mail_max_module++;
     }


     /* the mail main_conf context, it is the same in the all mail contexts */

```

```
    ctx->main_conf = ngx_pcalloc(cf->pool,
                                 sizeof(void *) * ngx_mail_max_module);
    if (ctx->main_conf == NULL) {
        return NGX_CONF_ERROR;
    }


    /*
     * the mail null srv_conf context, it is used to merge
     * the server{}s' srv_conf's
     */

    ctx->srv_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_mail_max_module);
    if (ctx->srv_conf == NULL) {
        return NGX_CONF_ERROR;
    }


    /*
     * create the main_conf's and the null srv_conf's of the all mail modules
     */

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_MAIL_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;
        mi = ngx_modules[m]->ctx_index;

        if (module->create_main_conf) {
            ctx->main_conf[mi] = module->create_main_conf(cf);
            if (ctx->main_conf[mi] == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        if (module->create_srv_conf) {
            ctx->srv_conf[mi] = module->create_srv_conf(cf);
            if (ctx->srv_conf[mi] == NULL) {
                return NGX_CONF_ERROR;
            }
        }
    }


    /* parse inside the mail{} block */

    pcf = *cf;
    cf->ctx = ctx;

    cf->module_type = NGX_MAIL_MODULE;
    cf->cmd_type = NGX_MAIL_MAIN_CONF;
    rv = ngx_conf_parse(cf, NULL);

    if (rv != NGX_CONF_OK) {
        *cf = pcf;
        return rv;
    }


    /* init mail{} main_conf's, merge the server{}s' srv_conf's */

    cmcf = ctx->main_conf[ngx_mail_core_module.ctx_index];
    cscfp = cmcf->servers.elts;

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_MAIL_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;
        mi = ngx_modules[m]->ctx_index;

        /* init mail{} main_conf's */

```

```
191         cf->ctx = ctx;
192
193         if (module->init_main_conf) {
194             rv = module->init_main_conf(cf, ctx->main_conf[mi]);
195             if (rv != NGX_CONF_OK) {
196                 *cf = pcf;
197                 return rv;
198             }
199         }
200
201         for (s = 0; s < cmcf->servers.nelts; s++) {
202
203             /* merge the server{}s' srv_conf's */
204
205             cf->ctx = cscfp[s]->ctx;
206
207             if (module->merge_srv_conf) {
208                 rv = module->merge_srv_conf(cf,
209                                             ctx->srv_conf[mi],
210                                             cscfp[s]->ctx->srv_conf[mi]);
211                 if (rv != NGX_CONF_OK) {
212                     *cf = pcf;
213                     return rv;
214                 }
215             }
216         }
217     }
218
219     *cf = pcf;
220
221
222     if (ngx_array_init(&ports, cf->temp_pool, 4, sizeof(ngx_mail_conf_port_t))
223         != NGX_OK)
224     {
225         return NGX_CONF_ERROR;
226     }
227
228     listen = cmcf->listen.elts;
229
230     for (i = 0; i < cmcf->listen.nelts; i++) {
231         if (ngx_mail_add_ports(cf, &ports, &listen[i]) != NGX_OK) {
232             return NGX_CONF_ERROR;
233         }
234     }
235
236     return ngx_mail_optimize_servers(cf, &ports);
237 }
238
239
240 static ngx_int_t
241 ngx_mail_add_ports(ngx_conf_t *cf, ngx_array_t *ports,
242     ngx_mail_listen_t *listen)
243 {
244     in_port_t              p;
245     ngx_uint_t             i;
246     struct sockaddr       *sa;
247     struct sockaddr_in    *sin;
248     ngx_mail_conf_port_t  *port;
249     ngx_mail_conf_addr_t  *addr;
250 #if (NGX_HAVE_INET6)
251     struct sockaddr_in6   *sin6;
252 #endif
253
254     sa = (struct sockaddr *) &listen->sockaddr;
255
256     switch (sa->sa_family) {
257
258 #if (NGX_HAVE_INET6)
259     case AF_INET6:
260         sin6 = (struct sockaddr_in6 *) sa;
261         p = sin6->sin6_port;
262         break;
263 #endif
264
265 #if (NGX_HAVE_UNIX_DOMAIN)
266     case AF_UNIX:
```

```
        p = 0;
        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) sa;
        p = sin->sin_port;
        break;
    }

    port = ports->elts;
    for (i = 0; i < ports->nelts; i++) {
        if (p == port[i].port && sa->sa_family == port[i].family) {

            /* a port is already in the port list */

            port = &port[i];
            goto found;
        }
    }

    /* add a port to the port list */

    port = ngx_array_push(ports);
    if (port == NULL) {
        return NGX_ERROR;
    }

    port->family = sa->sa_family;
    port->port = p;

    if (ngx_array_init(&port->addrs, cf->temp_pool, 2,
                       sizeof(ngx_mail_conf_addr_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

found:

    addr = ngx_array_push(&port->addrs);
    if (addr == NULL) {
        return NGX_ERROR;
    }

    addr->sockaddr = (struct sockaddr *) &listen->sockaddr;
    addr->socklen = listen->socklen;
    addr->ctx = listen->ctx;
    addr->bind = listen->bind;
    addr->wildcard = listen->wildcard;
    addr->so_keepalive = listen->so_keepalive;
#if (NGX_HAVE_KEEPALIVE_TUNABLE)
    addr->tcp_keepidle = listen->tcp_keepidle;
    addr->tcp_keepintvl = listen->tcp_keepintvl;
    addr->tcp_keepcnt = listen->tcp_keepcnt;
#endif
#if (NGX_MAIL_SSL)
    addr->ssl = listen->ssl;
#endif
#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    addr->ipv6only = listen->ipv6only;
#endif

    return NGX_OK;
}


static char *
ngx_mail_optimize_servers(ngx_conf_t *cf, ngx_array_t *ports)
{
    ngx_uint_t             i, p, last, bind_wildcard;
    ngx_listening_t       *ls;
    ngx_mail_port_t       *mport;
    ngx_mail_conf_port_t  *port;
    ngx_mail_conf_addr_t  *addr;

```

```
    port = ports->elts;
    for (p = 0; p < ports->nelts; p++) {

        ngx_sort(port[p].addrs.elts, (size_t) port[p].addrs.nelts,
                 sizeof(ngx_mail_conf_addr_t), ngx_mail_cmp_conf_addrs);

        addr = port[p].addrs.elts;
        last = port[p].addrs.nelts;

        /*
         * if there is the binding to the "*:port" then we need to bind()
         * to the "*:port" only and ignore the other bindings
         */

        if (addr[last - 1].wildcard) {
            addr[last - 1].bind = 1;
            bind_wildcard = 1;

        } else {
            bind_wildcard = 0;
        }

        i = 0;

        while (i < last) {

            if (bind_wildcard && !addr[i].bind) {
                i++;
                continue;
            }

            ls = ngx_create_listening(cf, addr[i].sockaddr, addr[i].socklen);
            if (ls == NULL) {
                return NGX_CONF_ERROR;
            }

            ls->addr_ntop = 1;
            ls->handler = ngx_mail_init_connection;
            ls->pool_size = 256;

            /* TODO: error_log directive */
            ls->logp = &cf->cycle->new_log;
            ls->log.data = &ls->addr_text;
            ls->log.handler = ngx_accept_log_error;

            ls->keepalive = addr[i].so_keepalive;
#if (NGX_HAVE_KEEPALIVE_TUNABLE)
            ls->keepidle = addr[i].tcp_keepidle;
            ls->keepintvl = addr[i].tcp_keepintvl;
            ls->keepcnt = addr[i].tcp_keepcnt;
#endif

#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
            ls->ipv6only = addr[i].ipv6only;
#endif

            mport = ngx_palloc(cf->pool, sizeof(ngx_mail_port_t));
            if (mport == NULL) {
                return NGX_CONF_ERROR;
            }

            ls->servers = mport;

            if (i == last - 1) {
                mport->naddrs = last;

            } else {
                mport->naddrs = 1;
                i = 0;
            }

            switch (ls->sockaddr->sa_family) {
#if (NGX_HAVE_INET6)
            case AF_INET6:
                if (ngx_mail_add_addrs6(cf, mport, addr) != NGX_OK) {
                    return NGX_CONF_ERROR;
```

```
                }
                break;
#endif
            default: /* AF_INET */
                if (ngx_mail_add_addrs(cf, mport, addr) != NGX_OK) {
                    return NGX_CONF_ERROR;
                }
                break;
            }

            addr++;
            last--;
        }
    }

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_mail_add_addrs(ngx_conf_t *cf, ngx_mail_port_t *mport,
    ngx_mail_conf_addr_t *addr)
{
    u_char              *p;
    size_t               len;
    ngx_uint_t           i;
    ngx_mail_in_addr_t  *addrs;
    struct sockaddr_in  *sin;
    u_char               buf[NGX_SOCKADDR_STRLEN];

    mport->addrs = ngx_pcalloc(cf->pool,
                               mport->naddrs * sizeof(ngx_mail_in_addr_t));
    if (mport->addrs == NULL) {
        return NGX_ERROR;
    }

    addrs = mport->addrs;

    for (i = 0; i < mport->naddrs; i++) {

        sin = (struct sockaddr_in *) addr[i].sockaddr;
        addrs[i].addr = sin->sin_addr.s_addr;

        addrs[i].conf.ctx = addr[i].ctx;
#if (NGX_MAIL_SSL)
        addrs[i].conf.ssl = addr[i].ssl;
#endif

        len = ngx_sock_ntop(addr[i].sockaddr, addr[i].socklen, buf,
                            NGX_SOCKADDR_STRLEN, 1);

        p = ngx_pnalloc(cf->pool, len);
        if (p == NULL) {
            return NGX_ERROR;
        }

        ngx_memcpy(p, buf, len);

        addrs[i].conf.addr_text.len = len;
        addrs[i].conf.addr_text.data = p;
    }

    return NGX_OK;
}


#if (NGX_HAVE_INET6)

static ngx_int_t
ngx_mail_add_addrs6(ngx_conf_t *cf, ngx_mail_port_t *mport,
    ngx_mail_conf_addr_t *addr)
{
    u_char               *p;
    size_t                len;
    ngx_uint_t            i;
    ngx_mail_in6_addr_t  *addrs6;
```

```
    struct sockaddr_in6  *sin6;
    u_char                buf[NGX_SOCKADDR_STRLEN];

    mport->addrs = ngx_pcalloc(cf->pool,
                               mport->naddrs * sizeof(ngx_mail_in6_addr_t));
    if (mport->addrs == NULL) {
        return NGX_ERROR;
    }

    addrs6 = mport->addrs;

    for (i = 0; i < mport->naddrs; i++) {

        sin6 = (struct sockaddr_in6 *) addr[i].sockaddr;
        addrs6[i].addr6 = sin6->sin6_addr;

        addrs6[i].conf.ctx = addr[i].ctx;
#if (NGX_MAIL_SSL)
        addrs6[i].conf.ssl = addr[i].ssl;
#endif

        len = ngx_sock_ntop(addr[i].sockaddr, addr[i].socklen, buf,
                            NGX_SOCKADDR_STRLEN, 1);

        p = ngx_pnalloc(cf->pool, len);
        if (p == NULL) {
            return NGX_ERROR;
        }

        ngx_memcpy(p, buf, len);

        addrs6[i].conf.addr_text.len = len;
        addrs6[i].conf.addr_text.data = p;
    }

    return NGX_OK;
}

#endif


static ngx_int_t
ngx_mail_cmp_conf_addrs(const void *one, const void *two)
{
    ngx_mail_conf_addr_t  *first, *second;

    first = (ngx_mail_conf_addr_t *) one;
    second = (ngx_mail_conf_addr_t *) two;

    if (first->wildcard) {
        /* a wildcard must be the last resort, shift it to the end */
        return 1;
    }

    if (second->wildcard) {
        /* a wildcard must be the last resort, shift it to the end */
        return -1;
    }

    if (first->bind && !second->bind) {
        /* shift explicit bind()ed addresses to the start */
        return -1;
    }

    if (!first->bind && second->bind) {
        /* shift explicit bind()ed addresses to the start */
        return 1;
    }

    /* do not sort by default */

    return 0;
}
```

One Level Up Top Level

src/mail/ngx_mail.h - nginx-1.7.10

Data types defined

- ngx_imap_state_e
- ngx_mail_addr_conf_t
- ngx_mail_auth_state_pt
- ngx_mail_conf_addr_t
- ngx_mail_conf_ctx_t
- ngx_mail_conf_port_t
- ngx_mail_core_main_conf_t
- ngx_mail_core_srv_conf_t
- ngx_mail_in6_addr_t
- ngx_mail_in_addr_t
- ngx_mail_init_protocol_pt
- ngx_mail_init_session_pt
- ngx_mail_listen_t
- ngx_mail_log_ctx_t
- ngx_mail_module_t
- ngx_mail_parse_command_pt
- ngx_mail_port_t
- ngx_mail_protocol_s
- ngx_mail_protocol_t
- ngx_mail_proxy_ctx_t
- ngx_mail_session_t
- ngx_pop3_state_e
- ngx_smtp_state_e

Macros defined

- NGX_IMAP_AUTHENTICATE
- NGX_IMAP_CAPABILITY
- NGX_IMAP_LOGIN
- NGX_IMAP_LOGOUT
- NGX_IMAP_NEXT

- NGX_IMAP_NOOP
- NGX_IMAP_STARTTLS
- NGX_MAIL_AUTH_APOP
- NGX_MAIL_AUTH_APOP_ENABLED
- NGX_MAIL_AUTH_CRAM_MD5
- NGX_MAIL_AUTH_CRAM_MD5_ENABLED
- NGX_MAIL_AUTH_LOGIN
- NGX_MAIL_AUTH_LOGIN_ENABLED
- NGX_MAIL_AUTH_LOGIN_USERNAME
- NGX_MAIL_AUTH_NONE
- NGX_MAIL_AUTH_NONE_ENABLED
- NGX_MAIL_AUTH_PLAIN
- NGX_MAIL_AUTH_PLAIN_ENABLED
- NGX_MAIL_IMAP_PROTOCOL
- NGX_MAIL_MAIN_CONF
- NGX_MAIL_MAIN_CONF_OFFSET
- NGX_MAIL_MODULE
- NGX_MAIL_PARSE_INVALID_COMMAND
- NGX_MAIL_POP3_PROTOCOL
- NGX_MAIL_SMTP_PROTOCOL
- NGX_MAIL_SRV_CONF
- NGX_MAIL_SRV_CONF_OFFSET
- NGX_POP3_APOP
- NGX_POP3_AUTH
- NGX_POP3_CAPA
- NGX_POP3_DELE
- NGX_POP3_LIST
- NGX_POP3_NOOP
- NGX_POP3_PASS
- NGX_POP3_QUIT
- NGX_POP3_RETR
- NGX_POP3_RSET
- NGX_POP3_STAT

- NGX_POP3_STLS
- NGX_POP3_TOP
- NGX_POP3_UIDL
- NGX_POP3_USER
- NGX_SMTP_AUTH
- NGX_SMTP_DATA
- NGX_SMTP_EHLO
- NGX_SMTP_EXPN
- NGX_SMTP_HELO
- NGX_SMTP_HELP
- NGX_SMTP_MAIL
- NGX_SMTP_NOOP
- NGX_SMTP_QUIT
- NGX_SMTP_RCPT
- NGX_SMTP_RSET
- NGX_SMTP_STARTTLS
- NGX_SMTP_VRFY
- _NGX_MAIL_H_INCLUDED_
- ngx_mail_conf_get_module_main_conf
- ngx_mail_conf_get_module_srv_conf
- ngx_mail_delete_ctx
- ngx_mail_get_module_ctx
- ngx_mail_get_module_main_conf
- ngx_mail_get_module_srv_conf
- ngx_mail_set_ctx

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_MAIL_H_INCLUDED_
9  #define _NGX_MAIL_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15  #include <ngx_event_connect.h>
16
```

```
#if (NGX_MAIL_SSL)
#include <ngx_mail_ssl_module.h>
#endif



typedef struct {
    void                  **main_conf;
    void                  **srv_conf;
} ngx_mail_conf_ctx_t;


typedef struct {
    u_char                  sockaddr[NGX_SOCKADDRLEN];
    socklen_t               socklen;

    /* server ctx */
    ngx_mail_conf_ctx_t    *ctx;

    unsigned                bind:1;
    unsigned                wildcard:1;
#if (NGX_MAIL_SSL)
    unsigned                ssl:1;
#endif
#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    unsigned                ipv6only:1;
#endif
    unsigned                so_keepalive:2;
#if (NGX_HAVE_KEEPALIVE_TUNABLE)
    int                     tcp_keepidle;
    int                     tcp_keepintvl;
    int                     tcp_keepcnt;
#endif
} ngx_mail_listen_t;


typedef struct {
    ngx_mail_conf_ctx_t    *ctx;
    ngx_str_t               addr_text;
#if (NGX_MAIL_SSL)
    ngx_uint_t              ssl;    /* unsigned   ssl:1; */
#endif
} ngx_mail_addr_conf_t;

typedef struct {
    in_addr_t               addr;
    ngx_mail_addr_conf_t    conf;
} ngx_mail_in_addr_t;


#if (NGX_HAVE_INET6)

typedef struct {
    struct in6_addr         addr6;
    ngx_mail_addr_conf_t    conf;
} ngx_mail_in6_addr_t;

#endif


typedef struct {
    /* ngx_mail_in_addr_t or ngx_mail_in6_addr_t */
    void                   *addrs;
    ngx_uint_t              naddrs;
} ngx_mail_port_t;


typedef struct {
    int                     family;
    in_port_t               port;
    ngx_array_t             addrs;       /* array of ngx_mail_conf_addr_t */
} ngx_mail_conf_port_t;


typedef struct {
    struct sockaddr        *sockaddr;
```

```
    socklen_t               socklen;

    ngx_mail_conf_ctx_t    *ctx;

    unsigned                bind:1;
    unsigned                wildcard:1;
#if (NGX_MAIL_SSL)
    unsigned                ssl:1;
#endif
#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    unsigned                ipv6only:1;
#endif
    unsigned                so_keepalive:2;
#if (NGX_HAVE_KEEPALIVE_TUNABLE)
    int                     tcp_keepidle;
    int                     tcp_keepintvl;
    int                     tcp_keepcnt;
#endif
} ngx_mail_conf_addr_t;


typedef struct {
    ngx_array_t             servers;     /* ngx_mail_core_srv_conf_t */
    ngx_array_t             listen;      /* ngx_mail_listen_t */
} ngx_mail_core_main_conf_t;


#define NGX_MAIL_POP3_PROTOCOL  0
#define NGX_MAIL_IMAP_PROTOCOL  1
#define NGX_MAIL_SMTP_PROTOCOL  2


typedef struct ngx_mail_protocol_s  ngx_mail_protocol_t;


typedef struct {
    ngx_mail_protocol_t    *protocol;

    ngx_msec_t              timeout;
    ngx_msec_t              resolver_timeout;

    ngx_flag_t              so_keepalive;

    ngx_str_t               server_name;

    u_char                 *file_name;
    ngx_int_t               line;

    ngx_resolver_t         *resolver;

    /* server ctx */
    ngx_mail_conf_ctx_t    *ctx;
} ngx_mail_core_srv_conf_t;


typedef enum {
    ngx_pop3_start = 0,
    ngx_pop3_user,
    ngx_pop3_passwd,
    ngx_pop3_auth_login_username,
    ngx_pop3_auth_login_password,
    ngx_pop3_auth_plain,
    ngx_pop3_auth_cram_md5
} ngx_pop3_state_e;


typedef enum {
    ngx_imap_start = 0,
    ngx_imap_auth_login_username,
    ngx_imap_auth_login_password,
    ngx_imap_auth_plain,
    ngx_imap_auth_cram_md5,
    ngx_imap_login,
    ngx_imap_user,
    ngx_imap_passwd
} ngx_imap_state_e;
```

```
169
170
171 typedef enum {
172     ngx_smtp_start = 0,
173     ngx_smtp_auth_login_username,
174     ngx_smtp_auth_login_password,
175     ngx_smtp_auth_plain,
176     ngx_smtp_auth_cram_md5,
177     ngx_smtp_helo,
178     ngx_smtp_helo_xclient,
179     ngx_smtp_helo_from,
180     ngx_smtp_xclient,
181     ngx_smtp_xclient_from,
182     ngx_smtp_xclient_helo,
183     ngx_smtp_from,
184     ngx_smtp_to
185 } ngx_smtp_state_e;
186
187
188 typedef struct {
189     ngx_peer_connection_t   upstream;
190     ngx_buf_t              *buffer;
191 } ngx_mail_proxy_ctx_t;
192
193
194 typedef struct {
195     uint32_t                signature;         /* "MAIL" */
196
197     ngx_connection_t       *connection;
198
199     ngx_str_t               out;
200     ngx_buf_t              *buffer;
201
202     void                   **ctx;
203     void                   **main_conf;
204     void                   **srv_conf;
205
206     ngx_resolver_ctx_t     *resolver_ctx;
207
208     ngx_mail_proxy_ctx_t   *proxy;
209
210     ngx_uint_t              mail_state;
211
212     unsigned                protocol:3;
213     unsigned                blocked:1;
214     unsigned                quit:1;
215     unsigned                quoted:1;
216     unsigned                backslash:1;
217     unsigned                no_sync_literal:1;
218     unsigned                starttls:1;
219     unsigned                esmtp:1;
220     unsigned                auth_method:3;
221     unsigned                auth_wait:1;
222
223     ngx_str_t               login;
224     ngx_str_t               passwd;
225
226     ngx_str_t               salt;
227     ngx_str_t               tag;
228     ngx_str_t               tagged_line;
229     ngx_str_t               text;
230
231     ngx_str_t              *addr_text;
232     ngx_str_t               host;
233     ngx_str_t               smtp_helo;
234     ngx_str_t               smtp_from;
235     ngx_str_t               smtp_to;
236
237     ngx_str_t               cmd;
238
239     ngx_uint_t              command;
240     ngx_array_t             args;
241
242     ngx_uint_t              login_attempt;
243
244     /* used to parse POP3/IMAP/SMTP command */
```

```
    ngx_uint_t              state;
    u_char                 *cmd_start;
    u_char                 *arg_start;
    u_char                 *arg_end;
    ngx_uint_t              literal_len;
} ngx_mail_session_t;


typedef struct {
    ngx_str_t              *client;
    ngx_mail_session_t     *session;
} ngx_mail_log_ctx_t;


#define NGX_POP3_USER          1
#define NGX_POP3_PASS          2
#define NGX_POP3_CAPA          3
#define NGX_POP3_QUIT          4
#define NGX_POP3_NOOP          5
#define NGX_POP3_STLS          6
#define NGX_POP3_APOP          7
#define NGX_POP3_AUTH          8
#define NGX_POP3_STAT          9
#define NGX_POP3_LIST          10
#define NGX_POP3_RETR          11
#define NGX_POP3_DELE          12
#define NGX_POP3_RSET          13
#define NGX_POP3_TOP           14
#define NGX_POP3_UIDL          15


#define NGX_IMAP_LOGIN         1
#define NGX_IMAP_LOGOUT        2
#define NGX_IMAP_CAPABILITY    3
#define NGX_IMAP_NOOP          4
#define NGX_IMAP_STARTTLS      5

#define NGX_IMAP_NEXT          6

#define NGX_IMAP_AUTHENTICATE  7


#define NGX_SMTP_HELO          1
#define NGX_SMTP_EHLO          2
#define NGX_SMTP_AUTH          3
#define NGX_SMTP_QUIT          4
#define NGX_SMTP_NOOP          5
#define NGX_SMTP_MAIL          6
#define NGX_SMTP_RSET          7
#define NGX_SMTP_RCPT          8
#define NGX_SMTP_DATA          9
#define NGX_SMTP_VRFY          10
#define NGX_SMTP_EXPN          11
#define NGX_SMTP_HELP          12
#define NGX_SMTP_STARTTLS      13


#define NGX_MAIL_AUTH_PLAIN             0
#define NGX_MAIL_AUTH_LOGIN             1
#define NGX_MAIL_AUTH_LOGIN_USERNAME    2
#define NGX_MAIL_AUTH_APOP              3
#define NGX_MAIL_AUTH_CRAM_MD5          4
#define NGX_MAIL_AUTH_NONE              5


#define NGX_MAIL_AUTH_PLAIN_ENABLED     0x0002
#define NGX_MAIL_AUTH_LOGIN_ENABLED     0x0004
#define NGX_MAIL_AUTH_APOP_ENABLED      0x0008
#define NGX_MAIL_AUTH_CRAM_MD5_ENABLED  0x0010
#define NGX_MAIL_AUTH_NONE_ENABLED      0x0020


#define NGX_MAIL_PARSE_INVALID_COMMAND  20
```

```
typedef void (*ngx_mail_init_session_pt)(ngx_mail_session_t *s,
    ngx_connection_t *c);
typedef void (*ngx_mail_init_protocol_pt)(ngx_event_t *rev);
typedef void (*ngx_mail_auth_state_pt)(ngx_event_t *rev);
typedef ngx_int_t (*ngx_mail_parse_command_pt)(ngx_mail_session_t *s);


struct ngx_mail_protocol_s {
    ngx_str_t                  name;
    in_port_t                  port[4];
    ngx_uint_t                 type;

    ngx_mail_init_session_pt   init_session;
    ngx_mail_init_protocol_pt  init_protocol;
    ngx_mail_parse_command_pt  parse_command;
    ngx_mail_auth_state_pt     auth_state;

    ngx_str_t                  internal_server_error;
};


typedef struct {
    ngx_mail_protocol_t        *protocol;

    void                       *(*create_main_conf)(ngx_conf_t *cf);
    char                       *(*init_main_conf)(ngx_conf_t *cf, void *conf);

    void                       *(*create_srv_conf)(ngx_conf_t *cf);
    char                       *(*merge_srv_conf)(ngx_conf_t *cf, void *prev,
                                      void *conf);
} ngx_mail_module_t;


#define NGX_MAIL_MODULE         0x4C49414D     /* "MAIL" */

#define NGX_MAIL_MAIN_CONF      0x02000000
#define NGX_MAIL_SRV_CONF       0x04000000


#define NGX_MAIL_MAIN_CONF_OFFSET  offsetof(ngx_mail_conf_ctx_t, main_conf)
#define NGX_MAIL_SRV_CONF_OFFSET   offsetof(ngx_mail_conf_ctx_t, srv_conf)


#define ngx_mail_get_module_ctx(s, module)     (s)->ctx[module.ctx_index]
#define ngx_mail_set_ctx(s, c, module)         s->ctx[module.ctx_index] = c;
#define ngx_mail_delete_ctx(s, module)         s->ctx[module.ctx_index] = NULL;


#define ngx_mail_get_module_main_conf(s, module)                              \
    (s)->main_conf[module.ctx_index]
#define ngx_mail_get_module_srv_conf(s, module)  (s)->srv_conf[module.ctx_index]

#define ngx_mail_conf_get_module_main_conf(cf, module)                        \
    ((ngx_mail_conf_ctx_t *) cf->ctx)->main_conf[module.ctx_index]
#define ngx_mail_conf_get_module_srv_conf(cf, module)                         \
    ((ngx_mail_conf_ctx_t *) cf->ctx)->srv_conf[module.ctx_index]


#if (NGX_MAIL_SSL)
void ngx_mail_starttls_handler(ngx_event_t *rev);
ngx_int_t ngx_mail_starttls_only(ngx_mail_session_t *s, ngx_connection_t *c);
#endif


void ngx_mail_init_connection(ngx_connection_t *c);

ngx_int_t ngx_mail_salt(ngx_mail_session_t *s, ngx_connection_t *c,
    ngx_mail_core_srv_conf_t *cscf);
ngx_int_t ngx_mail_auth_plain(ngx_mail_session_t *s, ngx_connection_t *c,
    ngx_uint_t n);
ngx_int_t ngx_mail_auth_login_username(ngx_mail_session_t *s,
    ngx_connection_t *c, ngx_uint_t n);
ngx_int_t ngx_mail_auth_login_password(ngx_mail_session_t *s,
    ngx_connection_t *c);
ngx_int_t ngx_mail_auth_cram_md5_salt(ngx_mail_session_t *s,
    ngx_connection_t *c, char *prefix, size_t len);
```

```
397 ngx_int_t ngx_mail_auth_cram_md5(ngx_mail_session_t *s, ngx_connection_t *c);
398 ngx_int_t ngx_mail_auth_parse(ngx_mail_session_t *s, ngx_connection_t *c);
399
400 void ngx_mail_send(ngx_event_t *wev);
401 ngx_int_t ngx_mail_read_command(ngx_mail_session_t *s, ngx_connection_t *c);
402 void ngx_mail_auth(ngx_mail_session_t *s, ngx_connection_t *c);
403 void ngx_mail_close_connection(ngx_connection_t *c);
404 void ngx_mail_session_internal_server_error(ngx_mail_session_t *s);
405 u_char *ngx_mail_log_error(ngx_log_t *log, u_char *buf, size_t len);
406
407
408 char *ngx_mail_capabilities(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
409
410
411 /* STUB */
412 void ngx_mail_proxy_init(ngx_mail_session_t *s, ngx_addr_t *peer);
413 void ngx_mail_auth_http_init(ngx_mail_session_t *s);
414 /**/
415
416
417 extern ngx_uint_t     ngx_mail_max_module;
418 extern ngx_module_t   ngx_mail_core_module;
419
420
421 #endif /* _NGX_MAIL_H_INCLUDED_ */
```

One Level Up Top Level

src/mail/ngx_mail_handler.c - nginx-1.7.10

Functions defined

- ngx_mail_auth
- ngx_mail_auth_cram_md5
- ngx_mail_auth_cram_md5_salt
- ngx_mail_auth_login_password
- ngx_mail_auth_login_username
- ngx_mail_auth_plain
- ngx_mail_close_connection
- ngx_mail_init_connection
- ngx_mail_init_session
- ngx_mail_log_error
- ngx_mail_read_command
- ngx_mail_salt
- ngx_mail_send
- ngx_mail_session_internal_server_error
- ngx_mail_ssl_handshake_handler
- ngx_mail_ssl_init_connection
- ngx_mail_starttls_handler
- ngx_mail_starttls_only

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12
13
14  static void ngx_mail_init_session(ngx_connection_t *c);
15
16  #if (NGX_MAIL_SSL)
17  static void ngx_mail_ssl_init_connection(ngx_ssl_t *ssl, ngx_connection_t *c);
18  static void ngx_mail_ssl_handshake_handler(ngx_connection_t *c);
19  #endif
20
21
22  void
23  ngx_mail_init_connection(ngx_connection_t *c)
24  {
```

```
    size_t                len;
    ngx_uint_t            i;
    ngx_mail_port_t      *port;
    struct sockaddr      *sa;
    struct sockaddr_in   *sin;
    ngx_mail_log_ctx_t   *ctx;
    ngx_mail_in_addr_t   *addr;
    ngx_mail_session_t   *s;
    ngx_mail_addr_conf_t *addr_conf;
    u_char                text[NGX_SOCKADDR_STRLEN];
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
    ngx_mail_in6_addr_t  *addr6;
#endif


    /* find the server configuration for the address:port */

    port = c->listening->servers;

    if (port->naddrs > 1) {

        /*
         * There are several addresses on this port and one of them
         * is the "*:port" wildcard so getsockname() is needed to determine
         * the server address.
         *
         * AcceptEx() already gave this address.
         */

        if (ngx_connection_local_sockaddr(c, NULL, 0) != NGX_OK) {
            ngx_mail_close_connection(c);
            return;
        }

        sa = c->local_sockaddr;

        switch (sa->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            sin6 = (struct sockaddr_in6 *) sa;

            addr6 = port->addrs;

            /* the last address is "*" */

            for (i = 0; i < port->naddrs - 1; i++) {
                if (ngx_memcmp(&addr6[i].addr6, &sin6->sin6_addr, 16) == 0) {
                    break;
                }
            }

            addr_conf = &addr6[i].conf;

            break;
#endif

        default: /* AF_INET */
            sin = (struct sockaddr_in *) sa;

            addr = port->addrs;

            /* the last address is "*" */

            for (i = 0; i < port->naddrs - 1; i++) {
                if (addr[i].addr == sin->sin_addr.s_addr) {
                    break;
                }
            }

            addr_conf = &addr[i].conf;

            break;
        }

```

```
    } else {
        switch (c->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            addr6 = port->addrs;
            addr_conf = &addr6[0].conf;
            break;
#endif

        default: /* AF_INET */
            addr = port->addrs;
            addr_conf = &addr[0].conf;
            break;
        }
    }

    s = ngx_pcalloc(c->pool, sizeof(ngx_mail_session_t));
    if (s == NULL) {
        ngx_mail_close_connection(c);
        return;
    }

    s->signature = NGX_MAIL_MODULE;

    s->main_conf = addr_conf->ctx->main_conf;
    s->srv_conf = addr_conf->ctx->srv_conf;

    s->addr_text = &addr_conf->addr_text;

    c->data = s;
    s->connection = c;

    len = ngx_sock_ntop(c->sockaddr, c->socklen, text, NGX_SOCKADDR_STRLEN, 1);

    ngx_log_error(NGX_LOG_INFO, c->log, 0, "*%uA client %*s connected to %V",
                  c->number, len, text, s->addr_text);

    ctx = ngx_palloc(c->pool, sizeof(ngx_mail_log_ctx_t));
    if (ctx == NULL) {
        ngx_mail_close_connection(c);
        return;
    }

    ctx->client = &c->addr_text;
    ctx->session = s;

    c->log->connection = c->number;
    c->log->handler = ngx_mail_log_error;
    c->log->data = ctx;
    c->log->action = "sending client greeting line";

    c->log_error = NGX_ERROR_INFO;

#if (NGX_MAIL_SSL)
    {
    ngx_mail_ssl_conf_t  *sslcf;

    sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);

    if (sslcf->enable) {
        c->log->action = "SSL handshaking";

        ngx_mail_ssl_init_connection(&sslcf->ssl, c);
        return;
    }

    if (addr_conf->ssl) {

        c->log->action = "SSL handshaking";

        if (sslcf->ssl.ctx == NULL) {
            ngx_log_error(NGX_LOG_ERR, c->log, 0,
                          "no \"ssl_certificate\" is defined "
                          "in server listening on SSL port");
            ngx_mail_close_connection(c);
```

```
            return;
        }

        ngx_mail_ssl_init_connection(&sslcf->ssl, c);
        return;
    }

    }
#endif

    ngx_mail_init_session(c);
}


#if (NGX_MAIL_SSL)

void
ngx_mail_starttls_handler(ngx_event_t *rev)
{
    ngx_connection_t     *c;
    ngx_mail_session_t   *s;
    ngx_mail_ssl_conf_t  *sslcf;

    c = rev->data;
    s = c->data;
    s->starttls = 1;

    c->log->action = "in starttls state";

    sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);

    ngx_mail_ssl_init_connection(&sslcf->ssl, c);
}


static void
ngx_mail_ssl_init_connection(ngx_ssl_t *ssl, ngx_connection_t *c)
{
    ngx_mail_session_t        *s;
    ngx_mail_core_srv_conf_t  *cscf;

    if (ngx_ssl_create_connection(ssl, c, 0) == NGX_ERROR) {
        ngx_mail_close_connection(c);
        return;
    }

    if (ngx_ssl_handshake(c) == NGX_AGAIN) {

        s = c->data;

        cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

        ngx_add_timer(c->read, cscf->timeout);

        c->ssl->handler = ngx_mail_ssl_handshake_handler;

        return;
    }

    ngx_mail_ssl_handshake_handler(c);
}


static void
ngx_mail_ssl_handshake_handler(ngx_connection_t *c)
{
    ngx_mail_session_t        *s;
    ngx_mail_core_srv_conf_t  *cscf;

    if (c->ssl->handshaked) {

        s = c->data;

        if (s->starttls) {
            cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

```

```
253             c->read->handler = cscf->protocol->init_protocol;
254             c->write->handler = ngx_mail_send;
255
256             cscf->protocol->init_protocol(c->read);
257
258             return;
259         }
260
261         c->read->ready = 0;
262
263         ngx_mail_init_session(c);
264         return;
265     }
266
267     ngx_mail_close_connection(c);
268 }
269
270 #endif
271
272
273 static void
274 ngx_mail_init_session(ngx_connection_t *c)
275 {
276     ngx_mail_session_t        *s;
277     ngx_mail_core_srv_conf_t  *cscf;
278
279     s = c->data;
280
281     cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);
282
283     s->protocol = cscf->protocol->type;
284
285     s->ctx = ngx_pcalloc(c->pool, sizeof(void *) * ngx_mail_max_module);
286     if (s->ctx == NULL) {
287         ngx_mail_session_internal_server_error(s);
288         return;
289     }
290
291     c->write->handler = ngx_mail_send;
292
293     cscf->protocol->init_session(s, c);
294 }
295
296
297 ngx_int_t
298 ngx_mail_salt(ngx_mail_session_t *s, ngx_connection_t *c,
299     ngx_mail_core_srv_conf_t *cscf)
300 {
301     s->salt.data = ngx_pnalloc(c->pool,
302                                sizeof(" <18446744073709551616.@>" CRLF) - 1
303                                + NGX_TIME_T_LEN
304                                + cscf->server_name.len);
305     if (s->salt.data == NULL) {
306         return NGX_ERROR;
307     }
308
309     s->salt.len = ngx_sprintf(s->salt.data, "<%ul.%T@%V>" CRLF,
310                               ngx_random(), ngx_time(), &cscf->server_name)
311                   - s->salt.data;
312
313     return NGX_OK;
314 }
315
316
317 #if (NGX_MAIL_SSL)
318
319 ngx_int_t
320 ngx_mail_starttls_only(ngx_mail_session_t *s, ngx_connection_t *c)
321 {
322     ngx_mail_ssl_conf_t  *sslcf;
323
324     if (c->ssl) {
325         return 0;
326     }
327
328     sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);
```

```
329
330     if (sslcf->starttls == NGX_MAIL_STARTTLS_ONLY) {
331         return 1;
332     }
333
334     return 0;
335 }
336
337 #endif
338
339
340 ngx_int_t
341 ngx_mail_auth_plain(ngx_mail_session_t *s, ngx_connection_t *c, ngx_uint_t n)
342 {
343     u_char     *p, *last;
344     ngx_str_t  *arg, plain;
345
346     arg = s->args.elts;
347
348 #if (NGX_DEBUG_MAIL_PASSWD)
349     ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
350                    "mail auth plain: \"%V\"", &arg[n]);
351 #endif
352
353     plain.data = ngx_pnalloc(c->pool, ngx_base64_decoded_length(arg[n].len));
354     if (plain.data == NULL) {
355         return NGX_ERROR;
356     }
357
358     if (ngx_decode_base64(&plain, &arg[n]) != NGX_OK) {
359         ngx_log_error(NGX_LOG_INFO, c->log, 0,
360             "client sent invalid base64 encoding in AUTH PLAIN command");
361         return NGX_MAIL_PARSE_INVALID_COMMAND;
362     }
363
364     p = plain.data;
365     last = p + plain.len;
366
367     while (p < last && *p++) { /* void */ }
368
369     if (p == last) {
370         ngx_log_error(NGX_LOG_INFO, c->log, 0,
371                       "client sent invalid login in AUTH PLAIN command");
372         return NGX_MAIL_PARSE_INVALID_COMMAND;
373     }
374
375     s->login.data = p;
376
377     while (p < last && *p) { p++; }
378
379     if (p == last) {
380         ngx_log_error(NGX_LOG_INFO, c->log, 0,
381                       "client sent invalid password in AUTH PLAIN command");
382         return NGX_MAIL_PARSE_INVALID_COMMAND;
383     }
384
385     s->login.len = p++ - s->login.data;
386
387     s->passwd.len = last - p;
388     s->passwd.data = p;
389
390 #if (NGX_DEBUG_MAIL_PASSWD)
391     ngx_log_debug2(NGX_LOG_DEBUG_MAIL, c->log, 0,
392                    "mail auth plain: \"%V\" \"%V\"", &s->login, &s->passwd);
393 #endif
394
395     return NGX_DONE;
396 }
397
398
399 ngx_int_t
400 ngx_mail_auth_login_username(ngx_mail_session_t *s, ngx_connection_t *c,
401     ngx_uint_t n)
402 {
403     ngx_str_t  *arg;
404
```

```
    arg = s->args.elts;

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "mail auth login username: \"%V\"", &arg[n]);

    s->login.data = ngx_pnalloc(c->pool, ngx_base64_decoded_length(arg[n].len));
    if (s->login.data == NULL) {
        return NGX_ERROR;
    }

    if (ngx_decode_base64(&s->login, &arg[n]) != NGX_OK) {
        ngx_log_error(NGX_LOG_INFO, c->log, 0,
            "client sent invalid base64 encoding in AUTH LOGIN command");
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "mail auth login username: \"%V\"", &s->login);

    return NGX_OK;
}


ngx_int_t
ngx_mail_auth_login_password(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t  *arg;

    arg = s->args.elts;

#if (NGX_DEBUG_MAIL_PASSWD)
    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "mail auth login password: \"%V\"", &arg[0]);
#endif

    s->passwd.data = ngx_pnalloc(c->pool,
                                 ngx_base64_decoded_length(arg[0].len));
    if (s->passwd.data == NULL) {
        return NGX_ERROR;
    }

    if (ngx_decode_base64(&s->passwd, &arg[0]) != NGX_OK) {
        ngx_log_error(NGX_LOG_INFO, c->log, 0,
            "client sent invalid base64 encoding in AUTH LOGIN command");
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

#if (NGX_DEBUG_MAIL_PASSWD)
    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "mail auth login password: \"%V\"", &s->passwd);
#endif

    return NGX_DONE;
}


ngx_int_t
ngx_mail_auth_cram_md5_salt(ngx_mail_session_t *s, ngx_connection_t *c,
    char *prefix, size_t len)
{
    u_char      *p;
    ngx_str_t    salt;
    ngx_uint_t   n;

    p = ngx_pnalloc(c->pool, len + ngx_base64_encoded_length(s->salt.len) + 2);
    if (p == NULL) {
        return NGX_ERROR;
    }

    salt.data = ngx_cpymem(p, prefix, len);
    s->salt.len -= 2;

    ngx_encode_base64(&salt, &s->salt);

    s->salt.len += 2;
    n = len + salt.len;
```

```
    p[n++] = CR; p[n++] = LF;

    s->out.len = n;
    s->out.data = p;

    return NGX_OK;
}


ngx_int_t
ngx_mail_auth_cram_md5(ngx_mail_session_t *s, ngx_connection_t *c)
{
    u_char     *p, *last;
    ngx_str_t  *arg;

    arg = s->args.elts;

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "mail auth cram-md5: \"%V\"", &arg[0]);

    s->login.data = ngx_pnalloc(c->pool, ngx_base64_decoded_length(arg[0].len));
    if (s->login.data == NULL) {
        return NGX_ERROR;
    }

    if (ngx_decode_base64(&s->login, &arg[0]) != NGX_OK) {
        ngx_log_error(NGX_LOG_INFO, c->log, 0,
            "client sent invalid base64 encoding in AUTH CRAM-MD5 command");
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    p = s->login.data;
    last = p + s->login.len;

    while (p < last) {
        if (*p++ == ' ') {
            s->login.len = p - s->login.data - 1;
            s->passwd.len = last - p;
            s->passwd.data = p;
            break;
        }
    }

    if (s->passwd.len != 32) {
        ngx_log_error(NGX_LOG_INFO, c->log, 0,
            "client sent invalid CRAM-MD5 hash in AUTH CRAM-MD5 command");
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "mail auth cram-md5: \"%V\" \"%V\"", &s->login, &s->passwd);

    s->auth_method = NGX_MAIL_AUTH_CRAM_MD5;

    return NGX_DONE;
}


void
ngx_mail_send(ngx_event_t *wev)
{
    ngx_int_t                  n;
    ngx_connection_t          *c;
    ngx_mail_session_t        *s;
    ngx_mail_core_srv_conf_t  *cscf;

    c = wev->data;
    s = c->data;

    if (wev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        c->timedout = 1;
        ngx_mail_close_connection(c);
        return;
    }

```

```
    if (s->out.len == 0) {
        if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
            ngx_mail_close_connection(c);
        }

        return;
    }

    n = c->send(c, s->out.data, s->out.len);

    if (n > 0) {
        s->out.data += n;
        s->out.len -= n;

        if (s->out.len != 0) {
            goto again;
        }

        if (wev->timer_set) {
            ngx_del_timer(wev);
        }

        if (s->quit) {
            ngx_mail_close_connection(c);
            return;
        }

        if (s->blocked) {
            c->read->handler(c->read);
        }

        return;
    }

    if (n == NGX_ERROR) {
        ngx_mail_close_connection(c);
        return;
    }

    /* n == NGX_AGAIN */

again:

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

    ngx_add_timer(c->write, cscf->timeout);

    if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
        ngx_mail_close_connection(c);
        return;
    }
}


ngx_int_t
ngx_mail_read_command(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ssize_t                    n;
    ngx_int_t                  rc;
    ngx_str_t                  l;
    ngx_mail_core_srv_conf_t  *cscf;

    n = c->recv(c, s->buffer->last, s->buffer->end - s->buffer->last);

    if (n == NGX_ERROR || n == 0) {
        ngx_mail_close_connection(c);
        return NGX_ERROR;
    }

    if (n > 0) {
        s->buffer->last += n;
    }

    if (n == NGX_AGAIN) {
        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            ngx_mail_session_internal_server_error(s);
```

```
            return NGX_ERROR;
        }

        if (s->buffer->pos == s->buffer->last) {
            return NGX_AGAIN;
        }
    }

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

    rc = cscf->protocol->parse_command(s);

    if (rc == NGX_AGAIN) {

        if (s->buffer->last < s->buffer->end) {
            return rc;
        }

        l.len = s->buffer->last - s->buffer->start;
        l.data = s->buffer->start;

        ngx_log_error(NGX_LOG_INFO, c->log, 0,
                      "client sent too long command \"%V\"", &l);

        s->quit = 1;

        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    if (rc == NGX_IMAP_NEXT || rc == NGX_MAIL_PARSE_INVALID_COMMAND) {
        return rc;
    }

    if (rc == NGX_ERROR) {
        ngx_mail_close_connection(c);
        return NGX_ERROR;
    }

    return NGX_OK;
}


void
ngx_mail_auth(ngx_mail_session_t *s, ngx_connection_t *c)
{
    s->args.nelts = 0;

    if (s->buffer->pos == s->buffer->last) {
        s->buffer->pos = s->buffer->start;
        s->buffer->last = s->buffer->start;
    }

    s->state = 0;

    if (c->read->timer_set) {
        ngx_del_timer(c->read);
    }

    s->login_attempt++;

    ngx_mail_auth_http_init(s);
}


void
ngx_mail_session_internal_server_error(ngx_mail_session_t *s)
{
    ngx_mail_core_srv_conf_t  *cscf;

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

    s->out = cscf->protocol->internal_server_error;
    s->quit = 1;

    ngx_mail_send(s->connection->write);
}
```

```
709
710
711 void
712 ngx_mail_close_connection(ngx_connection_t *c)
713 {
714     ngx_pool_t  *pool;
715
716     ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
717                    "close mail connection: %d", c->fd);
718
719 #if (NGX_MAIL_SSL)
720
721     if (c->ssl) {
722         if (ngx_ssl_shutdown(c) == NGX_AGAIN) {
723             c->ssl->handler = ngx_mail_close_connection;
724             return;
725         }
726     }
727
728 #endif
729
730 #if (NGX_STAT_STUB)
731     (void) ngx_atomic_fetch_add(ngx_stat_active, -1);
732 #endif
733
734     c->destroyed = 1;
735
736     pool = c->pool;
737
738     ngx_close_connection(c);
739
740     ngx_destroy_pool(pool);
741 }
742
743
744 u_char *
745 ngx_mail_log_error(ngx_log_t *log, u_char *buf, size_t len)
746 {
747     u_char              *p;
748     ngx_mail_session_t  *s;
749     ngx_mail_log_ctx_t  *ctx;
750
751     if (log->action) {
752         p = ngx_snprintf(buf, len, " while %s", log->action);
753         len -= p - buf;
754         buf = p;
755     }
756
757     ctx = log->data;
758
759     p = ngx_snprintf(buf, len, ", client: %V", ctx->client);
760     len -= p - buf;
761     buf = p;
762
763     s = ctx->session;
764
765     if (s == NULL) {
766         return p;
767     }
768
769     p = ngx_snprintf(buf, len, "%s, server: %V",
770                      s->starttls ? " using starttls" : "",
771                      s->addr_text);
772     len -= p - buf;
773     buf = p;
774
775     if (s->login.len == 0) {
776         return p;
777     }
778
779     p = ngx_snprintf(buf, len, ", login: \"%V\"", &s->login);
780     len -= p - buf;
781     buf = p;
782
783     if (s->proxy == NULL) {
784         return p;
```

```
785     }
786
787     p = ngx_snprintf(buf, len, ", upstream: %V", s->proxy->upstream.name);
788
789     return p;
790 }
```

One Level Up Top Level

src/mail/ngx_mail_ssl_module.h - nginx-1.7.10

Data types defined

- ngx_mail_ssl_conf_t

Macros defined

- NGX_MAIL_STARTTLS_OFF
- NGX_MAIL_STARTTLS_ON
- NGX_MAIL_STARTTLS_ONLY
- _NGX_MAIL_SSL_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_MAIL_SSL_H_INCLUDED_
9  #define _NGX_MAIL_SSL_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_mail.h>
15
16
17  #define NGX_MAIL_STARTTLS_OFF   0
18  #define NGX_MAIL_STARTTLS_ON    1
19  #define NGX_MAIL_STARTTLS_ONLY  2
20
21
22  typedef struct {
23      ngx_flag_t       enable;
24      ngx_flag_t       prefer_server_ciphers;
25
26      ngx_ssl_t        ssl;
27
28      ngx_uint_t       starttls;
29      ngx_uint_t       protocols;
30
31      ssize_t          builtin_session_cache;
32
33      time_t           session_timeout;
34
35      ngx_str_t        certificate;
36      ngx_str_t        certificate_key;
37      ngx_str_t        dhparam;
38      ngx_str_t        ecdh_curve;
39
40      ngx_str_t        ciphers;
41
42      ngx_array_t     *passwords;
43
44      ngx_shm_zone_t  *shm_zone;
45
46      ngx_flag_t       session_tickets;
47      ngx_array_t     *session_ticket_keys;
48
49      u_char          *file;
50      ngx_uint_t       line;
```

```
} ngx_mail_ssl_conf_t;


extern ngx_module_t  ngx_mail_ssl_module;


#endif /* _NGX_MAIL_SSL_H_INCLUDED_ */
```

One Level Up Top Level

src/mail/ngx_mail_core_module.c - nginx-1.7.10

Global variables defined

- ngx_conf_deprecated_so_keepalive
- ngx_mail_core_commands
- ngx_mail_core_module
- ngx_mail_core_module_ctx

Functions defined

- ngx_mail_capabilities
- ngx_mail_core_create_main_conf
- ngx_mail_core_create_srv_conf
- ngx_mail_core_listen
- ngx_mail_core_merge_srv_conf
- ngx_mail_core_protocol
- ngx_mail_core_resolver
- ngx_mail_core_server

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12
13
14  static void *ngx_mail_core_create_main_conf(ngx_conf_t *cf);
15  static void *ngx_mail_core_create_srv_conf(ngx_conf_t *cf);
16  static char *ngx_mail_core_merge_srv_conf(ngx_conf_t *cf, void *parent,
17      void *child);
18  static char *ngx_mail_core_server(ngx_conf_t *cf, ngx_command_t *cmd,
19      void *conf);
20  static char *ngx_mail_core_listen(ngx_conf_t *cf, ngx_command_t *cmd,
21      void *conf);
22  static char *ngx_mail_core_protocol(ngx_conf_t *cf, ngx_command_t *cmd,
23      void *conf);
24  static char *ngx_mail_core_resolver(ngx_conf_t *cf, ngx_command_t *cmd,
25      void *conf);
26
27
28  static ngx_conf_deprecated_t  ngx_conf_deprecated_so_keepalive = {
29      ngx_conf_deprecated, "so_keepalive",
30      "so_keepalive\" parameter of the \"listen"
31  };
32
33
34  static ngx_command_t  ngx_mail_core_commands[] = {
```

```
35
36      { ngx_string("server"),
37        NGX_MAIL_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
38        ngx_mail_core_server,
39        0,
40        0,
41        NULL },
42
43      { ngx_string("listen"),
44        NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
45        ngx_mail_core_listen,
46        NGX_MAIL_SRV_CONF_OFFSET,
47        0,
48        NULL },
49
50      { ngx_string("protocol"),
51        NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
52        ngx_mail_core_protocol,
53        NGX_MAIL_SRV_CONF_OFFSET,
54        0,
55        NULL },
56
57      { ngx_string("so_keepalive"),
58        NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
59        ngx_conf_set_flag_slot,
60        NGX_MAIL_SRV_CONF_OFFSET,
61        offsetof(ngx_mail_core_srv_conf_t, so_keepalive),
62        &ngx_conf_deprecated_so_keepalive },
63
64      { ngx_string("timeout"),
65        NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
66        ngx_conf_set_msec_slot,
67        NGX_MAIL_SRV_CONF_OFFSET,
68        offsetof(ngx_mail_core_srv_conf_t, timeout),
69        NULL },
70
71      { ngx_string("server_name"),
72        NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
73        ngx_conf_set_str_slot,
74        NGX_MAIL_SRV_CONF_OFFSET,
75        offsetof(ngx_mail_core_srv_conf_t, server_name),
76        NULL },
77
78      { ngx_string("resolver"),
79        NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
80        ngx_mail_core_resolver,
81        NGX_MAIL_SRV_CONF_OFFSET,
82        0,
83        NULL },
84
85      { ngx_string("resolver_timeout"),
86        NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
87        ngx_conf_set_msec_slot,
88        NGX_MAIL_SRV_CONF_OFFSET,
89        offsetof(ngx_mail_core_srv_conf_t, resolver_timeout),
90        NULL },
91
92        ngx_null_command
93  };
94
95
96  static ngx_mail_module_t  ngx_mail_core_module_ctx = {
97      NULL,                                  /* protocol */
98
99      ngx_mail_core_create_main_conf,        /* create main configuration */
100      NULL,                                  /* init main configuration */
101
102      ngx_mail_core_create_srv_conf,         /* create server configuration */
103      ngx_mail_core_merge_srv_conf           /* merge server configuration */
104  };
105
106
107  ngx_module_t  ngx_mail_core_module = {
108      NGX_MODULE_V1,
109      &ngx_mail_core_module_ctx,             /* module context */
110      ngx_mail_core_commands,                /* module directives */
```

```
    NGX_MAIL_MODULE,                     /* module type */
    NULL,                                /* init master */
    NULL,                                /* init module */
    NULL,                                /* init process */
    NULL,                                /* init thread */
    NULL,                                /* exit thread */
    NULL,                                /* exit process */
    NULL,                                /* exit master */
    NGX_MODULE_V1_PADDING
};


static void *
ngx_mail_core_create_main_conf(ngx_conf_t *cf)
{
    ngx_mail_core_main_conf_t  *cmcf;

    cmcf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_core_main_conf_t));
    if (cmcf == NULL) {
        return NULL;
    }

    if (ngx_array_init(&cmcf->servers, cf->pool, 4,
                       sizeof(ngx_mail_core_srv_conf_t *))
        != NGX_OK)
    {
        return NULL;
    }

    if (ngx_array_init(&cmcf->listen, cf->pool, 4, sizeof(ngx_mail_listen_t))
        != NGX_OK)
    {
        return NULL;
    }

    return cmcf;
}


static void *
ngx_mail_core_create_srv_conf(ngx_conf_t *cf)
{
    ngx_mail_core_srv_conf_t  *cscf;

    cscf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_core_srv_conf_t));
    if (cscf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     cscf->protocol = NULL;
     */

    cscf->timeout = NGX_CONF_UNSET_MSEC;
    cscf->resolver_timeout = NGX_CONF_UNSET_MSEC;
    cscf->so_keepalive = NGX_CONF_UNSET;

    cscf->resolver = NGX_CONF_UNSET_PTR;

    cscf->file_name = cf->conf_file->file.name.data;
    cscf->line = cf->conf_file->line;

    return cscf;
}


static char *
ngx_mail_core_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_mail_core_srv_conf_t *prev = parent;
    ngx_mail_core_srv_conf_t *conf = child;

    ngx_conf_merge_msec_value(conf->timeout, prev->timeout, 60000);
    ngx_conf_merge_msec_value(conf->resolver_timeout, prev->resolver_timeout,
```

```
                              30000);

    ngx_conf_merge_value(conf->so_keepalive, prev->so_keepalive, 0);


    ngx_conf_merge_str_value(conf->server_name, prev->server_name, "");

    if (conf->server_name.len == 0) {
        conf->server_name = cf->cycle->hostname;
    }

    if (conf->protocol == NULL) {
        ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                      "unknown mail protocol for server in %s:%ui",
                      conf->file_name, conf->line);
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_ptr_value(conf->resolver, prev->resolver, NULL);

    return NGX_CONF_OK;
}


static char *
ngx_mail_core_server(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char                        *rv;
    void                        *mconf;
    ngx_uint_t                   m;
    ngx_conf_t                   pcf;
    ngx_mail_module_t           *module;
    ngx_mail_conf_ctx_t         *ctx, *mail_ctx;
    ngx_mail_core_srv_conf_t    *cscf, **cscfp;
    ngx_mail_core_main_conf_t   *cmcf;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_mail_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    mail_ctx = cf->ctx;
    ctx->main_conf = mail_ctx->main_conf;

    /* the server{}'s srv_conf */

    ctx->srv_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_mail_max_module);
    if (ctx->srv_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_MAIL_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;

        if (module->create_srv_conf) {
            mconf = module->create_srv_conf(cf);
            if (mconf == NULL) {
                return NGX_CONF_ERROR;
            }

            ctx->srv_conf[ngx_modules[m]->ctx_index] = mconf;
        }
    }

    /* the server configuration context */

    cscf = ctx->srv_conf[ngx_mail_core_module.ctx_index];
    cscf->ctx = ctx;

    cmcf = ctx->main_conf[ngx_mail_core_module.ctx_index];

    cscfp = ngx_array_push(&cmcf->servers);
```

```
    if (cscfp == NULL) {
        return NGX_CONF_ERROR;
    }

    *cscfp = cscf;


    /* parse inside server{} */

    pcf = *cf;
    cf->ctx = ctx;
    cf->cmd_type = NGX_MAIL_SRV_CONF;

    rv = ngx_conf_parse(cf, NULL);

    *cf = pcf;

    return rv;
}


static char *
ngx_mail_core_listen(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_core_srv_conf_t  *cscf = conf;

    size_t                     len, off;
    in_port_t                  port;
    ngx_str_t                 *value;
    ngx_url_t                  u;
    ngx_uint_t                 i, m;
    struct sockaddr           *sa;
    ngx_mail_listen_t         *ls;
    ngx_mail_module_t         *module;
    struct sockaddr_in        *sin;
    ngx_mail_core_main_conf_t  *cmcf;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6       *sin6;
#endif

    value = cf->args->elts;

    ngx_memzero(&u, sizeof(ngx_url_t));

    u.url = value[1];
    u.listen = 1;

    if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
        if (u.err) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "%s in \"%V\" of the \"listen\" directive",
                               u.err, &u.url);
        }

        return NGX_CONF_ERROR;
    }

    cmcf = ngx_mail_conf_get_module_main_conf(cf, ngx_mail_core_module);

    ls = cmcf->listen.elts;

    for (i = 0; i < cmcf->listen.nelts; i++) {

        sa = (struct sockaddr *) ls[i].sockaddr;

        if (sa->sa_family != u.family) {
            continue;
        }

        switch (sa->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            off = offsetof(struct sockaddr_in6, sin6_addr);
            len = 16;
            sin6 = (struct sockaddr_in6 *) sa;
```

```
            port = sin6->sin6_port;
            break;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
        case AF_UNIX:
            off = offsetof(struct sockaddr_un, sun_path);
            len = sizeof(((struct sockaddr_un *) sa)->sun_path);
            port = 0;
            break;
#endif

        default: /* AF_INET */
            off = offsetof(struct sockaddr_in, sin_addr);
            len = 4;
            sin = (struct sockaddr_in *) sa;
            port = sin->sin_port;
            break;
        }

        if (ngx_memcmp(ls[i].sockaddr + off, u.sockaddr + off, len) != 0) {
            continue;
        }

        if (port != u.port) {
            continue;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "duplicate \"%V\" address and port pair", &u.url);
        return NGX_CONF_ERROR;
    }

    ls = ngx_array_push(&cmcf->listen);
    if (ls == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memzero(ls, sizeof(ngx_mail_listen_t));

    ngx_memcpy(ls->sockaddr, u.sockaddr, u.socklen);

    ls->socklen = u.socklen;
    ls->wildcard = u.wildcard;
    ls->ctx = cf->ctx;

#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    ls->ipv6only = 1;
#endif

    if (cscf->protocol == NULL) {
        for (m = 0; ngx_modules[m]; m++) {
            if (ngx_modules[m]->type != NGX_MAIL_MODULE) {
                continue;
            }

            module = ngx_modules[m]->ctx;

            if (module->protocol == NULL) {
                continue;
            }

            for (i = 0; module->protocol->port[i]; i++) {
                if (module->protocol->port[i] == u.port) {
                    cscf->protocol = module->protocol;
                    break;
                }
            }
        }
    }

    for (i = 2; i < cf->args->nelts; i++) {

        if (ngx_strcmp(value[i].data, "bind") == 0) {
            ls->bind = 1;
            continue;
```

```
        }

        if (ngx_strncmp(value[i].data, "ipv6only=o", 10) == 0) {
#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
            struct sockaddr  *sa;
            u_char            buf[NGX_SOCKADDR_STRLEN];

            sa = (struct sockaddr *) ls->sockaddr;

            if (sa->sa_family == AF_INET6) {

                if (ngx_strcmp(&value[i].data[10], "n") == 0) {
                    ls->ipv6only = 1;

                } else if (ngx_strcmp(&value[i].data[10], "ff") == 0) {
                    ls->ipv6only = 0;

                } else {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "invalid ipv6only flags \"%s\"",
                                       &value[i].data[9]);
                    return NGX_CONF_ERROR;
                }

                ls->bind = 1;

            } else {
                len = ngx_sock_ntop(sa, ls->socklen, buf,
                                    NGX_SOCKADDR_STRLEN, 1);

                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "ipv6only is not supported "
                                   "on addr \"%*s\", ignored", len, buf);
            }

            continue;
#else
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "bind ipv6only is not supported "
                               "on this platform");
            return NGX_CONF_ERROR;
#endif
        }

        if (ngx_strcmp(value[i].data, "ssl") == 0) {
#if (NGX_MAIL_SSL)
            ls->ssl = 1;
            continue;
#else
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "the \"ssl\" parameter requires "
                               "ngx_mail_ssl_module");
            return NGX_CONF_ERROR;
#endif
        }

        if (ngx_strncmp(value[i].data, "so_keepalive=", 13) == 0) {

            if (ngx_strcmp(&value[i].data[13], "on") == 0) {
                ls->so_keepalive = 1;

            } else if (ngx_strcmp(&value[i].data[13], "off") == 0) {
                ls->so_keepalive = 2;

            } else {

#if (NGX_HAVE_KEEPALIVE_TUNABLE)
                u_char     *p, *end;
                ngx_str_t   s;

                end = value[i].data + value[i].len;
                s.data = value[i].data + 13;

                p = ngx_strlchr(s.data, end, ':');
                if (p == NULL) {
                    p = end;
```

```
                }

                if (p > s.data) {
                    s.len = p - s.data;

                    ls->tcp_keepidle = ngx_parse_time(&s, 1);
                    if (ls->tcp_keepidle == (time_t) NGX_ERROR) {
                        goto invalid_so_keepalive;
                    }
                }

                s.data = (p < end) ? (p + 1) : end;

                p = ngx_strlchr(s.data, end, ':');
                if (p == NULL) {
                    p = end;
                }

                if (p > s.data) {
                    s.len = p - s.data;

                    ls->tcp_keepintvl = ngx_parse_time(&s, 1);
                    if (ls->tcp_keepintvl == (time_t) NGX_ERROR) {
                        goto invalid_so_keepalive;
                    }
                }

                s.data = (p < end) ? (p + 1) : end;

                if (s.data < end) {
                    s.len = end - s.data;

                    ls->tcp_keepcnt = ngx_atoi(s.data, s.len);
                    if (ls->tcp_keepcnt == NGX_ERROR) {
                        goto invalid_so_keepalive;
                    }
                }

                if (ls->tcp_keepidle == 0 && ls->tcp_keepintvl == 0
                    && ls->tcp_keepcnt == 0)
                {
                    goto invalid_so_keepalive;
                }

                ls->so_keepalive = 1;

#else

                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "the \"so_keepalive\" parameter accepts "
                                   "only \"on\" or \"off\" on this platform");
                return NGX_CONF_ERROR;

#endif
            }

            ls->bind = 1;

            continue;

#if (NGX_HAVE_KEEPALIVE_TUNABLE)
        invalid_so_keepalive:

            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid so_keepalive value: \"%s\"",
                               &value[i].data[13]);
            return NGX_CONF_ERROR;
#endif
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "the invalid \"%V\" parameter", &value[i]);
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
```

```
}


static char *
ngx_mail_core_protocol(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_core_srv_conf_t  *cscf = conf;

    ngx_str_t          *value;
    ngx_uint_t          m;
    ngx_mail_module_t  *module;

    value = cf->args->elts;

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_MAIL_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;

        if (module->protocol
            && ngx_strcmp(module->protocol->name.data, value[1].data) == 0)
        {
            cscf->protocol = module->protocol;

            return NGX_CONF_OK;
        }
    }

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "unknown protocol \"%V\"", &value[1]);
    return NGX_CONF_ERROR;
}


static char *
ngx_mail_core_resolver(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_core_srv_conf_t  *cscf = conf;

    ngx_str_t  *value;

    value = cf->args->elts;

    if (cscf->resolver != NGX_CONF_UNSET_PTR) {
        return "is duplicate";
    }

    if (ngx_strcmp(value[1].data, "off") == 0) {
        cscf->resolver = NULL;
        return NGX_CONF_OK;
    }

    cscf->resolver = ngx_resolver_create(cf, &value[1], cf->args->nelts - 1);
    if (cscf->resolver == NULL) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


char *
ngx_mail_capabilities(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    char  *p = conf;

    ngx_str_t    *c, *value;
    ngx_uint_t    i;
    ngx_array_t  *a;

    a = (ngx_array_t *) (p + cmd->offset);

    value = cf->args->elts;

```

```
643     for (i = 1; i < cf->args->nelts; i++) {
644         c = ngx_array_push(a);
645         if (c == NULL) {
646             return NGX_CONF_ERROR;
647         }
648 
649         *c = value[i];
650     }
651 
652     return NGX_CONF_OK;
653 }
```

One Level Up Top Level

src/mail/ngx_mail_auth_http_module.c - nginx-1.7.10

Global variables defined

- ngx_mail_auth_http_commands
- ngx_mail_auth_http_method
- ngx_mail_auth_http_module
- ngx_mail_auth_http_module_ctx
- ngx_mail_smtp_errcode

Data types defined

- ngx_mail_auth_http_conf_t
- ngx_mail_auth_http_ctx_s
- ngx_mail_auth_http_ctx_t
- ngx_mail_auth_http_handler_pt

Functions defined

- ngx_mail_auth_http
- ngx_mail_auth_http_block_read
- ngx_mail_auth_http_create_conf
- ngx_mail_auth_http_create_request
- ngx_mail_auth_http_dummy_handler
- ngx_mail_auth_http_escape
- ngx_mail_auth_http_header
- ngx_mail_auth_http_ignore_status_line
- ngx_mail_auth_http_init
- ngx_mail_auth_http_merge_conf
- ngx_mail_auth_http_parse_header_line
- ngx_mail_auth_http_process_headers
- ngx_mail_auth_http_read_handler
- ngx_mail_auth_http_write_handler
- ngx_mail_auth_sleep_handler

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
```

```
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_event.h>
 #include <ngx_event_connect.h>
 #include <ngx_mail.h>


 typedef struct {
     ngx_addr_t                     *peer;

     ngx_msec_t                      timeout;

     ngx_str_t                       host_header;
     ngx_str_t                       uri;
     ngx_str_t                       header;

     ngx_array_t                    *headers;

     u_char                         *file;
     ngx_uint_t                      line;
 } ngx_mail_auth_http_conf_t;


 typedef struct ngx_mail_auth_http_ctx_s  ngx_mail_auth_http_ctx_t;

 typedef void (*ngx_mail_auth_http_handler_pt)(ngx_mail_session_t *s,
     ngx_mail_auth_http_ctx_t *ctx);

 struct ngx_mail_auth_http_ctx_s {
     ngx_buf_t                      *request;
     ngx_buf_t                      *response;
     ngx_peer_connection_t           peer;

     ngx_mail_auth_http_handler_pt   handler;

     ngx_uint_t                      state;

     u_char                         *header_name_start;
     u_char                         *header_name_end;
     u_char                         *header_start;
     u_char                         *header_end;

     ngx_str_t                       addr;
     ngx_str_t                       port;
     ngx_str_t                       err;
     ngx_str_t                       errmsg;
     ngx_str_t                       errcode;

     time_t                          sleep;

     ngx_pool_t                     *pool;
 };


 static void ngx_mail_auth_http_write_handler(ngx_event_t *wev);
 static void ngx_mail_auth_http_read_handler(ngx_event_t *rev);
 static void ngx_mail_auth_http_ignore_status_line(ngx_mail_session_t *s,
     ngx_mail_auth_http_ctx_t *ctx);
 static void ngx_mail_auth_http_process_headers(ngx_mail_session_t *s,
     ngx_mail_auth_http_ctx_t *ctx);
 static void ngx_mail_auth_sleep_handler(ngx_event_t *rev);
 static ngx_int_t ngx_mail_auth_http_parse_header_line(ngx_mail_session_t *s,
     ngx_mail_auth_http_ctx_t *ctx);
 static void ngx_mail_auth_http_block_read(ngx_event_t *rev);
 static void ngx_mail_auth_http_dummy_handler(ngx_event_t *ev);
 static ngx_buf_t *ngx_mail_auth_http_create_request(ngx_mail_session_t *s,
     ngx_pool_t *pool, ngx_mail_auth_http_conf_t *ahcf);
 static ngx_int_t ngx_mail_auth_http_escape(ngx_pool_t *pool, ngx_str_t *text,
     ngx_str_t *escaped);

 static void *ngx_mail_auth_http_create_conf(ngx_conf_t *cf);
 static char *ngx_mail_auth_http_merge_conf(ngx_conf_t *cf, void *parent,
```

```
    void *child);
static char *ngx_mail_auth_http(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static char *ngx_mail_auth_http_header(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


static ngx_command_t  ngx_mail_auth_http_commands[] = {

    { ngx_string("auth_http"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_mail_auth_http,
      NGX_MAIL_SRV_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("auth_http_timeout"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_auth_http_conf_t, timeout),
       NULL },

     { ngx_string("auth_http_header"),
       NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE2,
       ngx_mail_auth_http_header,
       NGX_MAIL_SRV_CONF_OFFSET,
       0,
       NULL },

       ngx_null_command
};


static ngx_mail_module_t  ngx_mail_auth_http_module_ctx = {
    NULL,                                  /* protocol */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_mail_auth_http_create_conf,        /* create server configuration */
    ngx_mail_auth_http_merge_conf          /* merge server configuration */
};


ngx_module_t  ngx_mail_auth_http_module = {
    NGX_MODULE_V1,
    &ngx_mail_auth_http_module_ctx,        /* module context */
    ngx_mail_auth_http_commands,           /* module directives */
    NGX_MAIL_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t   ngx_mail_auth_http_method[] = {
    ngx_string("plain"),
    ngx_string("plain"),
    ngx_string("plain"),
    ngx_string("apop"),
    ngx_string("cram-md5"),
    ngx_string("none")
};

static ngx_str_t   ngx_mail_smtp_errcode = ngx_string("535 5.7.0");


void
ngx_mail_auth_http_init(ngx_mail_session_t *s)
{
    ngx_int_t                  rc;
```

```
    ngx_pool_t                 *pool;
    ngx_mail_auth_http_ctx_t   *ctx;
    ngx_mail_auth_http_conf_t  *ahcf;

    s->connection->log->action = "in http auth state";

    pool = ngx_create_pool(2048, s->connection->log);
    if (pool == NULL) {
        ngx_mail_session_internal_server_error(s);
        return;
    }

    ctx = ngx_pcalloc(pool, sizeof(ngx_mail_auth_http_ctx_t));
    if (ctx == NULL) {
        ngx_destroy_pool(pool);
        ngx_mail_session_internal_server_error(s);
        return;
    }

    ctx->pool = pool;

    ahcf = ngx_mail_get_module_srv_conf(s, ngx_mail_auth_http_module);

    ctx->request = ngx_mail_auth_http_create_request(s, pool, ahcf);
    if (ctx->request == NULL) {
        ngx_destroy_pool(ctx->pool);
        ngx_mail_session_internal_server_error(s);
        return;
    }

    ngx_mail_set_ctx(s, ctx, ngx_mail_auth_http_module);

    ctx->peer.sockaddr = ahcf->peer->sockaddr;
    ctx->peer.socklen = ahcf->peer->socklen;
    ctx->peer.name = &ahcf->peer->name;
    ctx->peer.get = ngx_event_get_peer;
    ctx->peer.log = s->connection->log;
    ctx->peer.log_error = NGX_ERROR_ERR;

    rc = ngx_event_connect_peer(&ctx->peer);

    if (rc == NGX_ERROR || rc == NGX_BUSY || rc == NGX_DECLINED) {
        if (ctx->peer.connection) {
            ngx_close_connection(ctx->peer.connection);
        }

        ngx_destroy_pool(ctx->pool);
        ngx_mail_session_internal_server_error(s);
        return;
    }

    ctx->peer.connection->data = s;
    ctx->peer.connection->pool = s->connection->pool;

    s->connection->read->handler = ngx_mail_auth_http_block_read;
    ctx->peer.connection->read->handler = ngx_mail_auth_http_read_handler;
    ctx->peer.connection->write->handler = ngx_mail_auth_http_write_handler;

    ctx->handler = ngx_mail_auth_http_ignore_status_line;

    ngx_add_timer(ctx->peer.connection->read, ahcf->timeout);
    ngx_add_timer(ctx->peer.connection->write, ahcf->timeout);

    if (rc == NGX_OK) {
        ngx_mail_auth_http_write_handler(ctx->peer.connection->write);
        return;
    }
}


static void
ngx_mail_auth_http_write_handler(ngx_event_t *wev)
{
    ssize_t                    n, size;
    ngx_connection_t           *c;
    ngx_mail_session_t         *s;
```

```
    ngx_mail_auth_http_ctx_t   *ctx;
    ngx_mail_auth_http_conf_t  *ahcf;

    c = wev->data;
    s = c->data;

    ctx = ngx_mail_get_module_ctx(s, ngx_mail_auth_http_module);

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, wev->log, 0,
                   "mail auth http write handler");

    if (wev->timedout) {
        ngx_log_error(NGX_LOG_ERR, wev->log, NGX_ETIMEDOUT,
                      "auth http server %V timed out", ctx->peer.name);
        ngx_close_connection(c);
        ngx_destroy_pool(ctx->pool);
        ngx_mail_session_internal_server_error(s);
        return;
    }

    size = ctx->request->last - ctx->request->pos;

    n = ngx_send(c, ctx->request->pos, size);

    if (n == NGX_ERROR) {
        ngx_close_connection(c);
        ngx_destroy_pool(ctx->pool);
        ngx_mail_session_internal_server_error(s);
        return;
    }

    if (n > 0) {
        ctx->request->pos += n;

        if (n == size) {
            wev->handler = ngx_mail_auth_http_dummy_handler;

            if (wev->timer_set) {
                ngx_del_timer(wev);
            }

            if (ngx_handle_write_event(wev, 0) != NGX_OK) {
                ngx_close_connection(c);
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
            }

            return;
        }
    }

    if (!wev->timer_set) {
        ahcf = ngx_mail_get_module_srv_conf(s, ngx_mail_auth_http_module);
        ngx_add_timer(wev, ahcf->timeout);
    }
}


static void
ngx_mail_auth_http_read_handler(ngx_event_t *rev)
{
    ssize_t                     n, size;
    ngx_connection_t           *c;
    ngx_mail_session_t         *s;
    ngx_mail_auth_http_ctx_t   *ctx;

    c = rev->data;
    s = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                   "mail auth http read handler");

    ctx = ngx_mail_get_module_ctx(s, ngx_mail_auth_http_module);

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_ERR, rev->log, NGX_ETIMEDOUT,
```

```
                      "auth http server %V timed out", ctx->peer.name);
        ngx_close_connection(c);
        ngx_destroy_pool(ctx->pool);
        ngx_mail_session_internal_server_error(s);
        return;
    }

    if (ctx->response == NULL) {
        ctx->response = ngx_create_temp_buf(ctx->pool, 1024);
        if (ctx->response == NULL) {
            ngx_close_connection(c);
            ngx_destroy_pool(ctx->pool);
            ngx_mail_session_internal_server_error(s);
            return;
        }
    }

    size = ctx->response->end - ctx->response->last;

    n = ngx_recv(c, ctx->response->pos, size);

    if (n > 0) {
        ctx->response->last += n;

        ctx->handler(s, ctx);
        return;
    }

    if (n == NGX_AGAIN) {
        return;
    }

    ngx_close_connection(c);
    ngx_destroy_pool(ctx->pool);
    ngx_mail_session_internal_server_error(s);
}


static void
ngx_mail_auth_http_ignore_status_line(ngx_mail_session_t *s,
    ngx_mail_auth_http_ctx_t *ctx)
{
    u_char  *p, ch;
    enum  {
        sw_start = 0,
        sw_H,
        sw_HT,
        sw_HTT,
        sw_HTTP,
        sw_skip,
        sw_almost_done
    } state;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
                   "mail auth http process status line");

    state = ctx->state;

    for (p = ctx->response->pos; p < ctx->response->last; p++) {
        ch = *p;

        switch (state) {

        /* "HTTP/" */
        case sw_start:
            if (ch == 'H') {
                state = sw_H;
                break;
            }
            goto next;

        case sw_H:
            if (ch == 'T') {
                state = sw_HT;
                break;
            }
```

```
            goto next;

        case sw_HT:
            if (ch == 'T') {
                state = sw_HTT;
                break;
            }
            goto next;

        case sw_HTT:
            if (ch == 'P') {
                state = sw_HTTP;
                break;
            }
            goto next;

        case sw_HTTP:
            if (ch == '/') {
                state = sw_skip;
                break;
            }
            goto next;

        /* any text until end of line */
        case sw_skip:
            switch (ch) {
            case CR:
                state = sw_almost_done;

                break;
            case LF:
                goto done;
            }
            break;

        /* end of status line */
        case sw_almost_done:
            if (ch == LF) {
                goto done;
            }

            ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                          "auth http server &V sent invalid response",
                          ctx->peer.name);
            ngx_close_connection(ctx->peer.connection);
            ngx_destroy_pool(ctx->pool);
            ngx_mail_session_internal_server_error(s);
            return;
        }
    }

    ctx->response->pos = p;
    ctx->state = state;

    return;

next:

    p = ctx->response->start - 1;

done:

    ctx->response->pos = p + 1;
    ctx->state = 0;
    ctx->handler = ngx_mail_auth_http_process_headers;
    ctx->handler(s, ctx);
}


static void
ngx_mail_auth_http_process_headers(ngx_mail_session_t *s,
    ngx_mail_auth_http_ctx_t *ctx)
{
    u_char               *p;
    time_t                timer;
    size_t                len, size;
```

```
    ngx_int_t             rc, port, n;
    ngx_addr_t           *peer;
    struct sockaddr_in   *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6  *sin6;
#endif

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
                   "mail auth http process headers");

    for ( ;; ) {
        rc = ngx_mail_auth_http_parse_header_line(s, ctx);

        if (rc == NGX_OK) {

#if (NGX_DEBUG)
            {
            ngx_str_t  key, value;

            key.len = ctx->header_name_end - ctx->header_name_start;
            key.data = ctx->header_name_start;
            value.len = ctx->header_end - ctx->header_start;
            value.data = ctx->header_start;

            ngx_log_debug2(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
                           "mail auth http header: \"%V: %V\"",
                           &key, &value);
            }
#endif

            len = ctx->header_name_end - ctx->header_name_start;

            if (len == sizeof("Auth-Status") - 1
                && ngx_strncasecmp(ctx->header_name_start,
                                   (u_char *) "Auth-Status",
                                   sizeof("Auth-Status") - 1)
                   == 0)
            {
                len = ctx->header_end - ctx->header_start;

                if (len == 2
                    && ctx->header_start[0] == 'O'
                    && ctx->header_start[1] == 'K')
                {
                    continue;
                }

                if (len == 4
                    && ctx->header_start[0] == 'W'
                    && ctx->header_start[1] == 'A'
                    && ctx->header_start[2] == 'I'
                    && ctx->header_start[3] == 'T')
                {
                    s->auth_wait = 1;
                    continue;
                }

                ctx->errmsg.len = len;
                ctx->errmsg.data = ctx->header_start;

                switch (s->protocol) {

                case NGX_MAIL_POP3_PROTOCOL:
                    size = sizeof("-ERR ") - 1 + len + sizeof(CRLF) - 1;
                    break;

                case NGX_MAIL_IMAP_PROTOCOL:
                    size = s->tag.len + sizeof("NO ") - 1 + len
                           + sizeof(CRLF) - 1;
                    break;

                default: /* NGX_MAIL_SMTP_PROTOCOL */
                    ctx->err = ctx->errmsg;
                    continue;
                }

```

```
            p = ngx_pnalloc(s->connection->pool, size);
            if (p == NULL) {
                ngx_close_connection(ctx->peer.connection);
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            ctx->err.data = p;

            switch (s->protocol) {

            case NGX_MAIL_POP3_PROTOCOL:
                *p++ = '-'; *p++ = 'E'; *p++ = 'R'; *p++ = 'R'; *p++ = ' ';
                break;

            case NGX_MAIL_IMAP_PROTOCOL:
                p = ngx_cpymem(p, s->tag.data, s->tag.len);
                *p++ = 'N'; *p++ = 'O'; *p++ = ' ';
                break;

            default: /* NGX_MAIL_SMTP_PROTOCOL */
                break;
            }

            p = ngx_cpymem(p, ctx->header_start, len);
            *p++ = CR; *p++ = LF;

            ctx->err.len = p - ctx->err.data;

            continue;
        }

        if (len == sizeof("Auth-Server") - 1
            && ngx_strncasecmp(ctx->header_name_start,
                               (u_char *) "Auth-Server",
                               sizeof("Auth-Server") - 1)
                == 0)
        {
            ctx->addr.len = ctx->header_end - ctx->header_start;
            ctx->addr.data = ctx->header_start;

            continue;
        }

        if (len == sizeof("Auth-Port") - 1
            && ngx_strncasecmp(ctx->header_name_start,
                               (u_char *) "Auth-Port",
                               sizeof("Auth-Port") - 1)
               == 0)
        {
            ctx->port.len = ctx->header_end - ctx->header_start;
            ctx->port.data = ctx->header_start;

            continue;
        }

        if (len == sizeof("Auth-User") - 1
            && ngx_strncasecmp(ctx->header_name_start,
                               (u_char *) "Auth-User",
                               sizeof("Auth-User") - 1)
               == 0)
        {
            s->login.len = ctx->header_end - ctx->header_start;

            s->login.data = ngx_pnalloc(s->connection->pool, s->login.len);
            if (s->login.data == NULL) {
                ngx_close_connection(ctx->peer.connection);
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            ngx_memcpy(s->login.data, ctx->header_start, s->login.len);

            continue;
```

```
612             }
613
614             if (len == sizeof("Auth-Pass") - 1
615                 && ngx_strncasecmp(ctx->header_name_start,
616                                    (u_char *) "Auth-Pass",
617                                    sizeof("Auth-Pass") - 1)
618                    == 0)
619             {
620                 s->passwd.len = ctx->header_end - ctx->header_start;
621
622                 s->passwd.data = ngx_pnalloc(s->connection->pool,
623                                              s->passwd.len);
624                 if (s->passwd.data == NULL) {
625                     ngx_close_connection(ctx->peer.connection);
626                     ngx_destroy_pool(ctx->pool);
627                     ngx_mail_session_internal_server_error(s);
628                     return;
629                 }
630
631                 ngx_memcpy(s->passwd.data, ctx->header_start, s->passwd.len);
632
633                 continue;
634             }
635
636             if (len == sizeof("Auth-Wait") - 1
637                 && ngx_strncasecmp(ctx->header_name_start,
638                                    (u_char *) "Auth-Wait",
639                                    sizeof("Auth-Wait") - 1)
640                    == 0)
641             {
642                 n = ngx_atoi(ctx->header_start,
643                              ctx->header_end - ctx->header_start);
644
645                 if (n != NGX_ERROR) {
646                     ctx->sleep = n;
647                 }
648
649                 continue;
650             }
651
652             if (len == sizeof("Auth-Error-Code") - 1
653                 && ngx_strncasecmp(ctx->header_name_start,
654                                    (u_char *) "Auth-Error-Code",
655                                    sizeof("Auth-Error-Code") - 1)
656                    == 0)
657             {
658                 ctx->errcode.len = ctx->header_end - ctx->header_start;
659
660                 ctx->errcode.data = ngx_pnalloc(s->connection->pool,
661                                                 ctx->errcode.len);
662                 if (ctx->errcode.data == NULL) {
663                     ngx_close_connection(ctx->peer.connection);
664                     ngx_destroy_pool(ctx->pool);
665                     ngx_mail_session_internal_server_error(s);
666                     return;
667                 }
668
669                 ngx_memcpy(ctx->errcode.data, ctx->header_start,
670                            ctx->errcode.len);
671
672                 continue;
673             }
674
675             /* ignore other headers */
676
677             continue;
678         }
679
680         if (rc == NGX_DONE) {
681             ngx_log_debug0(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
682                            "mail auth http header done");
683
684             ngx_close_connection(ctx->peer.connection);
685
686             if (ctx->err.len) {
687
```

```
            ngx_log_error(NGX_LOG_INFO, s->connection->log, 0,
                          "client login failed: \"%V\"", &ctx->errmsg);

            if (s->protocol == NGX_MAIL_SMTP_PROTOCOL) {

                if (ctx->errcode.len == 0) {
                    ctx->errcode = ngx_mail_smtp_errcode;
                }

                ctx->err.len = ctx->errcode.len + ctx->errmsg.len
                               + sizeof(" " CRLF) - 1;

                p = ngx_pnalloc(s->connection->pool, ctx->err.len);
                if (p == NULL) {
                    ngx_destroy_pool(ctx->pool);
                    ngx_mail_session_internal_server_error(s);
                    return;
                }

                ctx->err.data = p;

                p = ngx_cpymem(p, ctx->errcode.data, ctx->errcode.len);
                *p++ = ' ';
                p = ngx_cpymem(p, ctx->errmsg.data, ctx->errmsg.len);
                *p++ = CR; *p = LF;
            }

            s->out = ctx->err;
            timer = ctx->sleep;

            ngx_destroy_pool(ctx->pool);

            if (timer == 0) {
                s->quit = 1;
                ngx_mail_send(s->connection->write);
                return;
            }

            ngx_add_timer(s->connection->read, (ngx_msec_t) (timer * 1000));

            s->connection->read->handler = ngx_mail_auth_sleep_handler;

            return;
        }

        if (s->auth_wait) {
            timer = ctx->sleep;

            ngx_destroy_pool(ctx->pool);

            if (timer == 0) {
                ngx_mail_auth_http_init(s);
                return;
            }

            ngx_add_timer(s->connection->read, (ngx_msec_t) (timer * 1000));

            s->connection->read->handler = ngx_mail_auth_sleep_handler;

            return;
        }

        if (ctx->addr.len == 0 || ctx->port.len == 0) {
            ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                          "auth http server %V did not send server or port",
                          ctx->peer.name);
            ngx_destroy_pool(ctx->pool);
            ngx_mail_session_internal_server_error(s);
            return;
        }

        if (s->passwd.data == NULL
            && s->protocol != NGX_MAIL_SMTP_PROTOCOL)
        {
            ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                          "auth http server %V did not send password",
```

```
                              ctx->peer.name);
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            peer = ngx_pcalloc(s->connection->pool, sizeof(ngx_addr_t));
            if (peer == NULL) {
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            rc = ngx_parse_addr(s->connection->pool, peer,
                                ctx->addr.data, ctx->addr.len);

            switch (rc) {
            case NGX_OK:
                break;

            case NGX_DECLINED:
                ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                              "auth http server %V sent invalid server "
                              "address:\"%V\"",
                              ctx->peer.name, &ctx->addr);
                /* fall through */

            default:
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            port = ngx_atoi(ctx->port.data, ctx->port.len);
            if (port == NGX_ERROR || port < 1 || port > 65535) {
                ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                              "auth http server %V sent invalid server "
                              "port:\"%V\"",
                              ctx->peer.name, &ctx->port);
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            switch (peer->sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
            case AF_INET6:
                sin6 = (struct sockaddr_in6 *) peer->sockaddr;
                sin6->sin6_port = htons((in_port_t) port);
                break;
#endif

            default: /* AF_INET */
                sin = (struct sockaddr_in *) peer->sockaddr;
                sin->sin_port = htons((in_port_t) port);
                break;
            }

            len = ctx->addr.len + 1 + ctx->port.len;

            peer->name.len = len;

            peer->name.data = ngx_pnalloc(s->connection->pool, len);
            if (peer->name.data == NULL) {
                ngx_destroy_pool(ctx->pool);
                ngx_mail_session_internal_server_error(s);
                return;
            }

            len = ctx->addr.len;

            ngx_memcpy(peer->name.data, ctx->addr.data, len);

            peer->name.data[len++] = ':';

```

```
            ngx_memcpy(peer->name.data + len, ctx->port.data, ctx->port.len);

            ngx_destroy_pool(ctx->pool);
            ngx_mail_proxy_init(s, peer);

            return;
        }

        if (rc == NGX_AGAIN ) {
            return;
        }

        /* rc == NGX_ERROR */

        ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                      "auth http server %V sent invalid header in response",
                      ctx->peer.name);
        ngx_close_connection(ctx->peer.connection);
        ngx_destroy_pool(ctx->pool);
        ngx_mail_session_internal_server_error(s);

        return;
    }
}


static void
ngx_mail_auth_sleep_handler(ngx_event_t *rev)
{
    ngx_connection_t          *c;
    ngx_mail_session_t        *s;
    ngx_mail_core_srv_conf_t  *cscf;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0, "mail auth sleep handler");

    c = rev->data;
    s = c->data;

    if (rev->timedout) {

        rev->timedout = 0;

        if (s->auth_wait) {
            s->auth_wait = 0;
            ngx_mail_auth_http_init(s);
            return;
        }

        cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

        rev->handler = cscf->protocol->auth_state;

        s->mail_state = 0;
        s->auth_method = NGX_MAIL_AUTH_PLAIN;

        c->log->action = "in auth state";

        ngx_mail_send(c->write);

        if (c->destroyed) {
            return;
        }

        ngx_add_timer(rev, cscf->timeout);

        if (rev->ready) {
            rev->handler(rev);
            return;
        }

        if (ngx_handle_read_event(rev, 0) != NGX_OK) {
            ngx_mail_close_connection(c);
        }

        return;
    }
```

```
916
917     if (rev->active) {
918         if (ngx_handle_read_event(rev, 0) != NGX_OK) {
919             ngx_mail_close_connection(c);
920         }
921     }
922 }
923
924
925 static ngx_int_t
926 ngx_mail_auth_http_parse_header_line(ngx_mail_session_t *s,
927     ngx_mail_auth_http_ctx_t *ctx)
928 {
929     u_char      c, ch, *p;
930     enum {
931         sw_start = 0,
932         sw_name,
933         sw_space_before_value,
934         sw_value,
935         sw_space_after_value,
936         sw_almost_done,
937         sw_header_almost_done
938     } state;
939
940     state = ctx->state;
941
942     for (p = ctx->response->pos; p < ctx->response->last; p++) {
943         ch = *p;
944
945         switch (state) {
946
947         /* first char */
948         case sw_start:
949
950             switch (ch) {
951             case CR:
952                 ctx->header_end = p;
953                 state = sw_header_almost_done;
954                 break;
955             case LF:
956                 ctx->header_end = p;
957                 goto header_done;
958             default:
959                 state = sw_name;
960                 ctx->header_name_start = p;
961
962                 c = (u_char) (ch | 0x20);
963                 if (c >= 'a' && c <= 'z') {
964                     break;
965                 }
966
967                 if (ch >= '0' && ch <= '9') {
968                     break;
969                 }
970
971                 return NGX_ERROR;
972             }
973             break;
974
975         /* header name */
976         case sw_name:
977             c = (u_char) (ch | 0x20);
978             if (c >= 'a' && c <= 'z') {
979                 break;
980             }
981
982             if (ch == ':') {
983                 ctx->header_name_end = p;
984                 state = sw_space_before_value;
985                 break;
986             }
987
988             if (ch == '-') {
989                 break;
990             }
991
```

```
992             if (ch >= '0' && ch <= '9') {
993                 break;
994             }
995 
996             if (ch == CR) {
997                 ctx->header_name_end = p;
998                 ctx->header_start = p;
999                 ctx->header_end = p;
1000                 state = sw_almost_done;
1001                 break;
1002             }
1003 
1004             if (ch == LF) {
1005                 ctx->header_name_end = p;
1006                 ctx->header_start = p;
1007                 ctx->header_end = p;
1008                 goto done;
1009             }
1010 
1011             return NGX_ERROR;
1012 
1013         /* space* before header value */
1014         case sw_space_before_value:
1015             switch (ch) {
1016             case ' ':
1017                 break;
1018             case CR:
1019                 ctx->header_start = p;
1020                 ctx->header_end = p;
1021                 state = sw_almost_done;
1022                 break;
1023             case LF:
1024                 ctx->header_start = p;
1025                 ctx->header_end = p;
1026                 goto done;
1027             default:
1028                 ctx->header_start = p;
1029                 state = sw_value;
1030                 break;
1031             }
1032             break;
1033 
1034         /* header value */
1035         case sw_value:
1036             switch (ch) {
1037             case ' ':
1038                 ctx->header_end = p;
1039                 state = sw_space_after_value;
1040                 break;
1041             case CR:
1042                 ctx->header_end = p;
1043                 state = sw_almost_done;
1044                 break;
1045             case LF:
1046                 ctx->header_end = p;
1047                 goto done;
1048             }
1049             break;
1050 
1051         /* space* before end of header line */
1052         case sw_space_after_value:
1053             switch (ch) {
1054             case ' ':
1055                 break;
1056             case CR:
1057                 state = sw_almost_done;
1058                 break;
1059             case LF:
1060                 goto done;
1061             default:
1062                 state = sw_value;
1063                 break;
1064             }
1065             break;
1066 
1067         /* end of header line */
```

```
1068            case sw_almost_done:
1069                switch (ch) {
1070                case LF:
1071                    goto done;
1072                default:
1073                    return NGX_ERROR;
1074                }
1075
1076            /* end of header */
1077            case sw_header_almost_done:
1078                switch (ch) {
1079                case LF:
1080                    goto header_done;
1081                default:
1082                    return NGX_ERROR;
1083                }
1084            }
1085        }
1086
1087        ctx->response->pos = p;
1088        ctx->state = state;
1089
1090        return NGX_AGAIN;
1091
1092    done:
1093
1094        ctx->response->pos = p + 1;
1095        ctx->state = sw_start;
1096
1097        return NGX_OK;
1098
1099    header_done:
1100
1101        ctx->response->pos = p + 1;
1102        ctx->state = sw_start;
1103
1104        return NGX_DONE;
1105    }
1106
1107
1108    static void
1109    ngx_mail_auth_http_block_read(ngx_event_t *rev)
1110    {
1111        ngx_connection_t          *c;
1112        ngx_mail_session_t        *s;
1113        ngx_mail_auth_http_ctx_t  *ctx;
1114
1115        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
1116                       "mail auth http block read");
1117
1118        if (ngx_handle_read_event(rev, 0) != NGX_OK) {
1119            c = rev->data;
1120            s = c->data;
1121
1122            ctx = ngx_mail_get_module_ctx(s, ngx_mail_auth_http_module);
1123
1124            ngx_close_connection(ctx->peer.connection);
1125            ngx_destroy_pool(ctx->pool);
1126            ngx_mail_session_internal_server_error(s);
1127        }
1128    }
1129
1130
1131    static void
1132    ngx_mail_auth_http_dummy_handler(ngx_event_t *ev)
1133    {
1134        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, ev->log, 0,
1135                       "mail auth http dummy handler");
1136    }
1137
1138
1139    static ngx_buf_t *
1140    ngx_mail_auth_http_create_request(ngx_mail_session_t *s, ngx_pool_t *pool,
1141        ngx_mail_auth_http_conf_t *ahcf)
1142    {
1143        size_t                     len;
```

```
    ngx_buf_t                  *b;
    ngx_str_t                   login, passwd;
    ngx_mail_core_srv_conf_t   *cscf;

    if (ngx_mail_auth_http_escape(pool, &s->login, &login) != NGX_OK) {
        return NULL;
    }

    if (ngx_mail_auth_http_escape(pool, &s->passwd, &passwd) != NGX_OK) {
        return NULL;
    }

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

    len = sizeof("GET ") - 1 + ahcf->uri.len + sizeof(" HTTP/1.0" CRLF) - 1
          + sizeof("Host: ") - 1 + ahcf->host_header.len + sizeof(CRLF) - 1
          + sizeof("Auth-Method: ") - 1
                + ngx_mail_auth_http_method[s->auth_method].len
                + sizeof(CRLF) - 1
          + sizeof("Auth-User: ") - 1 + login.len + sizeof(CRLF) - 1
          + sizeof("Auth-Pass: ") - 1 + passwd.len + sizeof(CRLF) - 1
          + sizeof("Auth-Salt: ") - 1 + s->salt.len
          + sizeof("Auth-Protocol: ") - 1 + cscf->protocol->name.len
                + sizeof(CRLF) - 1
          + sizeof("Auth-Login-Attempt: ") - 1 + NGX_INT_T_LEN
                + sizeof(CRLF) - 1
          + sizeof("Client-IP: ") - 1 + s->connection->addr_text.len
                + sizeof(CRLF) - 1
          + sizeof("Client-Host: ") - 1 + s->host.len + sizeof(CRLF) - 1
          + sizeof("Auth-SMTP-Helo: ") - 1 + s->smtp_helo.len
          + sizeof("Auth-SMTP-From: ") - 1 + s->smtp_from.len
          + sizeof("Auth-SMTP-To: ") - 1 + s->smtp_to.len
          + ahcf->header.len
          + sizeof(CRLF) - 1;

    b = ngx_create_temp_buf(pool, len);
    if (b == NULL) {
        return NULL;
    }

    b->last = ngx_cpymem(b->last, "GET ", sizeof("GET ") - 1);
    b->last = ngx_copy(b->last, ahcf->uri.data, ahcf->uri.len);
    b->last = ngx_cpymem(b->last, " HTTP/1.0" CRLF,
                         sizeof(" HTTP/1.0" CRLF) - 1);

    b->last = ngx_cpymem(b->last, "Host: ", sizeof("Host: ") - 1);
    b->last = ngx_copy(b->last, ahcf->host_header.data,
                         ahcf->host_header.len);
    *b->last++ = CR; *b->last++ = LF;

    b->last = ngx_cpymem(b->last, "Auth-Method: ",
                         sizeof("Auth-Method: ") - 1);
    b->last = ngx_cpymem(b->last,
                         ngx_mail_auth_http_method[s->auth_method].data,
                         ngx_mail_auth_http_method[s->auth_method].len);
    *b->last++ = CR; *b->last++ = LF;

    b->last = ngx_cpymem(b->last, "Auth-User: ", sizeof("Auth-User: ") - 1);
    b->last = ngx_copy(b->last, login.data, login.len);
    *b->last++ = CR; *b->last++ = LF;

    b->last = ngx_cpymem(b->last, "Auth-Pass: ", sizeof("Auth-Pass: ") - 1);
    b->last = ngx_copy(b->last, passwd.data, passwd.len);
    *b->last++ = CR; *b->last++ = LF;

    if (s->auth_method != NGX_MAIL_AUTH_PLAIN && s->salt.len) {
        b->last = ngx_cpymem(b->last, "Auth-Salt: ", sizeof("Auth-Salt: ") - 1);
        b->last = ngx_copy(b->last, s->salt.data, s->salt.len);

        s->passwd.data = NULL;
    }

    b->last = ngx_cpymem(b->last, "Auth-Protocol: ",
                         sizeof("Auth-Protocol: ") - 1);
    b->last = ngx_cpymem(b->last, cscf->protocol->name.data,
                         cscf->protocol->name.len);
```

```
    *b->last++ = CR; *b->last++ = LF;

    b->last = ngx_sprintf(b->last, "Auth-Login-Attempt: %ui" CRLF,
                          s->login_attempt);

    b->last = ngx_cpymem(b->last, "Client-IP: ", sizeof("Client-IP: ") - 1);
    b->last = ngx_copy(b->last, s->connection->addr_text.data,
                       s->connection->addr_text.len);
    *b->last++ = CR; *b->last++ = LF;

    if (s->host.len) {
        b->last = ngx_cpymem(b->last, "Client-Host: ",
                             sizeof("Client-Host: ") - 1);
        b->last = ngx_copy(b->last, s->host.data, s->host.len);
        *b->last++ = CR; *b->last++ = LF;
    }

    if (s->auth_method == NGX_MAIL_AUTH_NONE) {

        /* HELO, MAIL FROM, and RCPT TO can't contain CRLF, no need to escape */

        b->last = ngx_cpymem(b->last, "Auth-SMTP-Helo: ",
                             sizeof("Auth-SMTP-Helo: ") - 1);
        b->last = ngx_copy(b->last, s->smtp_helo.data, s->smtp_helo.len);
        *b->last++ = CR; *b->last++ = LF;

        b->last = ngx_cpymem(b->last, "Auth-SMTP-From: ",
                             sizeof("Auth-SMTP-From: ") - 1);
        b->last = ngx_copy(b->last, s->smtp_from.data, s->smtp_from.len);
        *b->last++ = CR; *b->last++ = LF;

        b->last = ngx_cpymem(b->last, "Auth-SMTP-To: ",
                             sizeof("Auth-SMTP-To: ") - 1);
        b->last = ngx_copy(b->last, s->smtp_to.data, s->smtp_to.len);
        *b->last++ = CR; *b->last++ = LF;

    }

    if (ahcf->header.len) {
        b->last = ngx_copy(b->last, ahcf->header.data, ahcf->header.len);
    }

    /* add "\r\n" at the header end */
    *b->last++ = CR; *b->last++ = LF;

#if (NGX_DEBUG_MAIL_PASSWD)
    {
    ngx_str_t  l;

    l.len = b->last - b->pos;
    l.data = b->pos;
    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
                   "mail auth http header:%N\"%V\"", &l);
    }
#endif

    return b;
}


static ngx_int_t
ngx_mail_auth_http_escape(ngx_pool_t *pool, ngx_str_t *text, ngx_str_t *escaped)
{
    u_char     *p;
    uintptr_t   n;

    n = ngx_escape_uri(NULL, text->data, text->len, NGX_ESCAPE_MAIL_AUTH);

    if (n == 0) {
        *escaped = *text;
        return NGX_OK;
    }

    escaped->len = text->len + n * 2;

    p = ngx_pnalloc(pool, escaped->len);
```

```
    if (p == NULL) {
        return NGX_ERROR;
    }

    (void) ngx_escape_uri(p, text->data, text->len, NGX_ESCAPE_MAIL_AUTH);

    escaped->data = p;

    return NGX_OK;
}


static void *
ngx_mail_auth_http_create_conf(ngx_conf_t *cf)
{
    ngx_mail_auth_http_conf_t  *ahcf;

    ahcf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_auth_http_conf_t));
    if (ahcf == NULL) {
        return NULL;
    }

    ahcf->timeout = NGX_CONF_UNSET_MSEC;

    ahcf->file = cf->conf_file->file.name.data;
    ahcf->line = cf->conf_file->line;

    return ahcf;
}


static char *
ngx_mail_auth_http_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_mail_auth_http_conf_t *prev = parent;
    ngx_mail_auth_http_conf_t *conf = child;

    u_char           *p;
    size_t            len;
    ngx_uint_t        i;
    ngx_table_elt_t  *header;

    if (conf->peer == NULL) {
        conf->peer = prev->peer;
        conf->host_header = prev->host_header;
        conf->uri = prev->uri;

        if (conf->peer == NULL) {
            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                          "no \"auth_http\" is defined for server in %s:%ui",
                          conf->file, conf->line);

            return NGX_CONF_ERROR;
        }
    }

    ngx_conf_merge_msec_value(conf->timeout, prev->timeout, 60000);

    if (conf->headers == NULL) {
        conf->headers = prev->headers;
        conf->header = prev->header;
    }

    if (conf->headers && conf->header.len == 0) {
        len = 0;
        header = conf->headers->elts;
        for (i = 0; i < conf->headers->nelts; i++) {
            len += header[i].key.len + 2 + header[i].value.len + 2;
        }

        p = ngx_pnalloc(cf->pool, len);
        if (p == NULL) {
            return NGX_CONF_ERROR;
        }

        conf->header.len = len;
```

```
        conf->header.data = p;

        for (i = 0; i < conf->headers->nelts; i++) {
            p = ngx_cpymem(p, header[i].key.data, header[i].key.len);
            *p++ = ':'; *p++ = ' ';
            p = ngx_cpymem(p, header[i].value.data, header[i].value.len);
            *p++ = CR; *p++ = LF;
        }
    }

    return NGX_CONF_OK;
}


static char *
ngx_mail_auth_http(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_auth_http_conf_t *ahcf = conf;

    ngx_str_t  *value;
    ngx_url_t   u;

    value = cf->args->elts;

    ngx_memzero(&u, sizeof(ngx_url_t));

    u.url = value[1];
    u.default_port = 80;
    u.uri_part = 1;

    if (ngx_strncmp(u.url.data, "http://", 7) == 0) {
        u.url.len -= 7;
        u.url.data += 7;
    }

    if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
        if (u.err) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "%s in auth_http \"%V\"", u.err, &u.url);
        }

        return NGX_CONF_ERROR;
    }

    ahcf->peer = u.addrs;

    if (u.family != AF_UNIX) {
        ahcf->host_header = u.host;

    } else {
        ngx_str_set(&ahcf->host_header, "localhost");
    }

    ahcf->uri = u.uri;

    if (ahcf->uri.len == 0) {
        ngx_str_set(&ahcf->uri, "/");
    }

    return NGX_CONF_OK;
}


static char *
ngx_mail_auth_http_header(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_mail_auth_http_conf_t *ahcf = conf;

    ngx_str_t        *value;
    ngx_table_elt_t  *header;

    if (ahcf->headers == NULL) {
        ahcf->headers = ngx_array_create(cf->pool, 1, sizeof(ngx_table_elt_t));
        if (ahcf->headers == NULL) {
            return NGX_CONF_ERROR;
        }
```

```
1448      }
1449
1450      header = ngx_array_push(ahcf->headers);
1451      if (header == NULL) {
1452          return NGX_CONF_ERROR;
1453      }
1454
1455      value = cf->args->elts;
1456
1457      header->key = value[1];
1458      header->value = value[2];
1459
1460      return NGX_CONF_OK;
1461  }
```

One Level Up Top Level

src/mail/ngx_mail_proxy_module.c - nginx-1.7.10

Global variables defined

- ngx_mail_proxy_commands
- ngx_mail_proxy_module
- ngx_mail_proxy_module_ctx
- smtp_auth_ok

Data types defined

- ngx_mail_proxy_conf_t

Functions defined

- ngx_mail_proxy_block_read
- ngx_mail_proxy_close_session
- ngx_mail_proxy_create_conf
- ngx_mail_proxy_dummy_handler
- ngx_mail_proxy_handler
- ngx_mail_proxy_imap_handler
- ngx_mail_proxy_init
- ngx_mail_proxy_internal_server_error
- ngx_mail_proxy_merge_conf
- ngx_mail_proxy_pop3_handler
- ngx_mail_proxy_read_response
- ngx_mail_proxy_smtp_handler
- ngx_mail_proxy_upstream_error

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_event_connect.h>
12  #include <ngx_mail.h>
13
14
15  typedef struct {
16      ngx_flag_t  enable;
17      ngx_flag_t  pass_error_message;
```

```
    ngx_flag_t  xclient;
    size_t      buffer_size;
    ngx_msec_t  timeout;
} ngx_mail_proxy_conf_t;


static void ngx_mail_proxy_block_read(ngx_event_t *rev);
static void ngx_mail_proxy_pop3_handler(ngx_event_t *rev);
static void ngx_mail_proxy_imap_handler(ngx_event_t *rev);
static void ngx_mail_proxy_smtp_handler(ngx_event_t *rev);
static void ngx_mail_proxy_dummy_handler(ngx_event_t *ev);
static ngx_int_t ngx_mail_proxy_read_response(ngx_mail_session_t *s,
    ngx_uint_t state);
static void ngx_mail_proxy_handler(ngx_event_t *ev);
static void ngx_mail_proxy_upstream_error(ngx_mail_session_t *s);
static void ngx_mail_proxy_internal_server_error(ngx_mail_session_t *s);
static void ngx_mail_proxy_close_session(ngx_mail_session_t *s);
static void *ngx_mail_proxy_create_conf(ngx_conf_t *cf);
static char *ngx_mail_proxy_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);


static ngx_command_t  ngx_mail_proxy_commands[] = {

    { ngx_string("proxy"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_proxy_conf_t, enable),
      NULL },

    { ngx_string("proxy_buffer"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_proxy_conf_t, buffer_size),
      NULL },

    { ngx_string("proxy_timeout"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_proxy_conf_t, timeout),
      NULL },

    { ngx_string("proxy_pass_error_message"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_proxy_conf_t, pass_error_message),
      NULL },

    { ngx_string("xclient"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_proxy_conf_t, xclient),
      NULL },

      ngx_null_command
};


static ngx_mail_module_t  ngx_mail_proxy_module_ctx = {
    NULL,                                  /* protocol */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_mail_proxy_create_conf,            /* create server configuration */
    ngx_mail_proxy_merge_conf              /* merge server configuration */
};


ngx_module_t  ngx_mail_proxy_module = {
    NGX_MODULE_V1,
```

```
    &ngx_mail_proxy_module_ctx,            /* module context */
    ngx_mail_proxy_commands,               /* module directives */
    NGX_MAIL_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
};


static u_char  smtp_auth_ok[] = "235 2.0.0 OK" CRLF;


void
ngx_mail_proxy_init(ngx_mail_session_t *s, ngx_addr_t *peer)
{
    int                        keepalive;
    ngx_int_t                  rc;
    ngx_mail_proxy_ctx_t      *p;
    ngx_mail_proxy_conf_t     *pcf;
    ngx_mail_core_srv_conf_t  *cscf;

    s->connection->log->action = "connecting to upstream";

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

    if (cscf->so_keepalive) {
        keepalive = 1;

        if (setsockopt(s->connection->fd, SOL_SOCKET, SO_KEEPALIVE,
                       (const void *) &keepalive, sizeof(int))
                == -1)
        {
            ngx_log_error(NGX_LOG_ALERT, s->connection->log, ngx_socket_errno,
                          "setsockopt(SO_KEEPALIVE) failed");
        }
    }

    p = ngx_pcalloc(s->connection->pool, sizeof(ngx_mail_proxy_ctx_t));
    if (p == NULL) {
        ngx_mail_session_internal_server_error(s);
        return;
    }

    s->proxy = p;

    p->upstream.sockaddr = peer->sockaddr;
    p->upstream.socklen = peer->socklen;
    p->upstream.name = &peer->name;
    p->upstream.get = ngx_event_get_peer;
    p->upstream.log = s->connection->log;
    p->upstream.log_error = NGX_ERROR_ERR;

    rc = ngx_event_connect_peer(&p->upstream);

    if (rc == NGX_ERROR || rc == NGX_BUSY || rc == NGX_DECLINED) {
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    ngx_add_timer(p->upstream.connection->read, cscf->timeout);

    p->upstream.connection->data = s;
    p->upstream.connection->pool = s->connection->pool;

    s->connection->read->handler = ngx_mail_proxy_block_read;
    p->upstream.connection->write->handler = ngx_mail_proxy_dummy_handler;

    pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);

    s->proxy->buffer = ngx_create_temp_buf(s->connection->pool,
                                           pcf->buffer_size);
```

```
    if (s->proxy->buffer == NULL) {
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    s->out.len = 0;

    switch (s->protocol) {

    case NGX_MAIL_POP3_PROTOCOL:
        p->upstream.connection->read->handler = ngx_mail_proxy_pop3_handler;
        s->mail_state = ngx_pop3_start;
        break;

    case NGX_MAIL_IMAP_PROTOCOL:
        p->upstream.connection->read->handler = ngx_mail_proxy_imap_handler;
        s->mail_state = ngx_imap_start;
        break;

    default: /* NGX_MAIL_SMTP_PROTOCOL */
        p->upstream.connection->read->handler = ngx_mail_proxy_smtp_handler;
        s->mail_state = ngx_smtp_start;
        break;
    }
}


static void
ngx_mail_proxy_block_read(ngx_event_t *rev)
{
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0, "mail proxy block read");

    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
        c = rev->data;
        s = c->data;

        ngx_mail_proxy_close_session(s);
    }
}


static void
ngx_mail_proxy_pop3_handler(ngx_event_t *rev)
{
    u_char                 *p;
    ngx_int_t               rc;
    ngx_str_t               line;
    ngx_connection_t       *c;
    ngx_mail_session_t     *s;
    ngx_mail_proxy_conf_t  *pcf;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                   "mail proxy pop3 auth handler");

    c = rev->data;
    s = c->data;

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT,
                      "upstream timed out");
        c->timedout = 1;
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    rc = ngx_mail_proxy_read_response(s, 0);

    if (rc == NGX_AGAIN) {
        return;
    }

    if (rc == NGX_ERROR) {
        ngx_mail_proxy_upstream_error(s);
```

```
246         return;
247     }
248 
249     switch (s->mail_state) {
250 
251     case ngx_pop3_start:
252         ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0, "mail proxy send user");
253 
254         s->connection->log->action = "sending user name to upstream";
255 
256         line.len = sizeof("USER ")  - 1 + s->login.len + 2;
257         line.data = ngx_pnalloc(c->pool, line.len);
258         if (line.data == NULL) {
259             ngx_mail_proxy_internal_server_error(s);
260             return;
261         }
262 
263         p = ngx_cpymem(line.data, "USER ", sizeof("USER ") - 1);
264         p = ngx_cpymem(p, s->login.data, s->login.len);
265         *p++ = CR; *p = LF;
266 
267         s->mail_state = ngx_pop3_user;
268         break;
269 
270     case ngx_pop3_user:
271         ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0, "mail proxy send pass");
272 
273         s->connection->log->action = "sending password to upstream";
274 
275         line.len = sizeof("PASS ")  - 1 + s->passwd.len + 2;
276         line.data = ngx_pnalloc(c->pool, line.len);
277         if (line.data == NULL) {
278             ngx_mail_proxy_internal_server_error(s);
279             return;
280         }
281 
282         p = ngx_cpymem(line.data, "PASS ", sizeof("PASS ") - 1);
283         p = ngx_cpymem(p, s->passwd.data, s->passwd.len);
284         *p++ = CR; *p = LF;
285 
286         s->mail_state = ngx_pop3_passwd;
287         break;
288 
289     case ngx_pop3_passwd:
290         s->connection->read->handler = ngx_mail_proxy_handler;
291         s->connection->write->handler = ngx_mail_proxy_handler;
292         rev->handler = ngx_mail_proxy_handler;
293         c->write->handler = ngx_mail_proxy_handler;
294 
295         pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);
296         ngx_add_timer(s->connection->read, pcf->timeout);
297         ngx_del_timer(c->read);
298 
299         c->log->action = NULL;
300         ngx_log_error(NGX_LOG_INFO, c->log, 0, "client logged in");
301 
302         ngx_mail_proxy_handler(s->connection->write);
303 
304         return;
305 
306     default:
307 #if (NGX_SUPPRESS_WARN)
308         ngx_str_null(&line);
309 #endif
310         break;
311     }
312 
313     if (c->send(c, line.data, line.len) < (ssize_t) line.len) {
314         /*
315          * we treat the incomplete sending as NGX_ERROR
316          * because it is very strange here
317          */
318         ngx_mail_proxy_internal_server_error(s);
319         return;
320     }
321 
```

```
    s->proxy->buffer->pos = s->proxy->buffer->start;
    s->proxy->buffer->last = s->proxy->buffer->start;
}


static void
ngx_mail_proxy_imap_handler(ngx_event_t *rev)
{
    u_char                  *p;
    ngx_int_t                rc;
    ngx_str_t                line;
    ngx_connection_t        *c;
    ngx_mail_session_t      *s;
    ngx_mail_proxy_conf_t   *pcf;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                   "mail proxy imap auth handler");

    c = rev->data;
    s = c->data;

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT,
                      "upstream timed out");
        c->timedout = 1;
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    rc = ngx_mail_proxy_read_response(s, s->mail_state);

    if (rc == NGX_AGAIN) {
        return;
    }

    if (rc == NGX_ERROR) {
        ngx_mail_proxy_upstream_error(s);
        return;
    }

    switch (s->mail_state) {

    case ngx_imap_start:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                       "mail proxy send login");

        s->connection->log->action = "sending LOGIN command to upstream";

        line.len = s->tag.len + sizeof("LOGIN ") - 1
                   + 1 + NGX_SIZE_T_LEN + 1 + 2;
        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
            ngx_mail_proxy_internal_server_error(s);
            return;
        }

        line.len = ngx_sprintf(line.data, "%VLOGIN {%uz}" CRLF,
                               &s->tag, s->login.len)
                   - line.data;

        s->mail_state = ngx_imap_login;
        break;

    case ngx_imap_login:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0, "mail proxy send user");

        s->connection->log->action = "sending user name to upstream";

        line.len = s->login.len + 1 + 1 + NGX_SIZE_T_LEN + 1 + 2;
        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
            ngx_mail_proxy_internal_server_error(s);
            return;
        }

        line.len = ngx_sprintf(line.data, "%V {%uz}" CRLF,
```

```
                                &s->login, s->passwd.len)
                    - line.data;

        s->mail_state = ngx_imap_user;
        break;

    case ngx_imap_user:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                       "mail proxy send passwd");

        s->connection->log->action = "sending password to upstream";

        line.len = s->passwd.len + 2;
        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
            ngx_mail_proxy_internal_server_error(s);
            return;
        }

        p = ngx_cpymem(line.data, s->passwd.data, s->passwd.len);
        *p++ = CR; *p = LF;

        s->mail_state = ngx_imap_passwd;
        break;

    case ngx_imap_passwd:
        s->connection->read->handler = ngx_mail_proxy_handler;
        s->connection->write->handler = ngx_mail_proxy_handler;
        rev->handler = ngx_mail_proxy_handler;
        c->write->handler = ngx_mail_proxy_handler;

        pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);
        ngx_add_timer(s->connection->read, pcf->timeout);
        ngx_del_timer(c->read);

        c->log->action = NULL;
        ngx_log_error(NGX_LOG_INFO, c->log, 0, "client logged in");

        ngx_mail_proxy_handler(s->connection->write);

        return;

    default:
#if (NGX_SUPPRESS_WARN)
        ngx_str_null(&line);
#endif
        break;
    }

    if (c->send(c, line.data, line.len) < (ssize_t) line.len) {
        /*
         * we treat the incomplete sending as NGX_ERROR
         * because it is very strange here
         */
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    s->proxy->buffer->pos = s->proxy->buffer->start;
    s->proxy->buffer->last = s->proxy->buffer->start;
}


static void
ngx_mail_proxy_smtp_handler(ngx_event_t *rev)
{
    u_char                   *p;
    ngx_int_t                 rc;
    ngx_str_t                 line;
    ngx_buf_t                *b;
    ngx_connection_t         *c;
    ngx_mail_session_t       *s;
    ngx_mail_proxy_conf_t    *pcf;
    ngx_mail_core_srv_conf_t  *cscf;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
```

```
                  "mail proxy smtp auth handler");

    c = rev->data;
    s = c->data;

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT,
                      "upstream timed out");
        c->timedout = 1;
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    rc = ngx_mail_proxy_read_response(s, s->mail_state);

    if (rc == NGX_AGAIN) {
        return;
    }

    if (rc == NGX_ERROR) {
        ngx_mail_proxy_upstream_error(s);
        return;
    }

    switch (s->mail_state) {

    case ngx_smtp_start:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0, "mail proxy send ehlo");

        s->connection->log->action = "sending HELO/EHLO to upstream";

        cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

        line.len = sizeof("HELO ")  - 1 + cscf->server_name.len + 2;
        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
            ngx_mail_proxy_internal_server_error(s);
            return;
        }

        pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);

        p = ngx_cpymem(line.data,
                       ((s->esmtp || pcf->xclient) ? "EHLO " : "HELO "),
                       sizeof("HELO ") - 1);

        p = ngx_cpymem(p, cscf->server_name.data, cscf->server_name.len);
        *p++ = CR; *p = LF;

        if (pcf->xclient) {
            s->mail_state = ngx_smtp_helo_xclient;

        } else if (s->auth_method == NGX_MAIL_AUTH_NONE) {
            s->mail_state = ngx_smtp_helo_from;

        } else {
            s->mail_state = ngx_smtp_helo;
        }

        break;

    case ngx_smtp_helo_xclient:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                       "mail proxy send xclient");

        s->connection->log->action = "sending XCLIENT to upstream";

        line.len = sizeof("XCLIENT ADDR= LOGIN= NAME="
                          CRLF) - 1
                   + s->connection->addr_text.len + s->login.len + s->host.len;

#if (NGX_HAVE_INET6)
        if (s->connection->sockaddr->sa_family == AF_INET6) {
            line.len += sizeof("IPV6:") - 1;
        }
#endif
```

```
        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
            ngx_mail_proxy_internal_server_error(s);
            return;
        }

        p = ngx_cpymem(line.data, "XCLIENT ADDR=", sizeof("XCLIENT ADDR=") - 1);

#if (NGX_HAVE_INET6)
        if (s->connection->sockaddr->sa_family == AF_INET6) {
            p = ngx_cpymem(p, "IPV6:", sizeof("IPV6:") - 1);
        }
#endif

        p = ngx_copy(p, s->connection->addr_text.data,
                     s->connection->addr_text.len);

        if (s->login.len) {
            p = ngx_cpymem(p, " LOGIN=", sizeof(" LOGIN=") - 1);
            p = ngx_copy(p, s->login.data, s->login.len);
        }

        p = ngx_cpymem(p, " NAME=", sizeof(" NAME=") - 1);
        p = ngx_copy(p, s->host.data, s->host.len);

        *p++ = CR; *p++ = LF;

        line.len = p - line.data;

        if (s->smtp_helo.len) {
            s->mail_state = ngx_smtp_xclient_helo;

        } else if (s->auth_method == NGX_MAIL_AUTH_NONE) {
            s->mail_state = ngx_smtp_xclient_from;

        } else {
            s->mail_state = ngx_smtp_xclient;
        }

        break;

    case ngx_smtp_xclient_helo:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                       "mail proxy send client ehlo");

        s->connection->log->action = "sending client HELO/EHLO to upstream";

        line.len = sizeof("HELO " CRLF) - 1 + s->smtp_helo.len;

        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
            ngx_mail_proxy_internal_server_error(s);
            return;
        }

        line.len = ngx_sprintf(line.data,
                       ((s->esmtp) ? "EHLO %V" CRLF : "HELO %V" CRLF),
                       &s->smtp_helo)
                   - line.data;

        s->mail_state = (s->auth_method == NGX_MAIL_AUTH_NONE) ?
                            ngx_smtp_helo_from : ngx_smtp_helo;

        break;

    case ngx_smtp_helo_from:
    case ngx_smtp_xclient_from:
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
                       "mail proxy send mail from");

        s->connection->log->action = "sending MAIL FROM to upstream";

        line.len = s->smtp_from.len + sizeof(CRLF) - 1;
        line.data = ngx_pnalloc(c->pool, line.len);
        if (line.data == NULL) {
```

```
626             ngx_mail_proxy_internal_server_error(s);
627             return;
628         }
629 
630         p = ngx_cpymem(line.data, s->smtp_from.data, s->smtp_from.len);
631         *p++ = CR; *p = LF;
632 
633         s->mail_state = ngx_smtp_from;
634 
635         break;
636 
637     case ngx_smtp_from:
638         ngx_log_debug0(NGX_LOG_DEBUG_MAIL, rev->log, 0,
639                        "mail proxy send rcpt to");
640 
641         s->connection->log->action = "sending RCPT TO to upstream";
642 
643         line.len = s->smtp_to.len + sizeof(CRLF) - 1;
644         line.data = ngx_pnalloc(c->pool, line.len);
645         if (line.data == NULL) {
646             ngx_mail_proxy_internal_server_error(s);
647             return;
648         }
649 
650         p = ngx_cpymem(line.data, s->smtp_to.data, s->smtp_to.len);
651         *p++ = CR; *p = LF;
652 
653         s->mail_state = ngx_smtp_to;
654 
655         break;
656 
657     case ngx_smtp_helo:
658     case ngx_smtp_xclient:
659     case ngx_smtp_to:
660 
661         b = s->proxy->buffer;
662 
663         if (s->auth_method == NGX_MAIL_AUTH_NONE) {
664             b->pos = b->start;
665 
666         } else {
667             ngx_memcpy(b->start, smtp_auth_ok, sizeof(smtp_auth_ok) - 1);
668             b->last = b->start + sizeof(smtp_auth_ok) - 1;
669         }
670 
671         s->connection->read->handler = ngx_mail_proxy_handler;
672         s->connection->write->handler = ngx_mail_proxy_handler;
673         rev->handler = ngx_mail_proxy_handler;
674         c->write->handler = ngx_mail_proxy_handler;
675 
676         pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);
677         ngx_add_timer(s->connection->read, pcf->timeout);
678         ngx_del_timer(c->read);
679 
680         c->log->action = NULL;
681         ngx_log_error(NGX_LOG_INFO, c->log, 0, "client logged in");
682 
683         if (s->buffer->pos == s->buffer->last) {
684             ngx_mail_proxy_handler(s->connection->write);
685 
686         } else {
687             ngx_mail_proxy_handler(c->write);
688         }
689 
690         return;
691 
692     default:
693 #if (NGX_SUPPRESS_WARN)
694         ngx_str_null(&line);
695 #endif
696         break;
697     }
698 
699     if (c->send(c, line.data, line.len) < (ssize_t) line.len) {
700         /*
701          * we treat the incomplete sending as NGX_ERROR
```

```
         * because it is very strange here
         */
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    s->proxy->buffer->pos = s->proxy->buffer->start;
    s->proxy->buffer->last = s->proxy->buffer->start;
}


static void
ngx_mail_proxy_dummy_handler(ngx_event_t *wev)
{
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, wev->log, 0, "mail proxy dummy handler");

    if (ngx_handle_write_event(wev, 0) != NGX_OK) {
        c = wev->data;
        s = c->data;

        ngx_mail_proxy_close_session(s);
    }
}


static ngx_int_t
ngx_mail_proxy_read_response(ngx_mail_session_t *s, ngx_uint_t state)
{
    u_char                *p, *m;
    ssize_t                n;
    ngx_buf_t             *b;
    ngx_mail_proxy_conf_t  *pcf;

    s->connection->log->action = "reading response from upstream";

    b = s->proxy->buffer;

    n = s->proxy->upstream.connection->recv(s->proxy->upstream.connection,
                                            b->last, b->end - b->last);

    if (n == NGX_ERROR || n == 0) {
        return NGX_ERROR;
    }

    if (n == NGX_AGAIN) {
        return NGX_AGAIN;
    }

    b->last += n;

    if (b->last - b->pos < 4) {
        return NGX_AGAIN;
    }

    if (*(b->last - 2) != CR || *(b->last - 1) != LF) {
        if (b->last == b->end) {
            *(b->last - 1) = '\0';
            ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                          "upstream sent too long response line: \"%s\"",
                          b->pos);
            return NGX_ERROR;
        }

        return NGX_AGAIN;
    }

    p = b->pos;

    switch (s->protocol) {

    case NGX_MAIL_POP3_PROTOCOL:
        if (p[0] == '+' && p[1] == 'O' && p[2] == 'K') {
            return NGX_OK;
```

```
        }
        break;

    case NGX_MAIL_IMAP_PROTOCOL:
        switch (state) {

        case ngx_imap_start:
            if (p[0] == '*' && p[1] == ' ' && p[2] == 'O' && p[3] == 'K') {
                return NGX_OK;
            }
            break;

        case ngx_imap_login:
        case ngx_imap_user:
            if (p[0] == '+') {
                return NGX_OK;
            }
            break;

        case ngx_imap_passwd:
            if (ngx_strncmp(p, s->tag.data, s->tag.len) == 0) {
                p += s->tag.len;
                if (p[0] == 'O' && p[1] == 'K') {
                    return NGX_OK;
                }
            }
            break;
        }

        break;

    default: /* NGX_MAIL_SMTP_PROTOCOL */

        if (p[3] == '-') {
            /* multiline reply, check if we got last line */

            m = b->last - (sizeof(CRLF "200" CRLF) - 1);

            while (m > p) {
                if (m[0] == CR && m[1] == LF) {
                    break;
                }

                m--;
            }

            if (m <= p || m[5] == '-') {
                return NGX_AGAIN;
            }
        }

        switch (state) {

        case ngx_smtp_start:
            if (p[0] == '2' && p[1] == '2' && p[2] == '0') {
                return NGX_OK;
            }
            break;

        case ngx_smtp_helo:
        case ngx_smtp_helo_xclient:
        case ngx_smtp_helo_from:
        case ngx_smtp_from:
            if (p[0] == '2' && p[1] == '5' && p[2] == '0') {
                return NGX_OK;
            }
            break;

        case ngx_smtp_xclient:
        case ngx_smtp_xclient_from:
        case ngx_smtp_xclient_helo:
            if (p[0] == '2' && (p[1] == '2' || p[1] == '5') && p[2] == '0') {
                return NGX_OK;
            }
            break;

```

```
        case ngx_smtp_to:
            return NGX_OK;
        }

        break;
    }

    pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);

    if (pcf->pass_error_message == 0) {
        *(b->last - 2) = '\0';
        ngx_log_error(NGX_LOG_ERR, s->connection->log, 0,
                      "upstream sent invalid response: \"%s\"", p);
        return NGX_ERROR;
    }

    s->out.len = b->last - p - 2;
    s->out.data = p;

    ngx_log_error(NGX_LOG_INFO, s->connection->log, 0,
                  "upstream sent invalid response: \"%V\"", &s->out);

    s->out.len = b->last - b->pos;
    s->out.data = b->pos;

    return NGX_ERROR;
}


static void
ngx_mail_proxy_handler(ngx_event_t *ev)
{
    char                    *action, *recv_action, *send_action;
    size_t                   size;
    ssize_t                  n;
    ngx_buf_t               *b;
    ngx_uint_t               do_write;
    ngx_connection_t        *c, *src, *dst;
    ngx_mail_session_t      *s;
    ngx_mail_proxy_conf_t   *pcf;

    c = ev->data;
    s = c->data;

    if (ev->timedout) {
        c->log->action = "proxying";

        if (c == s->connection) {
            ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT,
                          "client timed out");
            c->timedout = 1;

        } else {
            ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT,
                          "upstream timed out");
        }

        ngx_mail_proxy_close_session(s);
        return;
    }

    if (c == s->connection) {
        if (ev->write) {
            recv_action = "proxying and reading from upstream";
            send_action = "proxying and sending to client";
            src = s->proxy->upstream.connection;
            dst = c;
            b = s->proxy->buffer;

        } else {
            recv_action = "proxying and reading from client";
            send_action = "proxying and sending to upstream";
            src = c;
            dst = s->proxy->upstream.connection;
            b = s->buffer;
        }
```

```
930
931     } else {
932         if (ev->write) {
933             recv_action = "proxying and reading from client";
934             send_action = "proxying and sending to upstream";
935             src = s->connection;
936             dst = c;
937             b = s->buffer;
938
939         } else {
940             recv_action = "proxying and reading from upstream";
941             send_action = "proxying and sending to client";
942             src = c;
943             dst = s->connection;
944             b = s->proxy->buffer;
945         }
946     }
947
948     do_write = ev->write ? 1 : 0;
949
950     ngx_log_debug3(NGX_LOG_DEBUG_MAIL, ev->log, 0,
951                    "mail proxy handler: %d, #%d > #%d",
952                    do_write, src->fd, dst->fd);
953
954     for ( ;; ) {
955
956         if (do_write) {
957
958             size = b->last - b->pos;
959
960             if (size && dst->write->ready) {
961                 c->log->action = send_action;
962
963                 n = dst->send(dst, b->pos, size);
964
965                 if (n == NGX_ERROR) {
966                     ngx_mail_proxy_close_session(s);
967                     return;
968                 }
969
970                 if (n > 0) {
971                     b->pos += n;
972
973                     if (b->pos == b->last) {
974                         b->pos = b->start;
975                         b->last = b->start;
976                     }
977                 }
978             }
979         }
980
981         size = b->end - b->last;
982
983         if (size && src->read->ready) {
984             c->log->action = recv_action;
985
986             n = src->recv(src, b->last, size);
987
988             if (n == NGX_AGAIN || n == 0) {
989                 break;
990             }
991
992             if (n > 0) {
993                 do_write = 1;
994                 b->last += n;
995
996                 continue;
997             }
998
999             if (n == NGX_ERROR) {
1000                 src->read->eof = 1;
1001             }
1002         }
1003
1004         break;
1005     }
```

```
1006
1007        c->log->action = "proxying";
1008
1009        if ((s->connection->read->eof && s->buffer->pos == s->buffer->last)
1010            || (s->proxy->upstream.connection->read->eof
1011                && s->proxy->buffer->pos == s->proxy->buffer->last)
1012            || (s->connection->read->eof
1013                && s->proxy->upstream.connection->read->eof))
1014        {
1015            action = c->log->action;
1016            c->log->action = NULL;
1017            ngx_log_error(NGX_LOG_INFO, c->log, 0, "proxied session done");
1018            c->log->action = action;
1019
1020            ngx_mail_proxy_close_session(s);
1021            return;
1022        }
1023
1024        if (ngx_handle_write_event(dst->write, 0) != NGX_OK) {
1025            ngx_mail_proxy_close_session(s);
1026            return;
1027        }
1028
1029        if (ngx_handle_read_event(dst->read, 0) != NGX_OK) {
1030            ngx_mail_proxy_close_session(s);
1031            return;
1032        }
1033
1034        if (ngx_handle_write_event(src->write, 0) != NGX_OK) {
1035            ngx_mail_proxy_close_session(s);
1036            return;
1037        }
1038
1039        if (ngx_handle_read_event(src->read, 0) != NGX_OK) {
1040            ngx_mail_proxy_close_session(s);
1041            return;
1042        }
1043
1044        if (c == s->connection) {
1045            pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);
1046            ngx_add_timer(c->read, pcf->timeout);
1047        }
1048    }
1049
1050
1051    static void
1052    ngx_mail_proxy_upstream_error(ngx_mail_session_t *s)
1053    {
1054        if (s->proxy->upstream.connection) {
1055            ngx_log_debug1(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
1056                           "close mail proxy connection: %d",
1057                           s->proxy->upstream.connection->fd);
1058
1059            ngx_close_connection(s->proxy->upstream.connection);
1060        }
1061
1062        if (s->out.len == 0) {
1063            ngx_mail_session_internal_server_error(s);
1064            return;
1065        }
1066
1067        s->quit = 1;
1068        ngx_mail_send(s->connection->write);
1069    }
1070
1071
1072    static void
1073    ngx_mail_proxy_internal_server_error(ngx_mail_session_t *s)
1074    {
1075        if (s->proxy->upstream.connection) {
1076            ngx_log_debug1(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
1077                           "close mail proxy connection: %d",
1078                           s->proxy->upstream.connection->fd);
1079
1080            ngx_close_connection(s->proxy->upstream.connection);
1081        }
```

```
1082
1083        ngx_mail_session_internal_server_error(s);
1084    }
1085
1086
1087    static void
1088    ngx_mail_proxy_close_session(ngx_mail_session_t *s)
1089    {
1090        if (s->proxy->upstream.connection) {
1091            ngx_log_debug1(NGX_LOG_DEBUG_MAIL, s->connection->log, 0,
1092                           "close mail proxy connection: %d",
1093                           s->proxy->upstream.connection->fd);
1094
1095            ngx_close_connection(s->proxy->upstream.connection);
1096        }
1097
1098        ngx_mail_close_connection(s->connection);
1099    }
1100
1101
1102    static void *
1103    ngx_mail_proxy_create_conf(ngx_conf_t *cf)
1104    {
1105        ngx_mail_proxy_conf_t  *pcf;
1106
1107        pcf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_proxy_conf_t));
1108        if (pcf == NULL) {
1109            return NULL;
1110        }
1111
1112        pcf->enable = NGX_CONF_UNSET;
1113        pcf->pass_error_message = NGX_CONF_UNSET;
1114        pcf->xclient = NGX_CONF_UNSET;
1115        pcf->buffer_size = NGX_CONF_UNSET_SIZE;
1116        pcf->timeout = NGX_CONF_UNSET_MSEC;
1117
1118        return pcf;
1119    }
1120
1121
1122    static char *
1123    ngx_mail_proxy_merge_conf(ngx_conf_t *cf, void *parent, void *child)
1124    {
1125        ngx_mail_proxy_conf_t *prev = parent;
1126        ngx_mail_proxy_conf_t *conf = child;
1127
1128        ngx_conf_merge_value(conf->enable, prev->enable, 0);
1129        ngx_conf_merge_value(conf->pass_error_message, prev->pass_error_message, 0);
1130        ngx_conf_merge_value(conf->xclient, prev->xclient, 1);
1131        ngx_conf_merge_size_value(conf->buffer_size, prev->buffer_size,
1132                                  (size_t) ngx_pagesize);
1133        ngx_conf_merge_msec_value(conf->timeout, prev->timeout, 24 * 60 * 60000);
1134
1135        return NGX_CONF_OK;
1136    }
```

One Level Up Top Level

src/mail/ngx_mail_parse.c - nginx-1.7.10

Functions defined

- ngx_mail_auth_parse
- ngx_mail_imap_parse_command
- ngx_mail_pop3_parse_command
- ngx_mail_smtp_parse_command

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_pop3_module.h>
13  #include <ngx_mail_imap_module.h>
14  #include <ngx_mail_smtp_module.h>
15
16
17  ngx_int_t
18  ngx_mail_pop3_parse_command(ngx_mail_session_t *s)
19  {
20      u_char      ch, *p, *c, c0, c1, c2, c3;
21      ngx_str_t  *arg;
22      enum {
23          sw_start = 0,
24          sw_spaces_before_argument,
25          sw_argument,
26          sw_almost_done
27      } state;
28
29      state = s->state;
30
31      for (p = s->buffer->pos; p < s->buffer->last; p++) {
32          ch = *p;
33
34          switch (state) {
35
36          /* POP3 command */
37          case sw_start:
38              if (ch == ' ' || ch == CR || ch == LF) {
39                  c = s->buffer->start;
40
41                  if (p - c == 4) {
42
43                      c0 = ngx_toupper(c[0]);
44                      c1 = ngx_toupper(c[1]);
45                      c2 = ngx_toupper(c[2]);
46                      c3 = ngx_toupper(c[3]);
47
48                      if (c0 == 'U' && c1 == 'S' && c2 == 'E' && c3 == 'R')
49                      {
50                          s->command = NGX_POP3_USER;
51
52                      } else if (c0 == 'P' && c1 == 'A' && c2 == 'S' && c3 == 'S')
53                      {
54                          s->command = NGX_POP3_PASS;
55
56                      } else if (c0 == 'A' && c1 == 'P' && c2 == 'O' && c3 == 'P')
```

```
                    {
                        s->command = NGX_POP3_APOP;

                    } else if (c0 == 'Q' && c1 == 'U' && c2 == 'I' && c3 == 'T')
                    {
                        s->command = NGX_POP3_QUIT;

                    } else if (c0 == 'C' && c1 == 'A' && c2 == 'P' && c3 == 'A')
                    {
                        s->command = NGX_POP3_CAPA;

                    } else if (c0 == 'A' && c1 == 'U' && c2 == 'T' && c3 == 'H')
                    {
                        s->command = NGX_POP3_AUTH;

                    } else if (c0 == 'N' && c1 == 'O' && c2 == 'O' && c3 == 'P')
                    {
                        s->command = NGX_POP3_NOOP;
#if (NGX_MAIL_SSL)
                    } else if (c0 == 'S' && c1 == 'T' && c2 == 'L' && c3 == 'S')
                    {
                        s->command = NGX_POP3_STLS;
#endif
                    } else {
                        goto invalid;
                    }

                } else {
                    goto invalid;
                }

                switch (ch) {
                case ' ':
                    state = sw_spaces_before_argument;
                    break;
                case CR:
                    state = sw_almost_done;
                    break;
                case LF:
                    goto done;
                }
                break;
            }

             if ((ch < 'A' || ch > 'Z') && (ch < 'a' || ch > 'z')) {
                 goto invalid;
             }

             break;

        case sw_spaces_before_argument:
            switch (ch) {
            case ' ':
                break;
            case CR:
                state = sw_almost_done;
                s->arg_end = p;
                break;
            case LF:
                s->arg_end = p;
                goto done;
            default:
                if (s->args.nelts <= 2) {
                    state = sw_argument;
                    s->arg_start = p;
                    break;
                }
                goto invalid;
            }
            break;

        case sw_argument:
            switch (ch) {

            case ' ':
```

```
                /*
                 * the space should be considered as part of the at username
                 * or password, but not of argument in other commands
                 */

                if (s->command == NGX_POP3_USER
                    || s->command == NGX_POP3_PASS)
                {
                    break;
                }

                /* fall through */

            case CR:
            case LF:
                arg = ngx_array_push(&s->args);
                if (arg == NULL) {
                    return NGX_ERROR;
                }
                arg->len = p - s->arg_start;
                arg->data = s->arg_start;
                s->arg_start = NULL;

                switch (ch) {
                case ' ':
                    state = sw_spaces_before_argument;
                    break;
                case CR:
                    state = sw_almost_done;
                    break;
                case LF:
                    goto done;
                }
                break;

            default:
                break;
            }
            break;

        case sw_almost_done:
            switch (ch) {
            case LF:
                goto done;
            default:
                goto invalid;
            }
        }
    }

    s->buffer->pos = p;
    s->state = state;

    return NGX_AGAIN;

done:

    s->buffer->pos = p + 1;

    if (s->arg_start) {
        arg = ngx_array_push(&s->args);
        if (arg == NULL) {
            return NGX_ERROR;
        }
        arg->len = s->arg_end - s->arg_start;
        arg->data = s->arg_start;
        s->arg_start = NULL;
    }

    s->state = (s->command != NGX_POP3_AUTH) ? sw_start : sw_argument;

    return NGX_OK;

invalid:

    s->state = sw_start;
```

```
    s->arg_start = NULL;

    return NGX_MAIL_PARSE_INVALID_COMMAND;
}


ngx_int_t
ngx_mail_imap_parse_command(ngx_mail_session_t *s)
{
    u_char      ch, *p, *c;
    ngx_str_t  *arg;
    enum {
        sw_start = 0,
        sw_spaces_before_command,
        sw_command,
        sw_spaces_before_argument,
        sw_argument,
        sw_backslash,
        sw_literal,
        sw_no_sync_literal_argument,
        sw_start_literal_argument,
        sw_literal_argument,
        sw_end_literal_argument,
        sw_almost_done
    } state;

    state = s->state;

    for (p = s->buffer->pos; p < s->buffer->last; p++) {
        ch = *p;

        switch (state) {

        /* IMAP tag */
        case sw_start:
            switch (ch) {
            case ' ':
                s->tag.len = p - s->buffer->start + 1;
                s->tag.data = s->buffer->start;
                state = sw_spaces_before_command;
                break;
            case CR:
                s->state = sw_start;
                return NGX_MAIL_PARSE_INVALID_COMMAND;
            case LF:
                s->state = sw_start;
                return NGX_MAIL_PARSE_INVALID_COMMAND;
            }
            break;

        case sw_spaces_before_command:
            switch (ch) {
            case ' ':
                break;
            case CR:
                s->state = sw_start;
                return NGX_MAIL_PARSE_INVALID_COMMAND;
            case LF:
                s->state = sw_start;
                return NGX_MAIL_PARSE_INVALID_COMMAND;
            default:
                s->cmd_start = p;
                state = sw_command;
                break;
            }
            break;

        case sw_command:
            if (ch == ' ' || ch == CR || ch == LF) {

                c = s->cmd_start;

                switch (p - c) {

                case 4:
                    if ((c[0] == 'N' || c[0] == 'n')
```

```
                        && (c[1] == 'O'|| c[1] == 'o')
                        && (c[2] == 'O'|| c[2] == 'o')
                        && (c[3] == 'P'|| c[3] == 'p'))
                    {
                        s->command = NGX_IMAP_NOOP;

                    } else {
                        goto invalid;
                    }
                    break;

                case 5:
                    if ((c[0] == 'L'|| c[0] == 'l')
                        && (c[1] == 'O'|| c[1] == 'o')
                        && (c[2] == 'G'|| c[2] == 'g')
                        && (c[3] == 'I'|| c[3] == 'i')
                        && (c[4] == 'N'|| c[4] == 'n'))
                    {
                        s->command = NGX_IMAP_LOGIN;

                    } else {
                        goto invalid;
                    }
                    break;

                case 6:
                    if ((c[0] == 'L'|| c[0] == 'l')
                        && (c[1] == 'O'|| c[1] == 'o')
                        && (c[2] == 'G'|| c[2] == 'g')
                        && (c[3] == 'O'|| c[3] == 'o')
                        && (c[4] == 'U'|| c[4] == 'u')
                        && (c[5] == 'T'|| c[5] == 't'))
                    {
                        s->command = NGX_IMAP_LOGOUT;

                    } else {
                        goto invalid;
                    }
                    break;

#if (NGX_MAIL_SSL)
                case 8:
                    if ((c[0] == 'S'|| c[0] == 's')
                        && (c[1] == 'T'|| c[1] == 't')
                        && (c[2] == 'A'|| c[2] == 'a')
                        && (c[3] == 'R'|| c[3] == 'r')
                        && (c[4] == 'T'|| c[4] == 't')
                        && (c[5] == 'T'|| c[5] == 't')
                        && (c[6] == 'L'|| c[6] == 'l')
                        && (c[7] == 'S'|| c[7] == 's'))
                    {
                        s->command = NGX_IMAP_STARTTLS;

                    } else {
                        goto invalid;
                    }
                    break;
#endif

                case 10:
                    if ((c[0] == 'C'|| c[0] == 'c')
                        && (c[1] == 'A'|| c[1] == 'a')
                        && (c[2] == 'P'|| c[2] == 'p')
                        && (c[3] == 'A'|| c[3] == 'a')
                        && (c[4] == 'B'|| c[4] == 'b')
                        && (c[5] == 'I'|| c[5] == 'i')
                        && (c[6] == 'L'|| c[6] == 'l')
                        && (c[7] == 'I'|| c[7] == 'i')
                        && (c[8] == 'T'|| c[8] == 't')
                        && (c[9] == 'Y'|| c[9] == 'y'))
                    {
                        s->command = NGX_IMAP_CAPABILITY;

                    } else {
                        goto invalid;
                    }
```

```
                break;

            case 12:
                if ((c[0] == 'A'|| c[0] == 'a')
                    && (c[1] == 'U'|| c[1] == 'u')
                    && (c[2] == 'T'|| c[2] == 't')
                    && (c[3] == 'H'|| c[3] == 'h')
                    && (c[4] == 'E'|| c[4] == 'e')
                    && (c[5] == 'N'|| c[5] == 'n')
                    && (c[6] == 'T'|| c[6] == 't')
                    && (c[7] == 'I'|| c[7] == 'i')
                    && (c[8] == 'C'|| c[8] == 'c')
                    && (c[9] == 'A'|| c[9] == 'a')
                    && (c[10] == 'T'|| c[10] == 't')
                    && (c[11] == 'E'|| c[11] == 'e'))
                {
                    s->command = NGX_IMAP_AUTHENTICATE;

                } else {
                    goto invalid;
                }
                break;

            default:
                goto invalid;
            }

            switch (ch) {
            case ' ':
                state = sw_spaces_before_argument;
                break;
            case CR:
                state = sw_almost_done;
                break;
            case LF:
                goto done;
            }
            break;
        }

        if ((ch < 'A' || ch > 'Z') && (ch < 'a' || ch > 'z')) {
            goto invalid;
        }

        break;

    case sw_spaces_before_argument:
        switch (ch) {
        case ' ':
            break;
        case CR:
            state = sw_almost_done;
            s->arg_end = p;
            break;
        case LF:
            s->arg_end = p;
            goto done;
        case '"':
            if (s->args.nelts <= 2) {
                s->quoted = 1;
                s->arg_start = p + 1;
                state = sw_argument;
                break;
            }
            goto invalid;
        case '{':
            if (s->args.nelts <= 2) {
                state = sw_literal;
                break;
            }
            goto invalid;
        default:
            if (s->args.nelts <= 2) {
                s->arg_start = p;
                state = sw_argument;
                break;
```

```
            }
            goto invalid;
        }
        break;

    case sw_argument:
        if (ch == ' ' && s->quoted) {
            break;
        }

        switch (ch) {
        case '"':
            if (!s->quoted) {
                break;
            }
            s->quoted = 0;
            /* fall through */
        case ' ':
        case CR:
        case LF:
            arg = ngx_array_push(&s->args);
            if (arg == NULL) {
                return NGX_ERROR;
            }
            arg->len = p - s->arg_start;
            arg->data = s->arg_start;
            s->arg_start = NULL;

            switch (ch) {
            case '"':
            case ' ':
                state = sw_spaces_before_argument;
                break;
            case CR:
                state = sw_almost_done;
                break;
            case LF:
                goto done;
            }
            break;
        case '\\':
            if (s->quoted) {
                s->backslash = 1;
                state = sw_backslash;
            }
            break;
        }
        break;

    case sw_backslash:
        switch (ch) {
        case CR:
        case LF:
            goto invalid;
        default:
            state = sw_argument;
        }
        break;

    case sw_literal:
        if (ch >= '0' && ch <= '9') {
            s->literal_len = s->literal_len * 10 + (ch - '0');
            break;
        }
        if (ch == '}') {
            state = sw_start_literal_argument;
            break;
        }
        if (ch == '+') {
            state = sw_no_sync_literal_argument;
            break;
        }
        goto invalid;

    case sw_no_sync_literal_argument:
        if (ch == '}') {
```

```
                s->no_sync_literal = 1;
                state = sw_start_literal_argument;
                break;
            }
            goto invalid;

        case sw_start_literal_argument:
            switch (ch) {
            case CR:
                break;
            case LF:
                s->buffer->pos = p + 1;
                s->arg_start = p + 1;
                if (s->no_sync_literal == 0) {
                    s->state = sw_literal_argument;
                    return NGX_IMAP_NEXT;
                }
                state = sw_literal_argument;
                s->no_sync_literal = 0;
                break;
            default:
                goto invalid;
            }
            break;

        case sw_literal_argument:
            if (s->literal_len && --s->literal_len) {
                break;
            }

            arg = ngx_array_push(&s->args);
            if (arg == NULL) {
                return NGX_ERROR;
            }
            arg->len = p + 1 - s->arg_start;
            arg->data = s->arg_start;
            s->arg_start = NULL;
            state = sw_end_literal_argument;

            break;

        case sw_end_literal_argument:
            switch (ch) {
            case '{':
                if (s->args.nelts <= 2) {
                    state = sw_literal;
                    break;
                }
                goto invalid;
            case CR:
                state = sw_almost_done;
                break;
            case LF:
                goto done;
            default:
                state = sw_spaces_before_argument;
                break;
            }
            break;

        case sw_almost_done:
            switch (ch) {
            case LF:
                goto done;
            default:
                goto invalid;
            }
        }
    }

    s->buffer->pos = p;
    s->state = state;

    return NGX_AGAIN;

done:
```

```
589
590     s->buffer->pos = p + 1;
591
592     if (s->arg_start) {
593         arg = ngx_array_push(&s->args);
594         if (arg == NULL) {
595             return NGX_ERROR;
596         }
597         arg->len = s->arg_end - s->arg_start;
598         arg->data = s->arg_start;
599
600         s->arg_start = NULL;
601         s->cmd_start = NULL;
602         s->quoted = 0;
603         s->no_sync_literal = 0;
604         s->literal_len = 0;
605     }
606
607     s->state = (s->command != NGX_IMAP_AUTHENTICATE) ? sw_start : sw_argument;
608
609     return NGX_OK;
610
611 invalid:
612
613     s->state = sw_start;
614     s->quoted = 0;
615     s->no_sync_literal = 0;
616     s->literal_len = 0;
617
618     return NGX_MAIL_PARSE_INVALID_COMMAND;
619 }
620
621
622 ngx_int_t
623 ngx_mail_smtp_parse_command(ngx_mail_session_t *s)
624 {
625     u_char      ch, *p, *c, c0, c1, c2, c3;
626     ngx_str_t  *arg;
627     enum {
628         sw_start = 0,
629         sw_command,
630         sw_invalid,
631         sw_spaces_before_argument,
632         sw_argument,
633         sw_almost_done
634     } state;
635
636     state = s->state;
637
638     for (p = s->buffer->pos; p < s->buffer->last; p++) {
639         ch = *p;
640
641         switch (state) {
642
643         /* SMTP command */
644         case sw_start:
645             s->cmd_start = p;
646             state = sw_command;
647
648             /* fall through */
649
650         case sw_command:
651             if (ch == ' ' || ch == CR || ch == LF) {
652                 c = s->cmd_start;
653
654                 if (p - c == 4) {
655
656                     c0 = ngx_toupper(c[0]);
657                     c1 = ngx_toupper(c[1]);
658                     c2 = ngx_toupper(c[2]);
659                     c3 = ngx_toupper(c[3]);
660
661                     if (c0 == 'H' && c1 == 'E' && c2 == 'L' && c3 == 'O')
662                     {
663                         s->command = NGX_SMTP_HELO;
664
```

```
                    } else if (c0 == 'E' && c1 == 'H' && c2 == 'L' && c3 == 'O')
                    {
                        s->command = NGX_SMTP_EHLO;

                    } else if (c0 == 'Q' && c1 == 'U' && c2 == 'I' && c3 == 'T')
                    {
                        s->command = NGX_SMTP_QUIT;

                    } else if (c0 == 'A' && c1 == 'U' && c2 == 'T' && c3 == 'H')
                    {
                        s->command = NGX_SMTP_AUTH;

                    } else if (c0 == 'N' && c1 == 'O' && c2 == 'O' && c3 == 'P')
                    {
                        s->command = NGX_SMTP_NOOP;

                    } else if (c0 == 'M' && c1 == 'A' && c2 == 'I' && c3 == 'L')
                    {
                        s->command = NGX_SMTP_MAIL;

                    } else if (c0 == 'R' && c1 == 'S' && c2 == 'E' && c3 == 'T')
                    {
                        s->command = NGX_SMTP_RSET;

                    } else if (c0 == 'R' && c1 == 'C' && c2 == 'P' && c3 == 'T')
                    {
                        s->command = NGX_SMTP_RCPT;

                    } else if (c0 == 'V' && c1 == 'R' && c2 == 'F' && c3 == 'Y')
                    {
                        s->command = NGX_SMTP_VRFY;

                    } else if (c0 == 'E' && c1 == 'X' && c2 == 'P' && c3 == 'N')
                    {
                        s->command = NGX_SMTP_EXPN;

                    } else if (c0 == 'H' && c1 == 'E' && c2 == 'L' && c3 == 'P')
                    {
                        s->command = NGX_SMTP_HELP;

                    } else {
                        goto invalid;
                    }
#if (NGX_MAIL_SSL)
                } else if (p - c == 8) {

                    if ((c[0] == 'S'|| c[0] == 's')
                        && (c[1] == 'T'|| c[1] == 't')
                        && (c[2] == 'A'|| c[2] == 'a')
                        && (c[3] == 'R'|| c[3] == 'r')
                        && (c[4] == 'T'|| c[4] == 't')
                        && (c[5] == 'T'|| c[5] == 't')
                        && (c[6] == 'L'|| c[6] == 'l')
                        && (c[7] == 'S'|| c[7] == 's'))
                    {
                        s->command = NGX_SMTP_STARTTLS;

                    } else {
                        goto invalid;
                    }
#endif
                } else {
                    goto invalid;
                }

                s->cmd.data = s->cmd_start;
                s->cmd.len = p - s->cmd_start;

                switch (ch) {
                case ' ':
                    state = sw_spaces_before_argument;
                    break;
                case CR:
                    state = sw_almost_done;
                    break;
                case LF:
```

```
                    goto done;
                }
                break;
            }

            if ((ch < 'A' || ch > 'Z') && (ch < 'a' || ch > 'z')) {
                goto invalid;
            }

            break;

        case sw_invalid:
            goto invalid;

        case sw_spaces_before_argument:
            switch (ch) {
            case ' ':
                break;
            case CR:
                state = sw_almost_done;
                s->arg_end = p;
                break;
            case LF:
                s->arg_end = p;
                goto done;
            default:
                if (s->args.nelts <= 10) {
                    state = sw_argument;
                    s->arg_start = p;
                    break;
                }
                goto invalid;
            }
            break;

        case sw_argument:
            switch (ch) {
            case ' ':
            case CR:
            case LF:
                arg = ngx_array_push(&s->args);
                if (arg == NULL) {
                    return NGX_ERROR;
                }
                arg->len = p - s->arg_start;
                arg->data = s->arg_start;
                s->arg_start = NULL;

                switch (ch) {
                case ' ':
                    state = sw_spaces_before_argument;
                    break;
                case CR:
                    state = sw_almost_done;
                    break;
                case LF:
                    goto done;
                }
                break;

            default:
                break;
            }
            break;

        case sw_almost_done:
            switch (ch) {
            case LF:
                goto done;
            default:
                goto invalid;
            }
        }
    }

    s->buffer->pos = p;
```

```
    s->state = state;

    return NGX_AGAIN;

done:

    s->buffer->pos = p + 1;

    if (s->arg_start) {
        arg = ngx_array_push(&s->args);
        if (arg == NULL) {
            return NGX_ERROR;
        }
        arg->len = s->arg_end - s->arg_start;
        arg->data = s->arg_start;
        s->arg_start = NULL;
    }

    s->state = (s->command != NGX_SMTP_AUTH) ? sw_start : sw_argument;

    return NGX_OK;

invalid:

    s->state = sw_invalid;
    s->arg_start = NULL;

    /* skip invalid command till LF */

    for (p = s->buffer->pos; p < s->buffer->last; p++) {
        if (*p == LF) {
            s->state = sw_start;
            p++;
            break;
        }
    }

    s->buffer->pos = p;

    return NGX_MAIL_PARSE_INVALID_COMMAND;
}


ngx_int_t
ngx_mail_auth_parse(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t                 *arg;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif

    if (s->args.nelts == 0) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    arg = s->args.elts;

    if (arg[0].len == 5) {

        if (ngx_strncasecmp(arg[0].data, (u_char *) "LOGIN", 5) == 0) {

            if (s->args.nelts == 1) {
                return NGX_MAIL_AUTH_LOGIN;
            }

            if (s->args.nelts == 2) {
                return NGX_MAIL_AUTH_LOGIN_USERNAME;
            }

            return NGX_MAIL_PARSE_INVALID_COMMAND;
        }

        if (ngx_strncasecmp(arg[0].data, (u_char *) "PLAIN", 5) == 0) {
```

```
893
894             if (s->args.nelts == 1) {
895                 return NGX_MAIL_AUTH_PLAIN;
896             }
897
898             if (s->args.nelts == 2) {
899                 return ngx_mail_auth_plain(s, c, 1);
900             }
901         }
902
903         return NGX_MAIL_PARSE_INVALID_COMMAND;
904     }
905
906     if (arg[0].len == 8) {
907
908         if (s->args.nelts != 1) {
909             return NGX_MAIL_PARSE_INVALID_COMMAND;
910         }
911
912         if (ngx_strncasecmp(arg[0].data, (u_char *) "CRAM-MD5", 8) == 0) {
913             return NGX_MAIL_AUTH_CRAM_MD5;
914         }
915     }
916
917     return NGX_MAIL_PARSE_INVALID_COMMAND;
918 }
```

One Level Up Top Level

src/mail/ngx_mail_pop3_module.c - nginx-1.7.10

Global variables defined

- ngx_mail_pop3_auth_cram_md5_capability
- ngx_mail_pop3_auth_methods
- ngx_mail_pop3_auth_plain_capability
- ngx_mail_pop3_commands
- ngx_mail_pop3_default_capabilities
- ngx_mail_pop3_module
- ngx_mail_pop3_module_ctx
- ngx_mail_pop3_protocol

Functions defined

- ngx_mail_pop3_create_srv_conf
- ngx_mail_pop3_merge_srv_conf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_pop3_module.h>
13
14
15  static void *ngx_mail_pop3_create_srv_conf(ngx_conf_t *cf);
16  static char *ngx_mail_pop3_merge_srv_conf(ngx_conf_t *cf, void *parent,
17      void *child);
18
19
20  static ngx_str_t  ngx_mail_pop3_default_capabilities[] = {
21      ngx_string("TOP"),
22      ngx_string("USER"),
23      ngx_string("UIDL"),
24      ngx_null_string
25  };
26
27
28  static ngx_conf_bitmask_t  ngx_mail_pop3_auth_methods[] = {
29      { ngx_string("plain"), NGX_MAIL_AUTH_PLAIN_ENABLED },
30      { ngx_string("apop"), NGX_MAIL_AUTH_APOP_ENABLED },
31      { ngx_string("cram-md5"), NGX_MAIL_AUTH_CRAM_MD5_ENABLED },
32      { ngx_null_string, 0 }
33  };
34
35
36  static ngx_str_t  ngx_mail_pop3_auth_plain_capability =
37      ngx_string("+OK methods supported:" CRLF
38                 "LOGIN" CRLF
```

```
                "PLAIN" CRLF
                "." CRLF);


static ngx_str_t  ngx_mail_pop3_auth_cram_md5_capability =
    ngx_string("+OK methods supported:" CRLF
                "LOGIN" CRLF
                "PLAIN" CRLF
                "CRAM-MD5" CRLF
                "." CRLF);


static ngx_mail_protocol_t  ngx_mail_pop3_protocol = {
    ngx_string("pop3"),
    { 110, 995, 0, 0 },
    NGX_MAIL_POP3_PROTOCOL,

    ngx_mail_pop3_init_session,
    ngx_mail_pop3_init_protocol,
    ngx_mail_pop3_parse_command,
    ngx_mail_pop3_auth_state,

    ngx_string("-ERR internal server error" CRLF)
};


static ngx_command_t  ngx_mail_pop3_commands[] = {

    { ngx_string("pop3_capabilities"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_mail_capabilities,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_pop3_srv_conf_t, capabilities),
      NULL },

    { ngx_string("pop3_auth"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_pop3_srv_conf_t, auth_methods),
      &ngx_mail_pop3_auth_methods },

      ngx_null_command
};


static ngx_mail_module_t  ngx_mail_pop3_module_ctx = {
    &ngx_mail_pop3_protocol,               /* protocol */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_mail_pop3_create_srv_conf,         /* create server configuration */
    ngx_mail_pop3_merge_srv_conf           /* merge server configuration */
};


ngx_module_t  ngx_mail_pop3_module = {
    NGX_MODULE_V1,
    &ngx_mail_pop3_module_ctx,             /* module context */
    ngx_mail_pop3_commands,                /* module directives */
     NGX_MAIL_MODULE,                       /* module type */
     NULL,                                  /* init master */
     NULL,                                  /* init module */
     NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
 };


 static void *
 ngx_mail_pop3_create_srv_conf(ngx_conf_t *cf)
 {
```

```
    ngx_mail_pop3_srv_conf_t  *pscf;

    pscf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_pop3_srv_conf_t));
    if (pscf == NULL) {
        return NULL;
    }

    if (ngx_array_init(&pscf->capabilities, cf->pool, 4, sizeof(ngx_str_t))
        != NGX_OK)
    {
        return NULL;
    }

    return pscf;
}


static char *
ngx_mail_pop3_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_mail_pop3_srv_conf_t *prev = parent;
    ngx_mail_pop3_srv_conf_t *conf = child;

    u_char      *p;
    size_t       size, stls_only_size;
    ngx_str_t   *c, *d;
    ngx_uint_t   i;

    ngx_conf_merge_bitmask_value(conf->auth_methods,
                                 prev->auth_methods,
                                 (NGX_CONF_BITMASK_SET
                                  |NGX_MAIL_AUTH_PLAIN_ENABLED));

    if (conf->capabilities.nelts == 0) {
        conf->capabilities = prev->capabilities;
    }

    if (conf->capabilities.nelts == 0) {

        for (d = ngx_mail_pop3_default_capabilities; d->len; d++) {
            c = ngx_array_push(&conf->capabilities);
            if (c == NULL) {
                return NGX_CONF_ERROR;
            }

            *c = *d;
        }
    }

    size = sizeof("+OK Capability list follows" CRLF) - 1
           + sizeof("." CRLF) - 1;

    stls_only_size = size + sizeof("STLS" CRLF) - 1;

    c = conf->capabilities.elts;
    for (i = 0; i < conf->capabilities.nelts; i++) {
        size += c[i].len + sizeof(CRLF) - 1;

        if (ngx_strcasecmp(c[i].data, (u_char *) "USER") == 0) {
            continue;
        }

        stls_only_size += c[i].len + sizeof(CRLF) - 1;
    }

    if (conf->auth_methods & NGX_MAIL_AUTH_CRAM_MD5_ENABLED) {
        size += sizeof("SASL LOGIN PLAIN CRAM-MD5" CRLF) - 1;

    } else {
        size += sizeof("SASL LOGIN PLAIN" CRLF) - 1;
    }

    p = ngx_pnalloc(cf->pool, size);
    if (p == NULL) {
        return NGX_CONF_ERROR;
    }
```

```
191
192     conf->capability.len = size;
193     conf->capability.data = p;
194
195     p = ngx_cpymem(p, "+OK Capability list follows" CRLF,
196                    sizeof("+OK Capability list follows" CRLF) - 1);
197
198     for (i = 0; i < conf->capabilities.nelts; i++) {
199         p = ngx_cpymem(p, c[i].data, c[i].len);
200         *p++ = CR; *p++ = LF;
201     }
202
203     if (conf->auth_methods & NGX_MAIL_AUTH_CRAM_MD5_ENABLED) {
204         p = ngx_cpymem(p, "SASL LOGIN PLAIN CRAM-MD5" CRLF,
205                        sizeof("SASL LOGIN PLAIN CRAM-MD5" CRLF) - 1);
206
207     } else {
208         p = ngx_cpymem(p, "SASL LOGIN PLAIN" CRLF,
209                        sizeof("SASL LOGIN PLAIN" CRLF) - 1);
210     }
211
212     *p++ = '.'; *p++ = CR; *p = LF;
213
214
215     size += sizeof("STLS" CRLF) - 1;
216
217     p = ngx_pnalloc(cf->pool, size);
218     if (p == NULL) {
219         return NGX_CONF_ERROR;
220     }
221
222     conf->starttls_capability.len = size;
223     conf->starttls_capability.data = p;
224
225     p = ngx_cpymem(p, conf->capability.data,
226                    conf->capability.len - (sizeof("." CRLF) - 1));
227
228     p = ngx_cpymem(p, "STLS" CRLF, sizeof("STLS" CRLF) - 1);
229     *p++ = '.'; *p++ = CR; *p = LF;
230
231
232     if (conf->auth_methods & NGX_MAIL_AUTH_CRAM_MD5_ENABLED) {
233         conf->auth_capability = ngx_mail_pop3_auth_cram_md5_capability;
234
235     } else {
236         conf->auth_capability = ngx_mail_pop3_auth_plain_capability;
237     }
238
239
240     p = ngx_pnalloc(cf->pool, stls_only_size);
241     if (p == NULL) {
242         return NGX_CONF_ERROR;
243     }
244
245     conf->starttls_only_capability.len = stls_only_size;
246     conf->starttls_only_capability.data = p;
247
248     p = ngx_cpymem(p, "+OK Capability list follows" CRLF,
249                    sizeof("+OK Capability list follows" CRLF) - 1);
250
251     for (i = 0; i < conf->capabilities.nelts; i++) {
252         if (ngx_strcasecmp(c[i].data, (u_char *) "USER") == 0) {
253             continue;
254         }
255
256         p = ngx_cpymem(p, c[i].data, c[i].len);
257         *p++ = CR; *p++ = LF;
258     }
259
260     p = ngx_cpymem(p, "STLS" CRLF, sizeof("STLS" CRLF) - 1);
261     *p++ = '.'; *p++ = CR; *p = LF;
262
263     return NGX_CONF_OK;
264 }
```

One Level Up Top Level

src/mail/ngx_mail_pop3_handler.c - nginx-1.7.10

Global variables defined

- pop3_greeting
- pop3_invalid_command
- pop3_next
- pop3_ok
- pop3_password
- pop3_username

Functions defined

- ngx_mail_pop3_apop
- ngx_mail_pop3_auth
- ngx_mail_pop3_auth_state
- ngx_mail_pop3_capa
- ngx_mail_pop3_init_protocol
- ngx_mail_pop3_init_session
- ngx_mail_pop3_pass
- ngx_mail_pop3_stls
- ngx_mail_pop3_user

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_pop3_module.h>
13
14
15  static ngx_int_t ngx_mail_pop3_user(ngx_mail_session_t *s, ngx_connection_t *c);
16  static ngx_int_t ngx_mail_pop3_pass(ngx_mail_session_t *s, ngx_connection_t *c);
17  static ngx_int_t ngx_mail_pop3_capa(ngx_mail_session_t *s, ngx_connection_t *c,
18      ngx_int_t stls);
19  static ngx_int_t ngx_mail_pop3_stls(ngx_mail_session_t *s, ngx_connection_t *c);
20  static ngx_int_t ngx_mail_pop3_apop(ngx_mail_session_t *s, ngx_connection_t *c);
21  static ngx_int_t ngx_mail_pop3_auth(ngx_mail_session_t *s, ngx_connection_t *c);
22
23
24  static u_char  pop3_greeting[] = "+OK POP3 ready" CRLF;
25  static u_char  pop3_ok[] = "+OK" CRLF;
26  static u_char  pop3_next[] = "+ " CRLF;
27  static u_char  pop3_username[] = "+ VXNlcm5hbWU6" CRLF;
```

```
28  static u_char  pop3_password[] = "+ UGFzc3dvcmQ6" CRLF;
29  static u_char  pop3_invalid_command[] = "-ERR invalid command" CRLF;
30
31
32  void
33  ngx_mail_pop3_init_session(ngx_mail_session_t *s, ngx_connection_t *c)
34  {
35      u_char                     *p;
36      ngx_mail_core_srv_conf_t   *cscf;
37      ngx_mail_pop3_srv_conf_t   *pscf;
38
39      pscf = ngx_mail_get_module_srv_conf(s, ngx_mail_pop3_module);
40      cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);
41
42      if (pscf->auth_methods
43          & (NGX_MAIL_AUTH_APOP_ENABLED|NGX_MAIL_AUTH_CRAM_MD5_ENABLED))
44      {
45          if (ngx_mail_salt(s, c, cscf) != NGX_OK) {
46              ngx_mail_session_internal_server_error(s);
47              return;
48          }
49
50          s->out.data = ngx_pnalloc(c->pool, sizeof(pop3_greeting) + s->salt.len);
51          if (s->out.data == NULL) {
52              ngx_mail_session_internal_server_error(s);
53              return;
54          }
55
56          p = ngx_cpymem(s->out.data, pop3_greeting, sizeof(pop3_greeting) - 3);
57          *p++ = ' ';
58          p = ngx_cpymem(p, s->salt.data, s->salt.len);
59
60          s->out.len = p - s->out.data;
61
62      } else {
63          ngx_str_set(&s->out, pop3_greeting);
64      }
65
66      c->read->handler = ngx_mail_pop3_init_protocol;
67
68      ngx_add_timer(c->read, cscf->timeout);
69
70      if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
71          ngx_mail_close_connection(c);
72      }
73
74      ngx_mail_send(c->write);
75  }
76
77
78  void
79  ngx_mail_pop3_init_protocol(ngx_event_t *rev)
80  {
81      ngx_connection_t    *c;
82      ngx_mail_session_t  *s;
83
84      c = rev->data;
85
86      c->log->action = "in auth state";
87
88      if (rev->timedout) {
89          ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
90          c->timedout = 1;
91          ngx_mail_close_connection(c);
92          return;
93      }
94
95      s = c->data;
96
97      if (s->buffer == NULL) {
98          if (ngx_array_init(&s->args, c->pool, 2, sizeof(ngx_str_t))
99              == NGX_ERROR)
100         {
101             ngx_mail_session_internal_server_error(s);
102             return;
103         }
```

```
104
105         s->buffer = ngx_create_temp_buf(c->pool, 128);
106         if (s->buffer == NULL) {
107             ngx_mail_session_internal_server_error(s);
108             return;
109         }
110     }
111
112     s->mail_state = ngx_pop3_start;
113     c->read->handler = ngx_mail_pop3_auth_state;
114
115     ngx_mail_pop3_auth_state(rev);
116 }
117
118
119 void
120 ngx_mail_pop3_auth_state(ngx_event_t *rev)
121 {
122     ngx_int_t             rc;
123     ngx_connection_t     *c;
124     ngx_mail_session_t   *s;
125
126     c = rev->data;
127     s = c->data;
128
129     ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "pop3 auth state");
130
131     if (rev->timedout) {
132         ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
133         c->timedout = 1;
134         ngx_mail_close_connection(c);
135         return;
136     }
137
138     if (s->out.len) {
139         ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "pop3 send handler busy");
140         s->blocked = 1;
141         return;
142     }
143
144     s->blocked = 0;
145
146     rc = ngx_mail_read_command(s, c);
147
148     if (rc == NGX_AGAIN || rc == NGX_ERROR) {
149         return;
150     }
151
152     ngx_str_set(&s->out, pop3_ok);
153
154     if (rc == NGX_OK) {
155         switch (s->mail_state) {
156
157         case ngx_pop3_start:
158
159             switch (s->command) {
160
161             case NGX_POP3_USER:
162                 rc = ngx_mail_pop3_user(s, c);
163                 break;
164
165             case NGX_POP3_CAPA:
166                 rc = ngx_mail_pop3_capa(s, c, 1);
167                 break;
168
169             case NGX_POP3_APOP:
170                 rc = ngx_mail_pop3_apop(s, c);
171                 break;
172
173             case NGX_POP3_AUTH:
174                 rc = ngx_mail_pop3_auth(s, c);
175                 break;
176
177             case NGX_POP3_QUIT:
178                 s->quit = 1;
179                 break;
```

```
180
181             case NGX_POP3_NOOP:
182                 break;
183
184             case NGX_POP3_STLS:
185                 rc = ngx_mail_pop3_stls(s, c);
186                 break;
187
188             default:
189                 rc = NGX_MAIL_PARSE_INVALID_COMMAND;
190                 break;
191             }
192
193             break;
194
195         case ngx_pop3_user:
196
197             switch (s->command) {
198
199             case NGX_POP3_PASS:
200                 rc = ngx_mail_pop3_pass(s, c);
201                 break;
202
203             case NGX_POP3_CAPA:
204                 rc = ngx_mail_pop3_capa(s, c, 0);
205                 break;
206
207             case NGX_POP3_QUIT:
208                 s->quit = 1;
209                 break;
210
211             case NGX_POP3_NOOP:
212                 break;
213
214             default:
215                 rc = NGX_MAIL_PARSE_INVALID_COMMAND;
216                 break;
217             }
218
219             break;
220
221         /* suppress warnings */
222         case ngx_pop3_passwd:
223             break;
224
225         case ngx_pop3_auth_login_username:
226             rc = ngx_mail_auth_login_username(s, c, 0);
227
228             ngx_str_set(&s->out, pop3_password);
229             s->mail_state = ngx_pop3_auth_login_password;
230             break;
231
232         case ngx_pop3_auth_login_password:
233             rc = ngx_mail_auth_login_password(s, c);
234             break;
235
236         case ngx_pop3_auth_plain:
237             rc = ngx_mail_auth_plain(s, c, 0);
238             break;
239
240         case ngx_pop3_auth_cram_md5:
241             rc = ngx_mail_auth_cram_md5(s, c);
242             break;
243         }
244     }
245
246     switch (rc) {
247
248     case NGX_DONE:
249         ngx_mail_auth(s, c);
250         return;
251
252     case NGX_ERROR:
253         ngx_mail_session_internal_server_error(s);
254         return;
255
```

```
    case NGX_MAIL_PARSE_INVALID_COMMAND:
        s->mail_state = ngx_pop3_start;
        s->state = 0;

        ngx_str_set(&s->out, pop3_invalid_command);

        /* fall through */

    case NGX_OK:

        s->args.nelts = 0;
        s->buffer->pos = s->buffer->start;
        s->buffer->last = s->buffer->start;

        if (s->state) {
            s->arg_start = s->buffer->start;
        }

        ngx_mail_send(c->write);
    }
}

static ngx_int_t
ngx_mail_pop3_user(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t  *arg;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif

    if (s->args.nelts != 1) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    arg = s->args.elts;
    s->login.len = arg[0].len;
    s->login.data = ngx_pnalloc(c->pool, s->login.len);
    if (s->login.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->login.data, arg[0].data, s->login.len);

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "pop3 login: \"%V\"", &s->login);

    s->mail_state = ngx_pop3_user;

    return NGX_OK;
}


static ngx_int_t
ngx_mail_pop3_pass(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t  *arg;

    if (s->args.nelts != 1) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    arg = s->args.elts;
    s->passwd.len = arg[0].len;
    s->passwd.data = ngx_pnalloc(c->pool, s->passwd.len);
    if (s->passwd.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->passwd.data, arg[0].data, s->passwd.len);

#if (NGX_DEBUG_MAIL_PASSWD)
    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "pop3 passwd: \"%V\"", &s->passwd);
```

```
#endif

    return NGX_DONE;
}


static ngx_int_t
ngx_mail_pop3_capa(ngx_mail_session_t *s, ngx_connection_t *c, ngx_int_t stls)
{
    ngx_mail_pop3_srv_conf_t  *pscf;

    pscf = ngx_mail_get_module_srv_conf(s, ngx_mail_pop3_module);

#if (NGX_MAIL_SSL)

    if (stls && c->ssl == NULL) {
        ngx_mail_ssl_conf_t  *sslcf;

        sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);

        if (sslcf->starttls == NGX_MAIL_STARTTLS_ON) {
            s->out = pscf->starttls_capability;
            return NGX_OK;
        }

        if (sslcf->starttls == NGX_MAIL_STARTTLS_ONLY) {
            s->out = pscf->starttls_only_capability;
            return NGX_OK;
        }
    }

#endif

    s->out = pscf->capability;
    return NGX_OK;
}


static ngx_int_t
ngx_mail_pop3_stls(ngx_mail_session_t *s, ngx_connection_t *c)
{
#if (NGX_MAIL_SSL)
    ngx_mail_ssl_conf_t  *sslcf;

    if (c->ssl == NULL) {
        sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);
        if (sslcf->starttls) {
            c->read->handler = ngx_mail_starttls_handler;
            return NGX_OK;
        }
    }

#endif

    return NGX_MAIL_PARSE_INVALID_COMMAND;
}


static ngx_int_t
ngx_mail_pop3_apop(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t                 *arg;
    ngx_mail_pop3_srv_conf_t  *pscf;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif

    if (s->args.nelts != 2) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    pscf = ngx_mail_get_module_srv_conf(s, ngx_mail_pop3_module);

```

```
    if (!(pscf->auth_methods & NGX_MAIL_AUTH_APOP_ENABLED)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    arg = s->args.elts;

    s->login.len = arg[0].len;
    s->login.data = ngx_pnalloc(c->pool, s->login.len);
    if (s->login.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->login.data, arg[0].data, s->login.len);

    s->passwd.len = arg[1].len;
    s->passwd.data = ngx_pnalloc(c->pool, s->passwd.len);
    if (s->passwd.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->passwd.data, arg[1].data, s->passwd.len);

    ngx_log_debug2(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "pop3 apop: \"%V\" \"%V\"", &s->login, &s->passwd);

    s->auth_method = NGX_MAIL_AUTH_APOP;

    return NGX_DONE;
}


static ngx_int_t
ngx_mail_pop3_auth(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_int_t                  rc;
    ngx_mail_pop3_srv_conf_t  *pscf;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif

    pscf = ngx_mail_get_module_srv_conf(s, ngx_mail_pop3_module);

    if (s->args.nelts == 0) {
        s->out = pscf->auth_capability;
        s->state = 0;

        return NGX_OK;
    }

    rc = ngx_mail_auth_parse(s, c);

    switch (rc) {

    case NGX_MAIL_AUTH_LOGIN:

        ngx_str_set(&s->out, pop3_username);
        s->mail_state = ngx_pop3_auth_login_username;

        return NGX_OK;

    case NGX_MAIL_AUTH_LOGIN_USERNAME:

        ngx_str_set(&s->out, pop3_password);
        s->mail_state = ngx_pop3_auth_login_password;

        return ngx_mail_auth_login_username(s, c, 1);

    case NGX_MAIL_AUTH_PLAIN:

        ngx_str_set(&s->out, pop3_next);
        s->mail_state = ngx_pop3_auth_plain;

        return NGX_OK;
```

```
484
485     case NGX_MAIL_AUTH_CRAM_MD5:
486
487         if (!(pscf->auth_methods & NGX_MAIL_AUTH_CRAM_MD5_ENABLED)) {
488             return NGX_MAIL_PARSE_INVALID_COMMAND;
489         }
490
491         if (ngx_mail_auth_cram_md5_salt(s, c, "+ ", 2) == NGX_OK) {
492             s->mail_state = ngx_pop3_auth_cram_md5;
493             return NGX_OK;
494         }
495
496         return NGX_ERROR;
497     }
498
499     return rc;
500 }
```

One Level Up Top Level

src/mail/ngx_mail_pop3_module.h - nginx-1.7.10

Data types defined

- ngx_mail_pop3_srv_conf_t

Macros defined

- _NGX_MAIL_POP3_MODULE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_MAIL_POP3_MODULE_H_INCLUDED_
9  #define _NGX_MAIL_POP3_MODULE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_mail.h>
15
16
17  typedef struct {
18      ngx_str_t    capability;
19      ngx_str_t    starttls_capability;
20      ngx_str_t    starttls_only_capability;
21      ngx_str_t    auth_capability;
22
23      ngx_uint_t   auth_methods;
24
25      ngx_array_t  capabilities;
26  } ngx_mail_pop3_srv_conf_t;
27
28
29  void ngx_mail_pop3_init_session(ngx_mail_session_t *s, ngx_connection_t *c);
30  void ngx_mail_pop3_init_protocol(ngx_event_t *rev);
31  void ngx_mail_pop3_auth_state(ngx_event_t *rev);
32  ngx_int_t ngx_mail_pop3_parse_command(ngx_mail_session_t *s);
33
34
35  extern ngx_module_t  ngx_mail_pop3_module;
36
37
38  #endif /* _NGX_MAIL_POP3_MODULE_H_INCLUDED_ */
```

src/mail/ngx_mail_imap_module.c - nginx-1.7.10

Global variables defined

- ngx_mail_imap_auth_methods
- ngx_mail_imap_auth_methods_names
- ngx_mail_imap_commands
- ngx_mail_imap_default_capabilities
- ngx_mail_imap_module
- ngx_mail_imap_module_ctx
- ngx_mail_imap_protocol

Functions defined

- ngx_mail_imap_create_srv_conf
- ngx_mail_imap_merge_srv_conf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_imap_module.h>
13
14
15  static void *ngx_mail_imap_create_srv_conf(ngx_conf_t *cf);
16  static char *ngx_mail_imap_merge_srv_conf(ngx_conf_t *cf, void *parent,
17      void *child);
18
19
20  static ngx_str_t  ngx_mail_imap_default_capabilities[] = {
21      ngx_string("IMAP4"),
22      ngx_string("IMAP4rev1"),
23      ngx_string("UIDPLUS"),
24      ngx_null_string
25  };
26
27
28  static ngx_conf_bitmask_t  ngx_mail_imap_auth_methods[] = {
29      { ngx_string("plain"), NGX_MAIL_AUTH_PLAIN_ENABLED },
30      { ngx_string("login"), NGX_MAIL_AUTH_LOGIN_ENABLED },
31      { ngx_string("cram-md5"), NGX_MAIL_AUTH_CRAM_MD5_ENABLED },
32      { ngx_null_string, 0 }
33  };
34
35
36  static ngx_str_t  ngx_mail_imap_auth_methods_names[] = {
37      ngx_string("AUTH=PLAIN"),
38      ngx_string("AUTH=LOGIN"),
39      ngx_null_string,  /* APOP */
40      ngx_string("AUTH=CRAM-MD5"),
41      ngx_null_string   /* NONE */
```

```
};


static ngx_mail_protocol_t  ngx_mail_imap_protocol = {
    ngx_string("imap"),
    { 143, 993, 0, 0 },
    NGX_MAIL_IMAP_PROTOCOL,

    ngx_mail_imap_init_session,
    ngx_mail_imap_init_protocol,
    ngx_mail_imap_parse_command,
    ngx_mail_imap_auth_state,

    ngx_string("* BAD internal server error" CRLF)
};


static ngx_command_t  ngx_mail_imap_commands[] = {

    { ngx_string("imap_client_buffer"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_imap_srv_conf_t, client_buffer_size),
      NULL },

    { ngx_string("imap_capabilities"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_mail_capabilities,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_imap_srv_conf_t, capabilities),
      NULL },

    { ngx_string("imap_auth"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_imap_srv_conf_t, auth_methods),
      &ngx_mail_imap_auth_methods },

      ngx_null_command
};


static ngx_mail_module_t  ngx_mail_imap_module_ctx = {
    &ngx_mail_imap_protocol,               /* protocol */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_mail_imap_create_srv_conf,         /* create server configuration */
    ngx_mail_imap_merge_srv_conf           /* merge server configuration */
};


ngx_module_t  ngx_mail_imap_module = {
    NGX_MODULE_V1,
    &ngx_mail_imap_module_ctx,             /* module context */
     ngx_mail_imap_commands,                /* module directives */
     NGX_MAIL_MODULE,                       /* module type */
     NULL,                                  /* init master */
     NULL,                                  /* init module */
     NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
 };


 static void *
 ngx_mail_imap_create_srv_conf(ngx_conf_t *cf)
 {
     ngx_mail_imap_srv_conf_t  *iscf;

```

```
    iscf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_imap_srv_conf_t));
    if (iscf == NULL) {
        return NULL;
    }

    iscf->client_buffer_size = NGX_CONF_UNSET_SIZE;

    if (ngx_array_init(&iscf->capabilities, cf->pool, 4, sizeof(ngx_str_t))
        != NGX_OK)
    {
        return NULL;
    }

    return iscf;
}


static char *
ngx_mail_imap_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_mail_imap_srv_conf_t *prev = parent;
    ngx_mail_imap_srv_conf_t *conf = child;

    u_char      *p, *auth;
    size_t       size;
    ngx_str_t   *c, *d;
    ngx_uint_t   i, m;

    ngx_conf_merge_size_value(conf->client_buffer_size,
                              prev->client_buffer_size,
                              (size_t) ngx_pagesize);

    ngx_conf_merge_bitmask_value(conf->auth_methods,
                              prev->auth_methods,
                              (NGX_CONF_BITMASK_SET
                               |NGX_MAIL_AUTH_PLAIN_ENABLED));


    if (conf->capabilities.nelts == 0) {
        conf->capabilities = prev->capabilities;
    }

    if (conf->capabilities.nelts == 0) {

        for (d = ngx_mail_imap_default_capabilities; d->len; d++) {
            c = ngx_array_push(&conf->capabilities);
            if (c == NULL) {
                return NGX_CONF_ERROR;
            }

            *c = *d;
        }
    }

    size = sizeof("* CAPABILITY" CRLF) - 1;

    c = conf->capabilities.elts;
    for (i = 0; i < conf->capabilities.nelts; i++) {
        size += 1 + c[i].len;
    }

    for (m = NGX_MAIL_AUTH_PLAIN_ENABLED, i = 0;
         m <= NGX_MAIL_AUTH_CRAM_MD5_ENABLED;
         m <<= 1, i++)
    {
        if (m & conf->auth_methods) {
            size += 1 + ngx_mail_imap_auth_methods_names[i].len;
        }
    }

    p = ngx_pnalloc(cf->pool, size);
    if (p == NULL) {
        return NGX_CONF_ERROR;
    }

    conf->capability.len = size;
```

```
194     conf->capability.data = p;
195
196     p = ngx_cpymem(p, "* CAPABILITY", sizeof("* CAPABILITY") - 1);
197
198     for (i = 0; i < conf->capabilities.nelts; i++) {
199         *p++ = ' ';
200         p = ngx_cpymem(p, c[i].data, c[i].len);
201     }
202
203     auth = p;
204
205     for (m = NGX_MAIL_AUTH_PLAIN_ENABLED, i = 0;
206          m <= NGX_MAIL_AUTH_CRAM_MD5_ENABLED;
207          m <<= 1, i++)
208     {
209         if (m & conf->auth_methods) {
210             *p++ = ' ';
211             p = ngx_cpymem(p, ngx_mail_imap_auth_methods_names[i].data,
212                            ngx_mail_imap_auth_methods_names[i].len);
213         }
214     }
215
216     *p++ = CR; *p = LF;
217
218
219     size += sizeof(" STARTTLS") - 1;
220
221     p = ngx_pnalloc(cf->pool, size);
222     if (p == NULL) {
223         return NGX_CONF_ERROR;
224     }
225
226     conf->starttls_capability.len = size;
227     conf->starttls_capability.data = p;
228
229     p = ngx_cpymem(p, conf->capability.data,
230                    conf->capability.len - (sizeof(CRLF) - 1));
231     p = ngx_cpymem(p, " STARTTLS", sizeof(" STARTTLS") - 1);
232     *p++ = CR; *p = LF;
233
234
235     size = (auth - conf->capability.data) + sizeof(CRLF) - 1
236             + sizeof(" STARTTLS LOGINDISABLED") - 1;
237
238     p = ngx_pnalloc(cf->pool, size);
239     if (p == NULL) {
240         return NGX_CONF_ERROR;
241     }
242
243     conf->starttls_only_capability.len = size;
244     conf->starttls_only_capability.data = p;
245
246     p = ngx_cpymem(p, conf->capability.data,
247                    auth - conf->capability.data);
248     p = ngx_cpymem(p, " STARTTLS LOGINDISABLED",
249                    sizeof(" STARTTLS LOGINDISABLED") - 1);
250     *p++ = CR; *p = LF;
251
252     return NGX_CONF_OK;
253 }
```

One Level Up Top Level

src/mail/ngx_mail_imap_handler.c - nginx-1.7.10

Global variables defined

- imap_bye
- imap_greeting
- imap_invalid_command
- imap_next
- imap_ok
- imap_password
- imap_plain_next
- imap_star
- imap_username

Functions defined

- ngx_mail_imap_auth_state
- ngx_mail_imap_authenticate
- ngx_mail_imap_capability
- ngx_mail_imap_init_protocol
- ngx_mail_imap_init_session
- ngx_mail_imap_login
- ngx_mail_imap_starttls

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_imap_module.h>
13
14
15  static ngx_int_t ngx_mail_imap_login(ngx_mail_session_t *s,
16      ngx_connection_t *c);
17  static ngx_int_t ngx_mail_imap_authenticate(ngx_mail_session_t *s,
18      ngx_connection_t *c);
19  static ngx_int_t ngx_mail_imap_capability(ngx_mail_session_t *s,
20      ngx_connection_t *c);
21  static ngx_int_t ngx_mail_imap_starttls(ngx_mail_session_t *s,
22      ngx_connection_t *c);
23
24
25  static u_char  imap_greeting[] = "* OK IMAP4 ready" CRLF;
```

```
26  static u_char  imap_star[] = "* ";
27  static u_char  imap_ok[] = "OK completed" CRLF;
28  static u_char  imap_next[] = "+ OK" CRLF;
29  static u_char  imap_plain_next[] = "+ " CRLF;
30  static u_char  imap_username[] = "+ VXNlcm5hbWU6" CRLF;
31  static u_char  imap_password[] = "+ UGFzc3dvcmQ6" CRLF;
32  static u_char  imap_bye[] = "* BYE" CRLF;
33  static u_char  imap_invalid_command[] = "BAD invalid command" CRLF;
34
35
36  void
37  ngx_mail_imap_init_session(ngx_mail_session_t *s, ngx_connection_t *c)
38  {
39      ngx_mail_core_srv_conf_t  *cscf;
40
41      cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);
42
43      ngx_str_set(&s->out, imap_greeting);
44
45      c->read->handler = ngx_mail_imap_init_protocol;
46
47      ngx_add_timer(c->read, cscf->timeout);
48
49      if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
50          ngx_mail_close_connection(c);
51      }
52
53      ngx_mail_send(c->write);
54  }
55
56
57  void
58  ngx_mail_imap_init_protocol(ngx_event_t *rev)
59  {
60      ngx_connection_t          *c;
61      ngx_mail_session_t        *s;
62      ngx_mail_imap_srv_conf_t  *iscf;
63
64      c = rev->data;
65
66      c->log->action = "in auth state";
67
68      if (rev->timedout) {
69          ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
70          c->timedout = 1;
71          ngx_mail_close_connection(c);
72          return;
73      }
74
75      s = c->data;
76
77      if (s->buffer == NULL) {
78          if (ngx_array_init(&s->args, c->pool, 2, sizeof(ngx_str_t))
79              == NGX_ERROR)
80          {
81              ngx_mail_session_internal_server_error(s);
82              return;
83          }
84
85          iscf = ngx_mail_get_module_srv_conf(s, ngx_mail_imap_module);
86
87          s->buffer = ngx_create_temp_buf(c->pool, iscf->client_buffer_size);
88          if (s->buffer == NULL) {
89              ngx_mail_session_internal_server_error(s);
90              return;
91          }
92      }
93
94      s->mail_state = ngx_imap_start;
95      c->read->handler = ngx_mail_imap_auth_state;
96
97      ngx_mail_imap_auth_state(rev);
98  }
99
100
101  void
```

```
ngx_mail_imap_auth_state(ngx_event_t *rev)
{
    u_char              *p, *dst, *src, *end;
    ngx_str_t           *arg;
    ngx_int_t            rc;
    ngx_uint_t           tag, i;
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    c = rev->data;
    s = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "imap auth state");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        c->timedout = 1;
        ngx_mail_close_connection(c);
        return;
    }

    if (s->out.len) {
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "imap send handler busy");
        s->blocked = 1;
        return;
    }

    s->blocked = 0;

    rc = ngx_mail_read_command(s, c);

    if (rc == NGX_AGAIN || rc == NGX_ERROR) {
        return;
    }

    tag = 1;
    s->text.len = 0;
    ngx_str_set(&s->out, imap_ok);

    if (rc == NGX_OK) {

        ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0, "imap auth command: %i",
                       s->command);

        if (s->backslash) {

            arg = s->args.elts;

            for (i = 0; i < s->args.nelts; i++) {
                dst = arg[i].data;
                end = dst + arg[i].len;

                for (src = dst; src < end; dst++) {
                    *dst = *src;
                    if (*src++ == '\\') {
                        *dst = *src++;
                    }
                }

                arg[i].len = dst - arg[i].data;
            }

            s->backslash = 0;
        }

        switch (s->mail_state) {

        case ngx_imap_start:

            switch (s->command) {

            case NGX_IMAP_LOGIN:
                rc = ngx_mail_imap_login(s, c);
                break;

            case NGX_IMAP_AUTHENTICATE:
```

```
178                 rc = ngx_mail_imap_authenticate(s, c);
179                 tag = (rc != NGX_OK);
180                 break;
181
182             case NGX_IMAP_CAPABILITY:
183                 rc = ngx_mail_imap_capability(s, c);
184                 break;
185
186             case NGX_IMAP_LOGOUT:
187                 s->quit = 1;
188                 ngx_str_set(&s->text, imap_bye);
189                 break;
190
191             case NGX_IMAP_NOOP:
192                 break;
193
194             case NGX_IMAP_STARTTLS:
195                 rc = ngx_mail_imap_starttls(s, c);
196                 break;
197
198             default:
199                 rc = NGX_MAIL_PARSE_INVALID_COMMAND;
200                 break;
201             }
202
203             break;
204
205         case ngx_imap_auth_login_username:
206             rc = ngx_mail_auth_login_username(s, c, 0);
207
208             tag = 0;
209             ngx_str_set(&s->out, imap_password);
210             s->mail_state = ngx_imap_auth_login_password;
211
212             break;
213
214         case ngx_imap_auth_login_password:
215             rc = ngx_mail_auth_login_password(s, c);
216             break;
217
218         case ngx_imap_auth_plain:
219             rc = ngx_mail_auth_plain(s, c, 0);
220             break;
221
222         case ngx_imap_auth_cram_md5:
223             rc = ngx_mail_auth_cram_md5(s, c);
224             break;
225         }
226
227     } else if (rc == NGX_IMAP_NEXT) {
228         tag = 0;
229         ngx_str_set(&s->out, imap_next);
230     }
231
232     switch (rc) {
233
234     case NGX_DONE:
235         ngx_mail_auth(s, c);
236         return;
237
238     case NGX_ERROR:
239         ngx_mail_session_internal_server_error(s);
240         return;
241
242     case NGX_MAIL_PARSE_INVALID_COMMAND:
243         s->state = 0;
244         ngx_str_set(&s->out, imap_invalid_command);
245         s->mail_state = ngx_imap_start;
246         break;
247     }
248
249     if (tag) {
250         if (s->tag.len == 0) {
251             ngx_str_set(&s->tag, imap_star);
252         }
253
```

```
        if (s->tagged_line.len < s->tag.len + s->text.len + s->out.len) {
            s->tagged_line.len = s->tag.len + s->text.len + s->out.len;
            s->tagged_line.data = ngx_pnalloc(c->pool, s->tagged_line.len);
            if (s->tagged_line.data == NULL) {
                ngx_mail_close_connection(c);
                return;
            }
        }

        p = s->tagged_line.data;

        if (s->text.len) {
            p = ngx_cpymem(p, s->text.data, s->text.len);
        }

        p = ngx_cpymem(p, s->tag.data, s->tag.len);
        ngx_memcpy(p, s->out.data, s->out.len);

        s->out.len = s->text.len + s->tag.len + s->out.len;
        s->out.data = s->tagged_line.data;
    }

    if (rc != NGX_IMAP_NEXT) {
        s->args.nelts = 0;

        if (s->state) {
            /* preserve tag */
            s->arg_start = s->buffer->start + s->tag.len;
            s->buffer->pos = s->arg_start;
            s->buffer->last = s->arg_start;

        } else {
            s->buffer->pos = s->buffer->start;
            s->buffer->last = s->buffer->start;
            s->tag.len = 0;
        }
    }

    ngx_mail_send(c->write);
}


static ngx_int_t
ngx_mail_imap_login(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t  *arg;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif

    arg = s->args.elts;

    if (s->args.nelts != 2 || arg[0].len == 0) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }

    s->login.len = arg[0].len;
    s->login.data = ngx_pnalloc(c->pool, s->login.len);
    if (s->login.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->login.data, arg[0].data, s->login.len);

    s->passwd.len = arg[1].len;
    s->passwd.data = ngx_pnalloc(c->pool, s->passwd.len);
    if (s->passwd.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->passwd.data, arg[1].data, s->passwd.len);

#if (NGX_DEBUG_MAIL_PASSWD)
```

```
    ngx_log_debug2(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "imap login:\"%V\" passwd:\"%V\"",
                   &s->login, &s->passwd);
#else
    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "imap login:\"%V\"", &s->login);
#endif

    return NGX_DONE;
}


static ngx_int_t
ngx_mail_imap_authenticate(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_int_t                  rc;
    ngx_mail_core_srv_conf_t  *cscf;
    ngx_mail_imap_srv_conf_t  *iscf;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif

    rc = ngx_mail_auth_parse(s, c);

    switch (rc) {

    case NGX_MAIL_AUTH_LOGIN:

        ngx_str_set(&s->out, imap_username);
        s->mail_state = ngx_imap_auth_login_username;

        return NGX_OK;

    case NGX_MAIL_AUTH_LOGIN_USERNAME:

        ngx_str_set(&s->out, imap_password);
        s->mail_state = ngx_imap_auth_login_password;

        return ngx_mail_auth_login_username(s, c, 1);

    case NGX_MAIL_AUTH_PLAIN:

        ngx_str_set(&s->out, imap_plain_next);
        s->mail_state = ngx_imap_auth_plain;

        return NGX_OK;

    case NGX_MAIL_AUTH_CRAM_MD5:

        iscf = ngx_mail_get_module_srv_conf(s, ngx_mail_imap_module);

        if (!(iscf->auth_methods & NGX_MAIL_AUTH_CRAM_MD5_ENABLED)) {
            return NGX_MAIL_PARSE_INVALID_COMMAND;
        }

        if (s->salt.data == NULL) {
            cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

            if (ngx_mail_salt(s, c, cscf) != NGX_OK) {
                return NGX_ERROR;
            }
        }

        if (ngx_mail_auth_cram_md5_salt(s, c, "+ ", 2) == NGX_OK) {
            s->mail_state = ngx_imap_auth_cram_md5;
            return NGX_OK;
        }

        return NGX_ERROR;
    }

    return rc;
}
```

```
406
407
408 static ngx_int_t
409 ngx_mail_imap_capability(ngx_mail_session_t *s, ngx_connection_t *c)
410 {
411     ngx_mail_imap_srv_conf_t  *iscf;
412
413     iscf = ngx_mail_get_module_srv_conf(s, ngx_mail_imap_module);
414
415 #if (NGX_MAIL_SSL)
416
417     if (c->ssl == NULL) {
418         ngx_mail_ssl_conf_t  *sslcf;
419
420         sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);
421
422         if (sslcf->starttls == NGX_MAIL_STARTTLS_ON) {
423             s->text = iscf->starttls_capability;
424             return NGX_OK;
425         }
426
427         if (sslcf->starttls == NGX_MAIL_STARTTLS_ONLY) {
428             s->text = iscf->starttls_only_capability;
429             return NGX_OK;
430         }
431     }
432 #endif
433
434     s->text = iscf->capability;
435
436     return NGX_OK;
437 }
438
439
440 static ngx_int_t
441 ngx_mail_imap_starttls(ngx_mail_session_t *s, ngx_connection_t *c)
442 {
443 #if (NGX_MAIL_SSL)
444     ngx_mail_ssl_conf_t  *sslcf;
445
446     if (c->ssl == NULL) {
447         sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);
448         if (sslcf->starttls) {
449             c->read->handler = ngx_mail_starttls_handler;
450             return NGX_OK;
451         }
452     }
453
454 #endif
455
456     return NGX_MAIL_PARSE_INVALID_COMMAND;
457 }
```

One Level Up Top Level

src/mail/ngx_mail_imap_module.h - nginx-1.7.10

Data types defined

- ngx_mail_imap_srv_conf_t

Macros defined

- _NGX_MAIL_IMAP_MODULE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_MAIL_IMAP_MODULE_H_INCLUDED_
9  #define _NGX_MAIL_IMAP_MODULE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_mail.h>
15
16
17  typedef struct {
18      size_t       client_buffer_size;
19
20      ngx_str_t    capability;
21      ngx_str_t    starttls_capability;
22      ngx_str_t    starttls_only_capability;
23
24      ngx_uint_t   auth_methods;
25
26      ngx_array_t  capabilities;
27  } ngx_mail_imap_srv_conf_t;
28
29
30  void ngx_mail_imap_init_session(ngx_mail_session_t *s, ngx_connection_t *c);
31  void ngx_mail_imap_init_protocol(ngx_event_t *rev);
32  void ngx_mail_imap_auth_state(ngx_event_t *rev);
33  ngx_int_t ngx_mail_imap_parse_command(ngx_mail_session_t *s);
34
35
36  extern ngx_module_t  ngx_mail_imap_module;
37
38
39  #endif /* _NGX_MAIL_IMAP_MODULE_H_INCLUDED_ */
```

src/mail/ngx_mail_smtp_module.c - nginx-1.7.10

Global variables defined

- ngx_mail_smtp_auth_methods
- ngx_mail_smtp_auth_methods_names
- ngx_mail_smtp_commands
- ngx_mail_smtp_module
- ngx_mail_smtp_module_ctx
- ngx_mail_smtp_protocol

Functions defined

- ngx_mail_smtp_create_srv_conf
- ngx_mail_smtp_merge_srv_conf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_smtp_module.h>
13
14
15  static void *ngx_mail_smtp_create_srv_conf(ngx_conf_t *cf);
16  static char *ngx_mail_smtp_merge_srv_conf(ngx_conf_t *cf, void *parent,
17      void *child);
18
19
20  static ngx_conf_bitmask_t  ngx_mail_smtp_auth_methods[] = {
21      { ngx_string("plain"), NGX_MAIL_AUTH_PLAIN_ENABLED },
22      { ngx_string("login"), NGX_MAIL_AUTH_LOGIN_ENABLED },
23      { ngx_string("cram-md5"), NGX_MAIL_AUTH_CRAM_MD5_ENABLED },
24      { ngx_string("none"), NGX_MAIL_AUTH_NONE_ENABLED },
25      { ngx_null_string, 0 }
26  };
27
28
29  static ngx_str_t  ngx_mail_smtp_auth_methods_names[] = {
30      ngx_string("PLAIN"),
31      ngx_string("LOGIN"),
32      ngx_null_string,  /* APOP */
33      ngx_string("CRAM-MD5"),
34      ngx_null_string   /* NONE */
35  };
36
37
38  static ngx_mail_protocol_t  ngx_mail_smtp_protocol = {
39      ngx_string("smtp"),
40      { 25, 465, 587, 0 },
41      NGX_MAIL_SMTP_PROTOCOL,
42
43      ngx_mail_smtp_init_session,
```

```
    ngx_mail_smtp_init_protocol,
    ngx_mail_smtp_parse_command,
    ngx_mail_smtp_auth_state,

    ngx_string("451 4.3.2 Internal server error" CRLF)
};


static ngx_command_t  ngx_mail_smtp_commands[] = {

    { ngx_string("smtp_client_buffer"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_smtp_srv_conf_t, client_buffer_size),
      NULL },

    { ngx_string("smtp_greeting_delay"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_smtp_srv_conf_t, greeting_delay),
      NULL },

    { ngx_string("smtp_capabilities"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_mail_capabilities,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_smtp_srv_conf_t, capabilities),
      NULL },

    { ngx_string("smtp_auth"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_smtp_srv_conf_t, auth_methods),
      &ngx_mail_smtp_auth_methods },

      ngx_null_command
};


static ngx_mail_module_t  ngx_mail_smtp_module_ctx = {
    &ngx_mail_smtp_protocol,               /* protocol */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_mail_smtp_create_srv_conf,         /* create server configuration */
    ngx_mail_smtp_merge_srv_conf           /* merge server configuration */
};


ngx_module_t  ngx_mail_smtp_module = {
    NGX_MODULE_V1,
    &ngx_mail_smtp_module_ctx,             /* module context */
     ngx_mail_smtp_commands,                /* module directives */
     NGX_MAIL_MODULE,                       /* module type */
     NULL,                                  /* init master */
     NULL,                                  /* init module */
     NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
 };


 static void *
 ngx_mail_smtp_create_srv_conf(ngx_conf_t *cf)
 {
     ngx_mail_smtp_srv_conf_t  *sscf;

     sscf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_smtp_srv_conf_t));
     if (sscf == NULL) {
```

```
        return NULL;
    }

    sscf->client_buffer_size = NGX_CONF_UNSET_SIZE;
    sscf->greeting_delay = NGX_CONF_UNSET_MSEC;

    if (ngx_array_init(&sscf->capabilities, cf->pool, 4, sizeof(ngx_str_t))
        != NGX_OK)
    {
        return NULL;
    }

    return sscf;
}


static char *
ngx_mail_smtp_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_mail_smtp_srv_conf_t *prev = parent;
    ngx_mail_smtp_srv_conf_t *conf = child;

    u_char                    *p, *auth, *last;
    size_t                     size;
    ngx_str_t                 *c;
    ngx_uint_t                 i, m, auth_enabled;
    ngx_mail_core_srv_conf_t  *cscf;

    ngx_conf_merge_size_value(conf->client_buffer_size,
                              prev->client_buffer_size,
                              (size_t) ngx_pagesize);

    ngx_conf_merge_msec_value(conf->greeting_delay,
                              prev->greeting_delay, 0);

    ngx_conf_merge_bitmask_value(conf->auth_methods,
                              prev->auth_methods,
                              (NGX_CONF_BITMASK_SET
                               |NGX_MAIL_AUTH_PLAIN_ENABLED
                               |NGX_MAIL_AUTH_LOGIN_ENABLED));


    cscf = ngx_mail_conf_get_module_srv_conf(cf, ngx_mail_core_module);

    size = sizeof("220  ESMTP ready" CRLF) - 1 + cscf->server_name.len;

    p = ngx_pnalloc(cf->pool, size);
    if (p == NULL) {
        return NGX_CONF_ERROR;
    }

    conf->greeting.len = size;
    conf->greeting.data = p;

    *p++ = '2'; *p++ = '2'; *p++ = '0'; *p++ = ' ';
    p = ngx_cpymem(p, cscf->server_name.data, cscf->server_name.len);
    ngx_memcpy(p, " ESMTP ready" CRLF, sizeof(" ESMTP ready" CRLF) - 1);


    size = sizeof("250 " CRLF) - 1 + cscf->server_name.len;

    p = ngx_pnalloc(cf->pool, size);
    if (p == NULL) {
        return NGX_CONF_ERROR;
    }

    conf->server_name.len = size;
    conf->server_name.data = p;

    *p++ = '2'; *p++ = '5'; *p++ = '0'; *p++ = ' ';
    p = ngx_cpymem(p, cscf->server_name.data, cscf->server_name.len);
    *p++ = CR; *p = LF;


    if (conf->capabilities.nelts == 0) {
        conf->capabilities = prev->capabilities;
```

```
    }

    size = sizeof("250-") - 1 + cscf->server_name.len + sizeof(CRLF) - 1;

    c = conf->capabilities.elts;
    for (i = 0; i < conf->capabilities.nelts; i++) {
        size += sizeof("250 ") - 1 + c[i].len + sizeof(CRLF) - 1;
    }

    auth_enabled = 0;

    for (m = NGX_MAIL_AUTH_PLAIN_ENABLED, i = 0;
         m <= NGX_MAIL_AUTH_CRAM_MD5_ENABLED;
         m <<= 1, i++)
    {
        if (m & conf->auth_methods) {
            size += 1 + ngx_mail_smtp_auth_methods_names[i].len;
            auth_enabled = 1;
        }
    }

    if (auth_enabled) {
        size += sizeof("250 AUTH") - 1 + sizeof(CRLF) - 1;
    }

    p = ngx_pnalloc(cf->pool, size);
    if (p == NULL) {
        return NGX_CONF_ERROR;
    }

    conf->capability.len = size;
    conf->capability.data = p;

    last = p;

    *p++ = '2'; *p++ = '5'; *p++ = '0'; *p++ = '-';
    p = ngx_cpymem(p, cscf->server_name.data, cscf->server_name.len);
    *p++ = CR; *p++ = LF;

    for (i = 0; i < conf->capabilities.nelts; i++) {
        last = p;
        *p++ = '2'; *p++ = '5'; *p++ = '0'; *p++ = '-';
        p = ngx_cpymem(p, c[i].data, c[i].len);
        *p++ = CR; *p++ = LF;
    }

    auth = p;

    if (auth_enabled) {
        last = p;

        *p++ = '2'; *p++ = '5'; *p++ = '0'; *p++ = ' ';
        *p++ = 'A'; *p++ = 'U'; *p++ = 'T'; *p++ = 'H';

        for (m = NGX_MAIL_AUTH_PLAIN_ENABLED, i = 0;
             m <= NGX_MAIL_AUTH_CRAM_MD5_ENABLED;
             m <<= 1, i++)
        {
            if (m & conf->auth_methods) {
                *p++ = ' ';
                p = ngx_cpymem(p, ngx_mail_smtp_auth_methods_names[i].data,
                               ngx_mail_smtp_auth_methods_names[i].len);
            }
        }

        *p++ = CR; *p = LF;

    } else {
        last[3] = ' ';
    }

    size += sizeof("250 STARTTLS" CRLF) - 1;

    p = ngx_pnalloc(cf->pool, size);
    if (p == NULL) {
        return NGX_CONF_ERROR;
```

```
272     }
273
274     conf->starttls_capability.len = size;
275     conf->starttls_capability.data = p;
276
277     p = ngx_cpymem(p, conf->capability.data, conf->capability.len);
278
279     p = ngx_cpymem(p, "250 STARTTLS" CRLF, sizeof("250 STARTTLS" CRLF) - 1);
280
281     p = conf->starttls_capability.data
282         + (last - conf->capability.data) + 3;
283     *p = '-';
284
285     size = (auth - conf->capability.data)
286             + sizeof("250 STARTTLS" CRLF) - 1;
287
288     p = ngx_pnalloc(cf->pool, size);
289     if (p == NULL) {
290         return NGX_CONF_ERROR;
291     }
292
293     conf->starttls_only_capability.len = size;
294     conf->starttls_only_capability.data = p;
295
296     p = ngx_cpymem(p, conf->capability.data, auth - conf->capability.data);
297
298     ngx_memcpy(p, "250 STARTTLS" CRLF, sizeof("250 STARTTLS" CRLF) - 1);
299
300     if (last < auth) {
301         p = conf->starttls_only_capability.data
302             + (last - conf->capability.data) + 3;
303         *p = '-';
304     }
305
306     return NGX_CONF_OK;
307 }
```

One Level Up Top Level

src/mail/ngx_mail_smtp_handler.c - nginx-1.7.10

Global variables defined

- smtp_auth_required
- smtp_bad_sequence
- smtp_bye
- smtp_invalid_argument
- smtp_invalid_command
- smtp_invalid_pipelining
- smtp_next
- smtp_ok
- smtp_password
- smtp_starttls
- smtp_tempunavail
- smtp_unavailable
- smtp_username

Functions defined

- ngx_mail_smtp_auth
- ngx_mail_smtp_auth_state
- ngx_mail_smtp_create_buffer
- ngx_mail_smtp_discard_command
- ngx_mail_smtp_greeting
- ngx_mail_smtp_helo
- ngx_mail_smtp_init_protocol
- ngx_mail_smtp_init_session
- ngx_mail_smtp_invalid_pipelining
- ngx_mail_smtp_log_rejected_command
- ngx_mail_smtp_mail
- ngx_mail_smtp_rcpt
- ngx_mail_smtp_resolve_addr_handler
- ngx_mail_smtp_resolve_name
- ngx_mail_smtp_resolve_name_handler

- ngx_mail_smtp_rset
- ngx_mail_smtp_starttls

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_mail.h>
12  #include <ngx_mail_smtp_module.h>
13
14
15  static void ngx_mail_smtp_resolve_addr_handler(ngx_resolver_ctx_t *ctx);
16  static void ngx_mail_smtp_resolve_name(ngx_event_t *rev);
17  static void ngx_mail_smtp_resolve_name_handler(ngx_resolver_ctx_t *ctx);
18  static void ngx_mail_smtp_greeting(ngx_mail_session_t *s, ngx_connection_t *c);
19  static void ngx_mail_smtp_invalid_pipelining(ngx_event_t *rev);
20  static ngx_int_t ngx_mail_smtp_create_buffer(ngx_mail_session_t *s,
21      ngx_connection_t *c);
22
23  static ngx_int_t ngx_mail_smtp_helo(ngx_mail_session_t *s, ngx_connection_t *c);
24  static ngx_int_t ngx_mail_smtp_auth(ngx_mail_session_t *s, ngx_connection_t *c);
25  static ngx_int_t ngx_mail_smtp_mail(ngx_mail_session_t *s, ngx_connection_t *c);
26  static ngx_int_t ngx_mail_smtp_starttls(ngx_mail_session_t *s,
27      ngx_connection_t *c);
28  static ngx_int_t ngx_mail_smtp_rset(ngx_mail_session_t *s, ngx_connection_t *c);
29  static ngx_int_t ngx_mail_smtp_rcpt(ngx_mail_session_t *s, ngx_connection_t *c);
30
31  static ngx_int_t ngx_mail_smtp_discard_command(ngx_mail_session_t *s,
32      ngx_connection_t *c, char *err);
33  static void ngx_mail_smtp_log_rejected_command(ngx_mail_session_t *s,
34      ngx_connection_t *c, char *err);
35
36
37  static u_char  smtp_ok[] = "250 2.0.0 OK" CRLF;
38  static u_char  smtp_bye[] = "221 2.0.0 Bye" CRLF;
39  static u_char  smtp_starttls[] = "220 2.0.0 Start TLS" CRLF;
40  static u_char  smtp_next[] = "334 " CRLF;
41  static u_char  smtp_username[] = "334 VXNlcm5hbWU6" CRLF;
42  static u_char  smtp_password[] = "334 UGFzc3dvcmQ6" CRLF;
43  static u_char  smtp_invalid_command[] = "500 5.5.1 Invalid command" CRLF;
44  static u_char  smtp_invalid_pipelining[] =
45     "503 5.5.0 Improper use of SMTP command pipelining" CRLF;
46  static u_char  smtp_invalid_argument[] = "501 5.5.4 Invalid argument" CRLF;
47  static u_char  smtp_auth_required[] = "530 5.7.1 Authentication required" CRLF;
48  static u_char  smtp_bad_sequence[] = "503 5.5.1 Bad sequence of commands" CRLF;
49
50
51  static ngx_str_t  smtp_unavailable = ngx_string("[UNAVAILABLE]");
52  static ngx_str_t  smtp_tempunavail = ngx_string("[TEMPUNAVAIL]");
53
54
55  void
56  ngx_mail_smtp_init_session(ngx_mail_session_t *s, ngx_connection_t *c)
57  {
58      ngx_resolver_ctx_t        *ctx;
59      ngx_mail_core_srv_conf_t  *cscf;
60
61      cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);
62
63      if (cscf->resolver == NULL) {
64          s->host = smtp_unavailable;
65          ngx_mail_smtp_greeting(s, c);
66          return;
67      }
68
```

```
#if (NGX_HAVE_UNIX_DOMAIN)
    if (c->sockaddr->sa_family == AF_UNIX) {
        s->host = smtp_tempunavail;
        ngx_mail_smtp_greeting(s, c);
        return;
    }
#endif

    c->log->action = "in resolving client address";

    ctx = ngx_resolve_start(cscf->resolver, NULL);
    if (ctx == NULL) {
        ngx_mail_close_connection(c);
        return;
    }

    ctx->addr.sockaddr = c->sockaddr;
    ctx->addr.socklen = c->socklen;
    ctx->handler = ngx_mail_smtp_resolve_addr_handler;
    ctx->data = s;
    ctx->timeout = cscf->resolver_timeout;

    if (ngx_resolve_addr(ctx) != NGX_OK) {
        ngx_mail_close_connection(c);
    }
}


static void
ngx_mail_smtp_resolve_addr_handler(ngx_resolver_ctx_t *ctx)
{
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    s = ctx->data;
    c = s->connection;

    if (ctx->state) {
        ngx_log_error(NGX_LOG_ERR, c->log, 0,
                      "%V could not be resolved (%i: %s)",
                      &c->addr_text, ctx->state,
                      ngx_resolver_strerror(ctx->state));

        if (ctx->state == NGX_RESOLVE_NXDOMAIN) {
            s->host = smtp_unavailable;

        } else {
            s->host = smtp_tempunavail;
        }

        ngx_resolve_addr_done(ctx);

        ngx_mail_smtp_greeting(s, s->connection);

        return;
    }

    c->log->action = "in resolving client hostname";

    s->host.data = ngx_pstrdup(c->pool, &ctx->name);
    if (s->host.data == NULL) {
        ngx_resolve_addr_done(ctx);
        ngx_mail_close_connection(c);
        return;
    }

    s->host.len = ctx->name.len;

    ngx_resolve_addr_done(ctx);

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "address resolved: %V", &s->host);

    c->read->handler = ngx_mail_smtp_resolve_name;

    ngx_post_event(c->read, &ngx_posted_events);
```

```
}


static void
ngx_mail_smtp_resolve_name(ngx_event_t *rev)
{
    ngx_connection_t          *c;
    ngx_mail_session_t        *s;
    ngx_resolver_ctx_t        *ctx;
    ngx_mail_core_srv_conf_t  *cscf;

    c = rev->data;
    s = c->data;

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

    ctx = ngx_resolve_start(cscf->resolver, NULL);
    if (ctx == NULL) {
        ngx_mail_close_connection(c);
        return;
    }

    ctx->name = s->host;
    ctx->handler = ngx_mail_smtp_resolve_name_handler;
    ctx->data = s;
    ctx->timeout = cscf->resolver_timeout;

    if (ngx_resolve_name(ctx) != NGX_OK) {
        ngx_mail_close_connection(c);
    }
}


static void
ngx_mail_smtp_resolve_name_handler(ngx_resolver_ctx_t *ctx)
{
    ngx_uint_t           i;
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    s = ctx->data;
    c = s->connection;

    if (ctx->state) {
        ngx_log_error(NGX_LOG_ERR, c->log, 0,
                      "\"%V\" could not be resolved (%i: %s)",
                      &ctx->name, ctx->state,
                      ngx_resolver_strerror(ctx->state));

        if (ctx->state == NGX_RESOLVE_NXDOMAIN) {
            s->host = smtp_unavailable;

        } else {
            s->host = smtp_tempunavail;
        }

    } else {

#if (NGX_DEBUG)
        {
        u_char     text[NGX_SOCKADDR_STRLEN];
        ngx_str_t  addr;

        addr.data = text;

        for (i = 0; i < ctx->naddrs; i++) {
            addr.len = ngx_sock_ntop(ctx->addrs[i].sockaddr,
                                     ctx->addrs[i].socklen,
                                     text, NGX_SOCKADDR_STRLEN, 0);

            ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                           "name was resolved to %V", &addr);
        }
        }
#endif

```

```
        for (i = 0; i < ctx->naddrs; i++) {
            if (ngx_cmp_sockaddr(ctx->addrs[i].sockaddr, ctx->addrs[i].socklen,
                                 c->sockaddr, c->socklen, 0)
                == NGX_OK)
            {
                goto found;
            }
        }

        s->host = smtp_unavailable;
    }

found:

    ngx_resolve_name_done(ctx);

    ngx_mail_smtp_greeting(s, c);
}


static void
ngx_mail_smtp_greeting(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_msec_t                 timeout;
    ngx_mail_core_srv_conf_t  *cscf;
    ngx_mail_smtp_srv_conf_t  *sscf;

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "smtp greeting for \"%V\"", &s->host);

    cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);
    sscf = ngx_mail_get_module_srv_conf(s, ngx_mail_smtp_module);

    timeout = sscf->greeting_delay ? sscf->greeting_delay : cscf->timeout;
    ngx_add_timer(c->read, timeout);

    if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
        ngx_mail_close_connection(c);
    }

    if (sscf->greeting_delay) {
         c->read->handler = ngx_mail_smtp_invalid_pipelining;
         return;
    }

    c->read->handler = ngx_mail_smtp_init_protocol;

    s->out = sscf->greeting;

    ngx_mail_send(c->write);
}


static void
ngx_mail_smtp_invalid_pipelining(ngx_event_t *rev)
{
    ngx_connection_t          *c;
    ngx_mail_session_t        *s;
    ngx_mail_core_srv_conf_t  *cscf;
    ngx_mail_smtp_srv_conf_t  *sscf;

    c = rev->data;
    s = c->data;

    c->log->action = "in delay pipelining state";

    if (rev->timedout) {

        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "delay greeting");

        rev->timedout = 0;

        cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

        c->read->handler = ngx_mail_smtp_init_protocol;

```

```
        ngx_add_timer(c->read, cscf->timeout);

        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            ngx_mail_close_connection(c);
            return;
        }

        sscf = ngx_mail_get_module_srv_conf(s, ngx_mail_smtp_module);

        s->out = sscf->greeting;

    } else {

        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "invalid pipelining");

        if (s->buffer == NULL) {
            if (ngx_mail_smtp_create_buffer(s, c) != NGX_OK) {
                return;
            }
        }

        if (ngx_mail_smtp_discard_command(s, c,
                                "client was rejected before greeting: \"%V\"")
            != NGX_OK)
        {
            return;
        }

        ngx_str_set(&s->out, smtp_invalid_pipelining);
        s->quit = 1;
    }

    ngx_mail_send(c->write);
}


void
ngx_mail_smtp_init_protocol(ngx_event_t *rev)
{
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    c = rev->data;

    c->log->action = "in auth state";

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        c->timedout = 1;
        ngx_mail_close_connection(c);
        return;
    }

    s = c->data;

    if (s->buffer == NULL) {
        if (ngx_mail_smtp_create_buffer(s, c) != NGX_OK) {
            return;
        }
    }

    s->mail_state = ngx_smtp_start;
    c->read->handler = ngx_mail_smtp_auth_state;

    ngx_mail_smtp_auth_state(rev);
}


static ngx_int_t
ngx_mail_smtp_create_buffer(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_mail_smtp_srv_conf_t  *sscf;

    if (ngx_array_init(&s->args, c->pool, 2, sizeof(ngx_str_t)) == NGX_ERROR) {
        ngx_mail_session_internal_server_error(s);
        return NGX_ERROR;
```

```
    }

    sscf = ngx_mail_get_module_srv_conf(s, ngx_mail_smtp_module);

    s->buffer = ngx_create_temp_buf(c->pool, sscf->client_buffer_size);
    if (s->buffer == NULL) {
        ngx_mail_session_internal_server_error(s);
        return NGX_ERROR;
    }

    return NGX_OK;
}


void
ngx_mail_smtp_auth_state(ngx_event_t *rev)
{
    ngx_int_t            rc;
    ngx_connection_t    *c;
    ngx_mail_session_t  *s;

    c = rev->data;
    s = c->data;

    ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "smtp auth state");

    if (rev->timedout) {
        ngx_log_error(NGX_LOG_INFO, c->log, NGX_ETIMEDOUT, "client timed out");
        c->timedout = 1;
        ngx_mail_close_connection(c);
        return;
    }

    if (s->out.len) {
        ngx_log_debug0(NGX_LOG_DEBUG_MAIL, c->log, 0, "smtp send handler busy");
        s->blocked = 1;
        return;
    }

    s->blocked = 0;

    rc = ngx_mail_read_command(s, c);

    if (rc == NGX_AGAIN || rc == NGX_ERROR) {
        return;
    }

    ngx_str_set(&s->out, smtp_ok);

    if (rc == NGX_OK) {
        switch (s->mail_state) {

        case ngx_smtp_start:

            switch (s->command) {

            case NGX_SMTP_HELO:
            case NGX_SMTP_EHLO:
                rc = ngx_mail_smtp_helo(s, c);
                break;

            case NGX_SMTP_AUTH:
                rc = ngx_mail_smtp_auth(s, c);
                break;

            case NGX_SMTP_QUIT:
                s->quit = 1;
                ngx_str_set(&s->out, smtp_bye);
                break;

            case NGX_SMTP_MAIL:
                rc = ngx_mail_smtp_mail(s, c);
                break;

            case NGX_SMTP_RCPT:
                rc = ngx_mail_smtp_rcpt(s, c);
```

```
            break;

        case NGX_SMTP_RSET:
            rc = ngx_mail_smtp_rset(s, c);
            break;

        case NGX_SMTP_NOOP:
            break;

        case NGX_SMTP_STARTTLS:
            rc = ngx_mail_smtp_starttls(s, c);
            ngx_str_set(&s->out, smtp_starttls);
            break;

        default:
            rc = NGX_MAIL_PARSE_INVALID_COMMAND;
            break;
        }

        break;

    case ngx_smtp_auth_login_username:
        rc = ngx_mail_auth_login_username(s, c, 0);

        ngx_str_set(&s->out, smtp_password);
        s->mail_state = ngx_smtp_auth_login_password;
        break;

    case ngx_smtp_auth_login_password:
        rc = ngx_mail_auth_login_password(s, c);
        break;

    case ngx_smtp_auth_plain:
        rc = ngx_mail_auth_plain(s, c, 0);
        break;

    case ngx_smtp_auth_cram_md5:
        rc = ngx_mail_auth_cram_md5(s, c);
        break;
    }
}

if (s->buffer->pos < s->buffer->last) {
    s->blocked = 1;
}

switch (rc) {

case NGX_DONE:
    ngx_mail_auth(s, c);
    return;

case NGX_ERROR:
    ngx_mail_session_internal_server_error(s);
    return;

case NGX_MAIL_PARSE_INVALID_COMMAND:
    s->mail_state = ngx_smtp_start;
    s->state = 0;
    ngx_str_set(&s->out, smtp_invalid_command);

    /* fall through */

case NGX_OK:
    s->args.nelts = 0;

    if (s->buffer->pos == s->buffer->last) {
        s->buffer->pos = s->buffer->start;
        s->buffer->last = s->buffer->start;
    }

    if (s->state) {
        s->arg_start = s->buffer->pos;
    }

    ngx_mail_send(c->write);
```

```
    }
}


static ngx_int_t
ngx_mail_smtp_helo(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t                  *arg;
    ngx_mail_smtp_srv_conf_t  *sscf;

    if (s->args.nelts != 1) {
        ngx_str_set(&s->out, smtp_invalid_argument);
        s->state = 0;
        return NGX_OK;
    }

    arg = s->args.elts;

    s->smtp_helo.len = arg[0].len;

    s->smtp_helo.data = ngx_pnalloc(c->pool, arg[0].len);
    if (s->smtp_helo.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->smtp_helo.data, arg[0].data, arg[0].len);

    ngx_str_null(&s->smtp_from);
    ngx_str_null(&s->smtp_to);

    sscf = ngx_mail_get_module_srv_conf(s, ngx_mail_smtp_module);

    if (s->command == NGX_SMTP_HELO) {
        s->out = sscf->server_name;

    } else {
        s->esmtp = 1;

#if (NGX_MAIL_SSL)

        if (c->ssl == NULL) {
            ngx_mail_ssl_conf_t  *sslcf;

            sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);

            if (sslcf->starttls == NGX_MAIL_STARTTLS_ON) {
                s->out = sscf->starttls_capability;
                return NGX_OK;
            }

            if (sslcf->starttls == NGX_MAIL_STARTTLS_ONLY) {
                s->out = sscf->starttls_only_capability;
                return NGX_OK;
            }
        }
#endif

        s->out = sscf->capability;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_mail_smtp_auth(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_int_t                   rc;
    ngx_mail_core_srv_conf_t  *cscf;
    ngx_mail_smtp_srv_conf_t  *sscf;

#if (NGX_MAIL_SSL)
    if (ngx_mail_starttls_only(s, c)) {
        return NGX_MAIL_PARSE_INVALID_COMMAND;
    }
#endif
```

```
601
602     if (s->args.nelts == 0) {
603         ngx_str_set(&s->out, smtp_invalid_argument);
604         s->state = 0;
605         return NGX_OK;
606     }
607
608     rc = ngx_mail_auth_parse(s, c);
609
610     switch (rc) {
611
612     case NGX_MAIL_AUTH_LOGIN:
613
614         ngx_str_set(&s->out, smtp_username);
615         s->mail_state = ngx_smtp_auth_login_username;
616
617         return NGX_OK;
618
619     case NGX_MAIL_AUTH_LOGIN_USERNAME:
620
621         ngx_str_set(&s->out, smtp_password);
622         s->mail_state = ngx_smtp_auth_login_password;
623
624         return ngx_mail_auth_login_username(s, c, 1);
625
626     case NGX_MAIL_AUTH_PLAIN:
627
628         ngx_str_set(&s->out, smtp_next);
629         s->mail_state = ngx_smtp_auth_plain;
630
631         return NGX_OK;
632
633     case NGX_MAIL_AUTH_CRAM_MD5:
634
635         sscf = ngx_mail_get_module_srv_conf(s, ngx_mail_smtp_module);
636
637         if (!(sscf->auth_methods & NGX_MAIL_AUTH_CRAM_MD5_ENABLED)) {
638             return NGX_MAIL_PARSE_INVALID_COMMAND;
639         }
640
641         if (s->salt.data == NULL) {
642             cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);
643
644             if (ngx_mail_salt(s, c, cscf) != NGX_OK) {
645                 return NGX_ERROR;
646             }
647         }
648
649         if (ngx_mail_auth_cram_md5_salt(s, c, "334 ", 4) == NGX_OK) {
650             s->mail_state = ngx_smtp_auth_cram_md5;
651             return NGX_OK;
652         }
653
654         return NGX_ERROR;
655     }
656
657     return rc;
658 }
659
660
661 static ngx_int_t
662 ngx_mail_smtp_mail(ngx_mail_session_t *s, ngx_connection_t *c)
663 {
664     ngx_str_t                  *arg, cmd;
665     ngx_mail_smtp_srv_conf_t   *sscf;
666
667     sscf = ngx_mail_get_module_srv_conf(s, ngx_mail_smtp_module);
668
669     if (!(sscf->auth_methods & NGX_MAIL_AUTH_NONE_ENABLED)) {
670         ngx_mail_smtp_log_rejected_command(s, c, "client was rejected: \"%V\"");
671         ngx_str_set(&s->out, smtp_auth_required);
672         return NGX_OK;
673     }
674
675     /* auth none */
676
```

```
    if (s->smtp_from.len) {
        ngx_str_set(&s->out, smtp_bad_sequence);
        return NGX_OK;
    }

    if (s->args.nelts == 0) {
        ngx_str_set(&s->out, smtp_invalid_argument);
        return NGX_OK;
    }

    arg = s->args.elts;
    arg += s->args.nelts - 1;

    cmd.len = arg->data + arg->len - s->cmd.data;
    cmd.data = s->cmd.data;

    s->smtp_from.len = cmd.len;

    s->smtp_from.data = ngx_pnalloc(c->pool, cmd.len);
    if (s->smtp_from.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->smtp_from.data, cmd.data, cmd.len);

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "smtp mail from:\"%V\"", &s->smtp_from);

    ngx_str_set(&s->out, smtp_ok);

    return NGX_OK;
}


static ngx_int_t
ngx_mail_smtp_rcpt(ngx_mail_session_t *s, ngx_connection_t *c)
{
    ngx_str_t  *arg, cmd;

    if (s->smtp_from.len == 0) {
        ngx_str_set(&s->out, smtp_bad_sequence);
        return NGX_OK;
    }

    if (s->args.nelts == 0) {
        ngx_str_set(&s->out, smtp_invalid_argument);
        return NGX_OK;
    }

    arg = s->args.elts;
    arg += s->args.nelts - 1;

    cmd.len = arg->data + arg->len - s->cmd.data;
    cmd.data = s->cmd.data;

    s->smtp_to.len = cmd.len;

    s->smtp_to.data = ngx_pnalloc(c->pool, cmd.len);
    if (s->smtp_to.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(s->smtp_to.data, cmd.data, cmd.len);

    ngx_log_debug1(NGX_LOG_DEBUG_MAIL, c->log, 0,
                   "smtp rcpt to:\"%V\"", &s->smtp_to);

    s->auth_method = NGX_MAIL_AUTH_NONE;

    return NGX_DONE;
}


static ngx_int_t
ngx_mail_smtp_rset(ngx_mail_session_t *s, ngx_connection_t *c)
{
```

```
    ngx_str_null(&s->smtp_from);
    ngx_str_null(&s->smtp_to);
    ngx_str_set(&s->out, smtp_ok);

    return NGX_OK;
}


static ngx_int_t
ngx_mail_smtp_starttls(ngx_mail_session_t *s, ngx_connection_t *c)
{
#if (NGX_MAIL_SSL)
    ngx_mail_ssl_conf_t  *sslcf;

    if (c->ssl == NULL) {
        sslcf = ngx_mail_get_module_srv_conf(s, ngx_mail_ssl_module);
        if (sslcf->starttls) {

            /*
             * RFC3207 requires us to discard any knowledge
             * obtained from client before STARTTLS.
             */

            ngx_str_null(&s->smtp_helo);
            ngx_str_null(&s->smtp_from);
            ngx_str_null(&s->smtp_to);

            s->buffer->pos = s->buffer->start;
            s->buffer->last = s->buffer->start;

            c->read->handler = ngx_mail_starttls_handler;
            return NGX_OK;
        }
    }

#endif

    return NGX_MAIL_PARSE_INVALID_COMMAND;
}


static ngx_int_t
ngx_mail_smtp_discard_command(ngx_mail_session_t *s, ngx_connection_t *c,
    char *err)
{
    ssize_t    n;

    n = c->recv(c, s->buffer->last, s->buffer->end - s->buffer->last);

    if (n == NGX_ERROR || n == 0) {
        ngx_mail_close_connection(c);
        return NGX_ERROR;
    }

    if (n > 0) {
        s->buffer->last += n;
    }

    if (n == NGX_AGAIN) {
        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            ngx_mail_session_internal_server_error(s);
            return NGX_ERROR;
        }

        return NGX_AGAIN;
    }

    ngx_mail_smtp_log_rejected_command(s, c, err);

    s->buffer->pos = s->buffer->start;
    s->buffer->last = s->buffer->start;

    return NGX_OK;
}
```

```
829 static void
830 ngx_mail_smtp_log_rejected_command(ngx_mail_session_t *s, ngx_connection_t *c,
831     char *err)
832 {
833     u_char      ch;
834     ngx_str_t   cmd;
835     ngx_uint_t  i;
836
837     if (c->log->log_level < NGX_LOG_INFO) {
838         return;
839     }
840
841     cmd.len = s->buffer->last - s->buffer->start;
842     cmd.data = s->buffer->start;
843
844     for (i = 0; i < cmd.len; i++) {
845         ch = cmd.data[i];
846
847         if (ch != CR && ch != LF) {
848             continue;
849         }
850
851         cmd.data[i] = '_';
852     }
853
854     cmd.len = i;
855
856     ngx_log_error(NGX_LOG_INFO, c->log, 0, err, &cmd);
857 }
```

One Level Up Top Level

src/mail/ngx_mail_smtp_module.h - nginx-1.7.10

Data types defined

- ngx_mail_smtp_srv_conf_t

Macros defined

- _NGX_MAIL_SMTP_MODULE_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_MAIL_SMTP_MODULE_H_INCLUDED_
9  #define _NGX_MAIL_SMTP_MODULE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_mail.h>
15  #include <ngx_mail_smtp_module.h>
16
17
18  typedef struct {
19      ngx_msec_t   greeting_delay;
20
21      size_t       client_buffer_size;
22
23      ngx_str_t    capability;
24      ngx_str_t    starttls_capability;
25      ngx_str_t    starttls_only_capability;
26
27      ngx_str_t    server_name;
28      ngx_str_t    greeting;
29
30      ngx_uint_t   auth_methods;
31
32      ngx_array_t  capabilities;
33  } ngx_mail_smtp_srv_conf_t;
34
35
36  void ngx_mail_smtp_init_session(ngx_mail_session_t *s, ngx_connection_t *c);
37  void ngx_mail_smtp_init_protocol(ngx_event_t *rev);
38  void ngx_mail_smtp_auth_state(ngx_event_t *rev);
39  ngx_int_t ngx_mail_smtp_parse_command(ngx_mail_session_t *s);
40
41
42  extern ngx_module_t  ngx_mail_smtp_module;
43
44
45  #endif /* _NGX_MAIL_SMTP_MODULE_H_INCLUDED_ */
```

src/event/ngx_event_accept.c - nginx-1.7.10

Functions defined

- ngx_accept_log_error
- ngx_close_accepted_connection
- ngx_disable_accept_events
- ngx_enable_accept_events
- ngx_event_accept
- ngx_trylock_accept_mutex

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10 #include <ngx_event.h>
11
12
13 static ngx_int_t ngx_enable_accept_events(ngx_cycle_t *cycle);
14 static ngx_int_t ngx_disable_accept_events(ngx_cycle_t *cycle);
15 static void ngx_close_accepted_connection(ngx_connection_t *c);
16
17
18 void
19 ngx_event_accept(ngx_event_t *ev)
20 {
21     socklen_t          socklen;
22     ngx_err_t          err;
23     ngx_log_t         *log;
24     ngx_uint_t         level;
25     ngx_socket_t       s;
26     ngx_event_t       *rev, *wev;
27     ngx_listening_t   *ls;
28     ngx_connection_t  *c, *lc;
29     ngx_event_conf_t  *ecf;
30     u_char             sa[NGX_SOCKADDRLEN];
31 #if (NGX_HAVE_ACCEPT4)
32     static ngx_uint_t  use_accept4 = 1;
33 #endif
34
35     if (ev->timedout) {
36         if (ngx_enable_accept_events((ngx_cycle_t *) ngx_cycle) != NGX_OK) {
37             return;
38         }
39
40         ev->timedout = 0;
41     }
42
43     ecf = ngx_event_get_conf(ngx_cycle->conf_ctx, ngx_event_core_module);
44
45     if (ngx_event_flags & NGX_USE_RTSIG_EVENT) {
46         ev->available = 1;
47
48     } else if (!(ngx_event_flags & NGX_USE_KQUEUE_EVENT)) {
49         ev->available = ecf->multi_accept;
50     }
51
```

```
    lc = ev->data;
    ls = lc->listening;
    ev->ready = 0;

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "accept on %V, ready: %d", &ls->addr_text, ev->available);

    do {
        socklen = NGX_SOCKADDRLEN;

#if (NGX_HAVE_ACCEPT4)
        if (use_accept4) {
            s = accept4(lc->fd, (struct sockaddr *) sa, &socklen,
                        SOCK_NONBLOCK);
        } else {
            s = accept(lc->fd, (struct sockaddr *) sa, &socklen);
        }
#else
        s = accept(lc->fd, (struct sockaddr *) sa, &socklen);
#endif

        if (s == (ngx_socket_t) -1) {
            err = ngx_socket_errno;

            if (err == NGX_EAGAIN) {
                ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ev->log, err,
                               "accept() not ready");
                return;
            }

            level = NGX_LOG_ALERT;

            if (err == NGX_ECONNABORTED) {
                level = NGX_LOG_ERR;

            } else if (err == NGX_EMFILE || err == NGX_ENFILE) {
                level = NGX_LOG_CRIT;
            }

#if (NGX_HAVE_ACCEPT4)
            ngx_log_error(level, ev->log, err,
                          use_accept4 ? "accept4() failed" : "accept() failed");

            if (use_accept4 && err == NGX_ENOSYS) {
                use_accept4 = 0;
                ngx_inherited_nonblocking = 0;
                continue;
            }
#else
            ngx_log_error(level, ev->log, err, "accept() failed");
#endif

            if (err == NGX_ECONNABORTED) {
                if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
                    ev->available--;
                }

                if (ev->available) {
                    continue;
                }
            }

            if (err == NGX_EMFILE || err == NGX_ENFILE) {
                if (ngx_disable_accept_events((ngx_cycle_t *) ngx_cycle)
                    != NGX_OK)
                {
                    return;
                }

                if (ngx_use_accept_mutex) {
                    if (ngx_accept_mutex_held) {
                        ngx_shmtx_unlock(&ngx_accept_mutex);
                        ngx_accept_mutex_held = 0;
                    }

                    ngx_accept_disabled = 1;
```

```
                } else {
                    ngx_add_timer(ev, ecf->accept_mutex_delay);
                }
            }

            return;
        }

#if (NGX_STAT_STUB)
        (void) ngx_atomic_fetch_add(ngx_stat_accepted, 1);
#endif

        ngx_accept_disabled = ngx_cycle->connection_n / 8
                              - ngx_cycle->free_connection_n;

        c = ngx_get_connection(s, ev->log);

        if (c == NULL) {
            if (ngx_close_socket(s) == -1) {
                ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_socket_errno,
                              ngx_close_socket_n " failed");
            }

            return;
        }

#if (NGX_STAT_STUB)
        (void) ngx_atomic_fetch_add(ngx_stat_active, 1);
#endif

        c->pool = ngx_create_pool(ls->pool_size, ev->log);
        if (c->pool == NULL) {
            ngx_close_accepted_connection(c);
            return;
        }

        c->sockaddr = ngx_palloc(c->pool, socklen);
        if (c->sockaddr == NULL) {
            ngx_close_accepted_connection(c);
            return;
        }

        ngx_memcpy(c->sockaddr, sa, socklen);

        log = ngx_palloc(c->pool, sizeof(ngx_log_t));
        if (log == NULL) {
            ngx_close_accepted_connection(c);
            return;
        }

        /* set a blocking mode for aio and non-blocking mode for others */

        if (ngx_inherited_nonblocking) {
            if (ngx_event_flags & NGX_USE_AIO_EVENT) {
                if (ngx_blocking(s) == -1) {
                    ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_socket_errno,
                                  ngx_blocking_n " failed");
                    ngx_close_accepted_connection(c);
                    return;
                }
            }

        } else {
            if (!(ngx_event_flags & (NGX_USE_AIO_EVENT|NGX_USE_RTSIG_EVENT))) {
                if (ngx_nonblocking(s) == -1) {
                    ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_socket_errno,
                                  ngx_nonblocking_n " failed");
                    ngx_close_accepted_connection(c);
                    return;
                }
            }
        }

        *log = ls->log;

```

```
        c->recv = ngx_recv;
        c->send = ngx_send;
        c->recv_chain = ngx_recv_chain;
        c->send_chain = ngx_send_chain;

        c->log = log;
        c->pool->log = log;

        c->socklen = socklen;
        c->listening = ls;
        c->local_sockaddr = ls->sockaddr;
        c->local_socklen = ls->socklen;

        c->unexpected_eof = 1;

#if (NGX_HAVE_UNIX_DOMAIN)
        if (c->sockaddr->sa_family == AF_UNIX) {
            c->tcp_nopush = NGX_TCP_NOPUSH_DISABLED;
            c->tcp_nodelay = NGX_TCP_NODELAY_DISABLED;
#if (NGX_SOLARIS)
            /* Solaris's sendfilev() supports AF_NCA, AF_INET, and AF_INET6 */
            c->sendfile = 0;
#endif
        }
#endif

        rev = c->read;
        wev = c->write;

        wev->ready = 1;

        if (ngx_event_flags & (NGX_USE_AIO_EVENT|NGX_USE_RTSIG_EVENT)) {
            /* rtsig, aio, iocp */
            rev->ready = 1;
        }

        if (ev->deferred_accept) {
            rev->ready = 1;
#if (NGX_HAVE_KQUEUE)
            rev->available = 1;
#endif
        }

        rev->log = log;
        wev->log = log;

        /*
         * TODO: MT: - ngx_atomic_fetch_add()
         *             or protection by critical section or light mutex
         *
         * TODO: MP: - allocated in a shared memory
         *           - ngx_atomic_fetch_add()
         *             or protection by critical section or light mutex
         */

        c->number = ngx_atomic_fetch_add(ngx_connection_counter, 1);

#if (NGX_STAT_STUB)
        (void) ngx_atomic_fetch_add(ngx_stat_handled, 1);
#endif

        if (ls->addr_ntop) {
            c->addr_text.data = ngx_pnalloc(c->pool, ls->addr_text_max_len);
            if (c->addr_text.data == NULL) {
                ngx_close_accepted_connection(c);
                return;
            }

            c->addr_text.len = ngx_sock_ntop(c->sockaddr, c->socklen,
                                             c->addr_text.data,
                                             ls->addr_text_max_len, 0);
            if (c->addr_text.len == 0) {
                ngx_close_accepted_connection(c);
                return;
            }
        }
```

```
#if (NGX_DEBUG)
        {

        ngx_str_t             addr;
        struct sockaddr_in   *sin;
        ngx_cidr_t           *cidr;
        ngx_uint_t            i;
        u_char                text[NGX_SOCKADDR_STRLEN];
#if (NGX_HAVE_INET6)
        struct sockaddr_in6  *sin6;
        ngx_uint_t            n;
#endif

        cidr = ecf->debug_connection.elts;
        for (i = 0; i < ecf->debug_connection.nelts; i++) {
            if (cidr[i].family != (ngx_uint_t) c->sockaddr->sa_family) {
                goto next;
            }

            switch (cidr[i].family) {

#if (NGX_HAVE_INET6)
            case AF_INET6:
                sin6 = (struct sockaddr_in6 *) c->sockaddr;
                for (n = 0; n < 16; n++) {
                    if ((sin6->sin6_addr.s6_addr[n]
                        & cidr[i].u.in6.mask.s6_addr[n])
                        != cidr[i].u.in6.addr.s6_addr[n])
                    {
                        goto next;
                    }
                }
                break;
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
            case AF_UNIX:
                break;
#endif

            default: /* AF_INET */
                sin = (struct sockaddr_in *) c->sockaddr;
                if ((sin->sin_addr.s_addr & cidr[i].u.in.mask)
                    != cidr[i].u.in.addr)
                {
                    goto next;
                }
                break;
            }

            log->log_level = NGX_LOG_DEBUG_CONNECTION|NGX_LOG_DEBUG_ALL;
            break;

        next:
            continue;
        }

        if (log->log_level & NGX_LOG_DEBUG_EVENT) {
            addr.data = text;
            addr.len = ngx_sock_ntop(c->sockaddr, c->socklen, text,
                                     NGX_SOCKADDR_STRLEN, 1);

            ngx_log_debug3(NGX_LOG_DEBUG_EVENT, log, 0,
                           "*%uA accept: %V fd:%d", c->number, &addr, s);
        }

        }
#endif

        if (ngx_add_conn && (ngx_event_flags & NGX_USE_EPOLL_EVENT) == 0) {
            if (ngx_add_conn(c) == NGX_ERROR) {
                ngx_close_accepted_connection(c);
                return;
            }
        }
```

```
356
357         log->data = NULL;
358         log->handler = NULL;
359
360         ls->handler(c);
361
362         if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
363             ev->available--;
364         }
365
366     } while (ev->available);
367 }
368
369
370 ngx_int_t
371 ngx_trylock_accept_mutex(ngx_cycle_t *cycle)
372 {
373     if (ngx_shmtx_trylock(&ngx_accept_mutex)) {
374
375         ngx_log_debug0(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
376                        "accept mutex locked");
377
378         if (ngx_accept_mutex_held
379             && ngx_accept_events == 0
380             && !(ngx_event_flags & NGX_USE_RTSIG_EVENT))
381         {
382             return NGX_OK;
383         }
384
385         if (ngx_enable_accept_events(cycle) == NGX_ERROR) {
386             ngx_shmtx_unlock(&ngx_accept_mutex);
387             return NGX_ERROR;
388         }
389
390         ngx_accept_events = 0;
391         ngx_accept_mutex_held = 1;
392
393         return NGX_OK;
394     }
395
396     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
397                    "accept mutex lock failed: %ui", ngx_accept_mutex_held);
398
399     if (ngx_accept_mutex_held) {
400         if (ngx_disable_accept_events(cycle) == NGX_ERROR) {
401             return NGX_ERROR;
402         }
403
404         ngx_accept_mutex_held = 0;
405     }
406
407     return NGX_OK;
408 }
409
410
411 static ngx_int_t
412 ngx_enable_accept_events(ngx_cycle_t *cycle)
413 {
414     ngx_uint_t         i;
415     ngx_listening_t   *ls;
416     ngx_connection_t  *c;
417
418     ls = cycle->listening.elts;
419     for (i = 0; i < cycle->listening.nelts; i++) {
420
421         c = ls[i].connection;
422
423         if (c->read->active) {
424             continue;
425         }
426
427         if (ngx_event_flags & NGX_USE_RTSIG_EVENT) {
428
429             if (ngx_add_conn(c) == NGX_ERROR) {
430                 return NGX_ERROR;
431             }
```

```
        } else {
            if (ngx_add_event(c->read, NGX_READ_EVENT, 0) == NGX_ERROR) {
                return NGX_ERROR;
            }
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_disable_accept_events(ngx_cycle_t *cycle)
{
    ngx_uint_t         i;
    ngx_listening_t   *ls;
    ngx_connection_t  *c;

    ls = cycle->listening.elts;
    for (i = 0; i < cycle->listening.nelts; i++) {

        c = ls[i].connection;

        if (!c->read->active) {
            continue;
        }

        if (ngx_event_flags & NGX_USE_RTSIG_EVENT) {
            if (ngx_del_conn(c, NGX_DISABLE_EVENT) == NGX_ERROR) {
                return NGX_ERROR;
            }

        } else {
            if (ngx_del_event(c->read, NGX_READ_EVENT, NGX_DISABLE_EVENT)
                == NGX_ERROR)
            {
                return NGX_ERROR;
            }
        }
    }

    return NGX_OK;
}


static void
ngx_close_accepted_connection(ngx_connection_t *c)
{
    ngx_socket_t  fd;

    ngx_free_connection(c);

    fd = c->fd;
    c->fd = (ngx_socket_t) -1;

    if (ngx_close_socket(fd) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_socket_errno,
                      ngx_close_socket_n " failed");
    }

    if (c->pool) {
        ngx_destroy_pool(c->pool);
    }

#if (NGX_STAT_STUB)
    (void) ngx_atomic_fetch_add(ngx_stat_active, -1);
#endif
}


u_char *
ngx_accept_log_error(ngx_log_t *log, u_char *buf, size_t len)
{
    return ngx_snprintf(buf, len, " while accepting new connection on %V",
                        log->data);
```

```
508 }
```

One Level Up Top Level

src/http/ngx_http_core_module.c - nginx-1.7.10

Global variables defined

- ngx_conf_deprecated_open_file_cache_retest
- ngx_conf_deprecated_optimize_server_names
- ngx_conf_deprecated_satisfy_any
- ngx_http_client_temp_path
- ngx_http_core_aio
- ngx_http_core_commands
- ngx_http_core_default_types
- ngx_http_core_get_method
- ngx_http_core_if_modified_since
- ngx_http_core_image_gif_type
- ngx_http_core_image_jpeg_type
- ngx_http_core_keepalive_disable
- ngx_http_core_lingering_close
- ngx_http_core_lowat_post
- ngx_http_core_module
- ngx_http_core_module_ctx
- ngx_http_core_pool_size_p
- ngx_http_core_request_body_in_file
- ngx_http_core_satisfy
- ngx_http_core_text_html_type
- ngx_http_gzip_http_version
- ngx_http_gzip_no_cache
- ngx_http_gzip_no_store
- ngx_http_gzip_private
- ngx_http_gzip_proxied_mask
- ngx_methods_names

Data types defined

- ngx_http_method_name_t

Functions defined

- ngx_http_auth_basic_user
- ngx_http_cleanup_add
- ngx_http_core_access_phase
- ngx_http_core_content_phase
- ngx_http_core_create_loc_conf
- ngx_http_core_create_main_conf
- ngx_http_core_create_srv_conf
- ngx_http_core_directio
- ngx_http_core_error_log
- ngx_http_core_error_page
- ngx_http_core_find_config_phase
- ngx_http_core_find_location
- ngx_http_core_find_static_location
- ngx_http_core_generic_phase
- ngx_http_core_init_main_conf
- ngx_http_core_internal
- ngx_http_core_keepalive
- ngx_http_core_limit_except
- ngx_http_core_listen
- ngx_http_core_location
- ngx_http_core_lowat_check
- ngx_http_core_merge_loc_conf
- ngx_http_core_merge_srv_conf
- ngx_http_core_open_file_cache
- ngx_http_core_pool_size
- ngx_http_core_post_access_phase
- ngx_http_core_post_rewrite_phase
- ngx_http_core_preconfiguration
- ngx_http_core_regex_location
- ngx_http_core_resolver
- ngx_http_core_rewrite_phase
- ngx_http_core_root
- ngx_http_core_run_phases

- ngx_http_core_server
- ngx_http_core_server_name
- ngx_http_core_try_files
- ngx_http_core_try_files_phase
- ngx_http_core_type
- ngx_http_core_types
- ngx_http_disable_symlinks
- ngx_http_get_forwarded_addr
- ngx_http_get_forwarded_addr_internal
- ngx_http_gzip_accept_encoding
- ngx_http_gzip_disable
- ngx_http_gzip_ok
- ngx_http_gzip_quantity
- ngx_http_handler
- ngx_http_internal_redirect
- ngx_http_map_uri_to_path
- ngx_http_named_location
- ngx_http_output_filter
- ngx_http_send_header
- ngx_http_send_response
- ngx_http_set_content_type
- ngx_http_set_disable_symlinks
- ngx_http_set_etag
- ngx_http_set_exten
- ngx_http_subrequest
- ngx_http_test_content_type
- ngx_http_update_location_config
- ngx_http_weak_etag

Macros defined

- NGX_HTTP_REQUEST_BODY_FILE_CLEAN
- NGX_HTTP_REQUEST_BODY_FILE_OFF
- NGX_HTTP_REQUEST_BODY_FILE_ON

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      u_char     *name;
15      uint32_t   method;
16  } ngx_http_method_name_t;
17
18
19  #define NGX_HTTP_REQUEST_BODY_FILE_OFF    0
20  #define NGX_HTTP_REQUEST_BODY_FILE_ON     1
21  #define NGX_HTTP_REQUEST_BODY_FILE_CLEAN  2
22
23
24  static ngx_int_t ngx_http_core_find_location(ngx_http_request_t *r);
25  static ngx_int_t ngx_http_core_find_static_location(ngx_http_request_t *r,
26      ngx_http_location_tree_node_t *node);
27
28  static ngx_int_t ngx_http_core_preconfiguration(ngx_conf_t *cf);
29  static void *ngx_http_core_create_main_conf(ngx_conf_t *cf);
30  static char *ngx_http_core_init_main_conf(ngx_conf_t *cf, void *conf);
31  static void *ngx_http_core_create_srv_conf(ngx_conf_t *cf);
32  static char *ngx_http_core_merge_srv_conf(ngx_conf_t *cf,
33      void *parent, void *child);
34  static void *ngx_http_core_create_loc_conf(ngx_conf_t *cf);
35  static char *ngx_http_core_merge_loc_conf(ngx_conf_t *cf,
36      void *parent, void *child);
37
38  static char *ngx_http_core_server(ngx_conf_t *cf, ngx_command_t *cmd,
39      void *dummy);
40  static char *ngx_http_core_location(ngx_conf_t *cf, ngx_command_t *cmd,
41      void *dummy);
42  static ngx_int_t ngx_http_core_regex_location(ngx_conf_t *cf,
43      ngx_http_core_loc_conf_t *clcf, ngx_str_t *regex, ngx_uint_t caseless);
44
45  static char *ngx_http_core_types(ngx_conf_t *cf, ngx_command_t *cmd,
46      void *conf);
47  static char *ngx_http_core_type(ngx_conf_t *cf, ngx_command_t *dummy,
48      void *conf);
49
50  static char *ngx_http_core_listen(ngx_conf_t *cf, ngx_command_t *cmd,
51      void *conf);
52  static char *ngx_http_core_server_name(ngx_conf_t *cf, ngx_command_t *cmd,
53      void *conf);
54  static char *ngx_http_core_root(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
55  static char *ngx_http_core_limit_except(ngx_conf_t *cf, ngx_command_t *cmd,
56      void *conf);
57  static char *ngx_http_core_directio(ngx_conf_t *cf, ngx_command_t *cmd,
58      void *conf);
59  static char *ngx_http_core_error_page(ngx_conf_t *cf, ngx_command_t *cmd,
60      void *conf);
61  static char *ngx_http_core_try_files(ngx_conf_t *cf, ngx_command_t *cmd,
62      void *conf);
63  static char *ngx_http_core_open_file_cache(ngx_conf_t *cf, ngx_command_t *cmd,
64      void *conf);
65  static char *ngx_http_core_error_log(ngx_conf_t *cf, ngx_command_t *cmd,
66      void *conf);
67  static char *ngx_http_core_keepalive(ngx_conf_t *cf, ngx_command_t *cmd,
68      void *conf);
69  static char *ngx_http_core_internal(ngx_conf_t *cf, ngx_command_t *cmd,
70      void *conf);
71  static char *ngx_http_core_resolver(ngx_conf_t *cf, ngx_command_t *cmd,
72      void *conf);
73  #if (NGX_HTTP_GZIP)
```

```
74  static ngx_int_t ngx_http_gzip_accept_encoding(ngx_str_t *ae);
75  static ngx_uint_t ngx_http_gzip_quantity(u_char *p, u_char *last);
76  static char *ngx_http_gzip_disable(ngx_conf_t *cf, ngx_command_t *cmd,
77      void *conf);
78  #endif
79  static ngx_int_t ngx_http_get_forwarded_addr_internal(ngx_http_request_t *r,
80      ngx_addr_t *addr, u_char *xff, size_t xfflen, ngx_array_t *proxies,
81      int recursive);
82  #if (NGX_HAVE_OPENAT)
83  static char *ngx_http_disable_symlinks(ngx_conf_t *cf, ngx_command_t *cmd,
84      void *conf);
85  #endif
86
87  static char *ngx_http_core_lowat_check(ngx_conf_t *cf, void *post, void *data);
88  static char *ngx_http_core_pool_size(ngx_conf_t *cf, void *post, void *data);
89
90  static ngx_conf_post_t  ngx_http_core_lowat_post =
91      { ngx_http_core_lowat_check };
92
93  static ngx_conf_post_handler_pt  ngx_http_core_pool_size_p =
94      ngx_http_core_pool_size;
95
96  static ngx_conf_deprecated_t  ngx_conf_deprecated_optimize_server_names = {
97      ngx_conf_deprecated, "optimize_server_names", "server_name_in_redirect"
98  };
99
100  static ngx_conf_deprecated_t  ngx_conf_deprecated_open_file_cache_retest = {
101      ngx_conf_deprecated, "open_file_cache_retest", "open_file_cache_valid"
102  };
103
104  static ngx_conf_deprecated_t  ngx_conf_deprecated_satisfy_any = {
105      ngx_conf_deprecated, "satisfy_any", "satisfy"
106  };
107
108
109  static ngx_conf_enum_t  ngx_http_core_request_body_in_file[] = {
110      { ngx_string("off"), NGX_HTTP_REQUEST_BODY_FILE_OFF },
111      { ngx_string("on"), NGX_HTTP_REQUEST_BODY_FILE_ON },
112      { ngx_string("clean"), NGX_HTTP_REQUEST_BODY_FILE_CLEAN },
113      { ngx_null_string, 0 }
114  };
115
116
117  #if (NGX_HAVE_FILE_AIO)
118
119  static ngx_conf_enum_t  ngx_http_core_aio[] = {
120      { ngx_string("off"), NGX_HTTP_AIO_OFF  },
121      { ngx_string("on"), NGX_HTTP_AIO_ON },
122  #if (NGX_HAVE_AIO_SENDFILE)
123      { ngx_string("sendfile"), NGX_HTTP_AIO_SENDFILE },
124  #endif
125      { ngx_null_string, 0 }
126  };
127
128  #endif
129
130
131  static ngx_conf_enum_t  ngx_http_core_satisfy[] = {
132      { ngx_string("all"), NGX_HTTP_SATISFY_ALL },
133      { ngx_string("any"), NGX_HTTP_SATISFY_ANY },
134      { ngx_null_string, 0 }
135  };
136
137
138  static ngx_conf_enum_t  ngx_http_core_lingering_close[] = {
139      { ngx_string("off"), NGX_HTTP_LINGERING_OFF },
140      { ngx_string("on"), NGX_HTTP_LINGERING_ON },
141      { ngx_string("always"), NGX_HTTP_LINGERING_ALWAYS },
142      { ngx_null_string, 0 }
143  };
144
145
146  static ngx_conf_enum_t  ngx_http_core_if_modified_since[] = {
147      { ngx_string("off"), NGX_HTTP_IMS_OFF },
148      { ngx_string("exact"), NGX_HTTP_IMS_EXACT },
149      { ngx_string("before"), NGX_HTTP_IMS_BEFORE },
```

```
    { ngx_null_string, 0 }
};


static ngx_conf_bitmask_t  ngx_http_core_keepalive_disable[] = {
    { ngx_string("none"), NGX_HTTP_KEEPALIVE_DISABLE_NONE },
    { ngx_string("msie6"), NGX_HTTP_KEEPALIVE_DISABLE_MSIE6 },
    { ngx_string("safari"), NGX_HTTP_KEEPALIVE_DISABLE_SAFARI },
    { ngx_null_string, 0 }
};


static ngx_path_init_t  ngx_http_client_temp_path = {
    ngx_string(NGX_HTTP_CLIENT_TEMP_PATH), { 0, 0, 0 }
};


#if (NGX_HTTP_GZIP)

static ngx_conf_enum_t  ngx_http_gzip_http_version[] = {
    { ngx_string("1.0"), NGX_HTTP_VERSION_10 },
    { ngx_string("1.1"), NGX_HTTP_VERSION_11 },
    { ngx_null_string, 0 }
};


static ngx_conf_bitmask_t  ngx_http_gzip_proxied_mask[] = {
    { ngx_string("off"), NGX_HTTP_GZIP_PROXIED_OFF },
    { ngx_string("expired"), NGX_HTTP_GZIP_PROXIED_EXPIRED },
    { ngx_string("no-cache"), NGX_HTTP_GZIP_PROXIED_NO_CACHE },
    { ngx_string("no-store"), NGX_HTTP_GZIP_PROXIED_NO_STORE },
    { ngx_string("private"), NGX_HTTP_GZIP_PROXIED_PRIVATE },
    { ngx_string("no_last_modified"), NGX_HTTP_GZIP_PROXIED_NO_LM },
    { ngx_string("no_etag"), NGX_HTTP_GZIP_PROXIED_NO_ETAG },
    { ngx_string("auth"), NGX_HTTP_GZIP_PROXIED_AUTH },
    { ngx_string("any"), NGX_HTTP_GZIP_PROXIED_ANY },
    { ngx_null_string, 0 }
};


static ngx_str_t  ngx_http_gzip_no_cache = ngx_string("no-cache");
static ngx_str_t  ngx_http_gzip_no_store = ngx_string("no-store");
static ngx_str_t  ngx_http_gzip_private = ngx_string("private");

#endif


static ngx_command_t  ngx_http_core_commands[] = {

    { ngx_string("variables_hash_max_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_core_main_conf_t, variables_hash_max_size),
      NULL },

    { ngx_string("variables_hash_bucket_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_core_main_conf_t, variables_hash_bucket_size),
      NULL },

    { ngx_string("server_names_hash_max_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_core_main_conf_t, server_names_hash_max_size),
      NULL },

    { ngx_string("server_names_hash_bucket_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_core_main_conf_t, server_names_hash_bucket_size),
      NULL },
```

```
226
227     { ngx_string("server"),
228       NGX_HTTP_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
229       ngx_http_core_server,
230       0,
231       0,
232       NULL },
233
234     { ngx_string("connection_pool_size"),
235       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
236       ngx_conf_set_size_slot,
237       NGX_HTTP_SRV_CONF_OFFSET,
238       offsetof(ngx_http_core_srv_conf_t, connection_pool_size),
239       &ngx_http_core_pool_size_p },
240
241     { ngx_string("request_pool_size"),
242       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
243       ngx_conf_set_size_slot,
244       NGX_HTTP_SRV_CONF_OFFSET,
245       offsetof(ngx_http_core_srv_conf_t, request_pool_size),
246       &ngx_http_core_pool_size_p },
247
248     { ngx_string("client_header_timeout"),
249       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
250       ngx_conf_set_msec_slot,
251       NGX_HTTP_SRV_CONF_OFFSET,
252       offsetof(ngx_http_core_srv_conf_t, client_header_timeout),
253       NULL },
254
255     { ngx_string("client_header_buffer_size"),
256       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE1,
257       ngx_conf_set_size_slot,
258       NGX_HTTP_SRV_CONF_OFFSET,
259       offsetof(ngx_http_core_srv_conf_t, client_header_buffer_size),
260       NULL },
261
262     { ngx_string("large_client_header_buffers"),
263       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_TAKE2,
264       ngx_conf_set_bufs_slot,
265       NGX_HTTP_SRV_CONF_OFFSET,
266       offsetof(ngx_http_core_srv_conf_t, large_client_header_buffers),
267       NULL },
268
269     { ngx_string("optimize_server_names"),
270       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
271       ngx_conf_set_flag_slot,
272       NGX_HTTP_LOC_CONF_OFFSET,
273       offsetof(ngx_http_core_loc_conf_t, server_name_in_redirect),
274       &ngx_conf_deprecated_optimize_server_names },
275
276     { ngx_string("ignore_invalid_headers"),
277       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
278       ngx_conf_set_flag_slot,
279       NGX_HTTP_SRV_CONF_OFFSET,
280       offsetof(ngx_http_core_srv_conf_t, ignore_invalid_headers),
281       NULL },
282
283     { ngx_string("merge_slashes"),
284       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
285       ngx_conf_set_flag_slot,
286       NGX_HTTP_SRV_CONF_OFFSET,
287       offsetof(ngx_http_core_srv_conf_t, merge_slashes),
288       NULL },
289
290     { ngx_string("underscores_in_headers"),
291       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_CONF_FLAG,
292       ngx_conf_set_flag_slot,
293       NGX_HTTP_SRV_CONF_OFFSET,
294       offsetof(ngx_http_core_srv_conf_t, underscores_in_headers),
295       NULL },
296
297     { ngx_string("location"),
298       NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_BLOCK|NGX_CONF_TAKE12,
299       ngx_http_core_location,
300       NGX_HTTP_SRV_CONF_OFFSET,
301       0,
```

```
302         NULL },
303
304     { ngx_string("listen"),
305       NGX_HTTP_SRV_CONF|NGX_CONF_1MORE,
306       ngx_http_core_listen,
307       NGX_HTTP_SRV_CONF_OFFSET,
308       0,
309       NULL },
310
311     { ngx_string("server_name"),
312       NGX_HTTP_SRV_CONF|NGX_CONF_1MORE,
313       ngx_http_core_server_name,
314       NGX_HTTP_SRV_CONF_OFFSET,
315       0,
316       NULL },
317
318     { ngx_string("types_hash_max_size"),
319       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
320       ngx_conf_set_num_slot,
321       NGX_HTTP_LOC_CONF_OFFSET,
322       offsetof(ngx_http_core_loc_conf_t, types_hash_max_size),
323       NULL },
324
325     { ngx_string("types_hash_bucket_size"),
326       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
327       ngx_conf_set_num_slot,
328       NGX_HTTP_LOC_CONF_OFFSET,
329       offsetof(ngx_http_core_loc_conf_t, types_hash_bucket_size),
330       NULL },
331
332     { ngx_string("types"),
333       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF
334                                           |NGX_CONF_BLOCK|NGX_CONF_NOARGS,
335       ngx_http_core_types,
336       NGX_HTTP_LOC_CONF_OFFSET,
337       0,
338       NULL },
339
340     { ngx_string("default_type"),
341       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
342       ngx_conf_set_str_slot,
343       NGX_HTTP_LOC_CONF_OFFSET,
344       offsetof(ngx_http_core_loc_conf_t, default_type),
345       NULL },
346
347     { ngx_string("root"),
348       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
349                         |NGX_CONF_TAKE1,
350       ngx_http_core_root,
351       NGX_HTTP_LOC_CONF_OFFSET,
352       0,
353       NULL },
354
355     { ngx_string("alias"),
356       NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
357       ngx_http_core_root,
358       NGX_HTTP_LOC_CONF_OFFSET,
359       0,
360       NULL },
361
362     { ngx_string("limit_except"),
363       NGX_HTTP_LOC_CONF|NGX_CONF_BLOCK|NGX_CONF_1MORE,
364       ngx_http_core_limit_except,
365       NGX_HTTP_LOC_CONF_OFFSET,
366       0,
367       NULL },
368
369     { ngx_string("client_max_body_size"),
370       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
371       ngx_conf_set_off_slot,
372       NGX_HTTP_LOC_CONF_OFFSET,
373       offsetof(ngx_http_core_loc_conf_t, client_max_body_size),
374       NULL },
375
376     { ngx_string("client_body_buffer_size"),
377       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
```

```
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, client_body_buffer_size),
      NULL },

    { ngx_string("client_body_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, client_body_timeout),
      NULL },

    { ngx_string("client_body_temp_path"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1234,
      ngx_conf_set_path_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, client_body_temp_path),
      NULL },

    { ngx_string("client_body_in_file_only"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, client_body_in_file_only),
      &ngx_http_core_request_body_in_file },

    { ngx_string("client_body_in_single_buffer"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, client_body_in_single_buffer),
      NULL },

    { ngx_string("sendfile"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                        |NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, sendfile),
      NULL },

    { ngx_string("sendfile_max_chunk"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, sendfile_max_chunk),
      NULL },

#if (NGX_HAVE_FILE_AIO)

    { ngx_string("aio"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, aio),
      &ngx_http_core_aio },

#endif

    { ngx_string("read_ahead"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, read_ahead),
      NULL },

    { ngx_string("directio"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_core_directio,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("directio_alignment"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_off_slot,
```

```
454         NGX_HTTP_LOC_CONF_OFFSET,
455         offsetof(ngx_http_core_loc_conf_t, directio_alignment),
456         NULL },
457
458     { ngx_string("tcp_nopush"),
459         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
460         ngx_conf_set_flag_slot,
461         NGX_HTTP_LOC_CONF_OFFSET,
462         offsetof(ngx_http_core_loc_conf_t, tcp_nopush),
463         NULL },
464
465     { ngx_string("tcp_nodelay"),
466         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
467         ngx_conf_set_flag_slot,
468         NGX_HTTP_LOC_CONF_OFFSET,
469         offsetof(ngx_http_core_loc_conf_t, tcp_nodelay),
470         NULL },
471
472     { ngx_string("send_timeout"),
473         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
474         ngx_conf_set_msec_slot,
475         NGX_HTTP_LOC_CONF_OFFSET,
476         offsetof(ngx_http_core_loc_conf_t, send_timeout),
477         NULL },
478
479     { ngx_string("send_lowat"),
480         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
481         ngx_conf_set_size_slot,
482         NGX_HTTP_LOC_CONF_OFFSET,
483         offsetof(ngx_http_core_loc_conf_t, send_lowat),
484         &ngx_http_core_lowat_post },
485
486     { ngx_string("postpone_output"),
487         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
488         ngx_conf_set_size_slot,
489         NGX_HTTP_LOC_CONF_OFFSET,
490         offsetof(ngx_http_core_loc_conf_t, postpone_output),
491         NULL },
492
493     { ngx_string("limit_rate"),
494         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
495                           |NGX_CONF_TAKE1,
496         ngx_conf_set_size_slot,
497         NGX_HTTP_LOC_CONF_OFFSET,
498         offsetof(ngx_http_core_loc_conf_t, limit_rate),
499         NULL },
500
501     { ngx_string("limit_rate_after"),
502         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
503                           |NGX_CONF_TAKE1,
504         ngx_conf_set_size_slot,
505         NGX_HTTP_LOC_CONF_OFFSET,
506         offsetof(ngx_http_core_loc_conf_t, limit_rate_after),
507         NULL },
508
509     { ngx_string("keepalive_timeout"),
510         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
511         ngx_http_core_keepalive,
512         NGX_HTTP_LOC_CONF_OFFSET,
513         0,
514         NULL },
515
516     { ngx_string("keepalive_requests"),
517         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
518         ngx_conf_set_num_slot,
519         NGX_HTTP_LOC_CONF_OFFSET,
520         offsetof(ngx_http_core_loc_conf_t, keepalive_requests),
521         NULL },
522
523     { ngx_string("keepalive_disable"),
524         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
525         ngx_conf_set_bitmask_slot,
526         NGX_HTTP_LOC_CONF_OFFSET,
527         offsetof(ngx_http_core_loc_conf_t, keepalive_disable),
528         &ngx_http_core_keepalive_disable },
529
```

```
    { ngx_string("satisfy"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, satisfy),
      &ngx_http_core_satisfy },

    { ngx_string("satisfy_any"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, satisfy),
      &ngx_conf_deprecated_satisfy_any },

    { ngx_string("internal"),
      NGX_HTTP_LOC_CONF|NGX_CONF_NOARGS,
      ngx_http_core_internal,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("lingering_close"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, lingering_close),
      &ngx_http_core_lingering_close },

    { ngx_string("lingering_time"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, lingering_time),
      NULL },

    { ngx_string("lingering_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, lingering_timeout),
      NULL },

    { ngx_string("reset_timedout_connection"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, reset_timedout_connection),
      NULL },

    { ngx_string("server_name_in_redirect"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, server_name_in_redirect),
      NULL },

    { ngx_string("port_in_redirect"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, port_in_redirect),
      NULL },

    { ngx_string("msie_padding"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, msie_padding),
      NULL },

    { ngx_string("msie_refresh"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, msie_refresh),
      NULL },
```

```
606
607     { ngx_string("log_not_found"),
608       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
609       ngx_conf_set_flag_slot,
610       NGX_HTTP_LOC_CONF_OFFSET,
611       offsetof(ngx_http_core_loc_conf_t, log_not_found),
612       NULL },
613
614     { ngx_string("log_subrequest"),
615       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
616       ngx_conf_set_flag_slot,
617       NGX_HTTP_LOC_CONF_OFFSET,
618       offsetof(ngx_http_core_loc_conf_t, log_subrequest),
619       NULL },
620
621     { ngx_string("recursive_error_pages"),
622       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
623       ngx_conf_set_flag_slot,
624       NGX_HTTP_LOC_CONF_OFFSET,
625       offsetof(ngx_http_core_loc_conf_t, recursive_error_pages),
626       NULL },
627
628     { ngx_string("server_tokens"),
629       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
630       ngx_conf_set_flag_slot,
631       NGX_HTTP_LOC_CONF_OFFSET,
632       offsetof(ngx_http_core_loc_conf_t, server_tokens),
633       NULL },
634
635     { ngx_string("if_modified_since"),
636       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
637       ngx_conf_set_enum_slot,
638       NGX_HTTP_LOC_CONF_OFFSET,
639       offsetof(ngx_http_core_loc_conf_t, if_modified_since),
640       &ngx_http_core_if_modified_since },
641
642     { ngx_string("max_ranges"),
643       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
644       ngx_conf_set_num_slot,
645       NGX_HTTP_LOC_CONF_OFFSET,
646       offsetof(ngx_http_core_loc_conf_t, max_ranges),
647       NULL },
648
649     { ngx_string("chunked_transfer_encoding"),
650       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
651       ngx_conf_set_flag_slot,
652       NGX_HTTP_LOC_CONF_OFFSET,
653       offsetof(ngx_http_core_loc_conf_t, chunked_transfer_encoding),
654       NULL },
655
656     { ngx_string("etag"),
657       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
658       ngx_conf_set_flag_slot,
659       NGX_HTTP_LOC_CONF_OFFSET,
660       offsetof(ngx_http_core_loc_conf_t, etag),
661       NULL },
662
663     { ngx_string("error_page"),
664       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
665                         |NGX_CONF_2MORE,
666       ngx_http_core_error_page,
667       NGX_HTTP_LOC_CONF_OFFSET,
668       0,
669       NULL },
670
671     { ngx_string("try_files"),
672       NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_2MORE,
673       ngx_http_core_try_files,
674       NGX_HTTP_LOC_CONF_OFFSET,
675       0,
676       NULL },
677
678     { ngx_string("post_action"),
679       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
680                         |NGX_CONF_TAKE1,
681       ngx_conf_set_str_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, post_action),
      NULL },

    { ngx_string("error_log"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_core_error_log,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("open_file_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
      ngx_http_core_open_file_cache,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, open_file_cache),
      NULL },

    { ngx_string("open_file_cache_valid"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_sec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, open_file_cache_valid),
      NULL },

    { ngx_string("open_file_cache_retest"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_sec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, open_file_cache_valid),
      &ngx_conf_deprecated_open_file_cache_retest },

    { ngx_string("open_file_cache_min_uses"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, open_file_cache_min_uses),
      NULL },

    { ngx_string("open_file_cache_errors"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, open_file_cache_errors),
      NULL },

    { ngx_string("open_file_cache_events"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, open_file_cache_events),
      NULL },

    { ngx_string("resolver"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_core_resolver,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("resolver_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, resolver_timeout),
      NULL },

#if (NGX_HTTP_GZIP)

    { ngx_string("gzip_vary"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, gzip_vary),
      NULL },
```

```
    { ngx_string("gzip_http_version"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, gzip_http_version),
      &ngx_http_gzip_http_version },

    { ngx_string("gzip_proxied"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_core_loc_conf_t, gzip_proxied),
      &ngx_http_gzip_proxied_mask },

    { ngx_string("gzip_disable"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_gzip_disable,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

#endif

#if (NGX_HAVE_OPENAT)

    { ngx_string("disable_symlinks"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
      ngx_http_disable_symlinks,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

#endif

      ngx_null_command
};


static ngx_http_module_t  ngx_http_core_module_ctx = {
    ngx_http_core_preconfiguration,        /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_core_create_main_conf,        /* create main configuration */
    ngx_http_core_init_main_conf,          /* init main configuration */

    ngx_http_core_create_srv_conf,         /* create server configuration */
    ngx_http_core_merge_srv_conf,          /* merge server configuration */

    ngx_http_core_create_loc_conf,         /* create location configuration */
    ngx_http_core_merge_loc_conf           /* merge location configuration */
};


ngx_module_t  ngx_http_core_module = {
    NGX_MODULE_V1,
    &ngx_http_core_module_ctx,             /* module context */
    ngx_http_core_commands,                /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


ngx_str_t  ngx_http_core_get_method = { 3, (u_char *) "GET " };


void
ngx_http_handler(ngx_http_request_t *r)
{
    ngx_http_core_main_conf_t  *cmcf;
```

```
834
835     r->connection->log->action = NULL;
836
837     r->connection->unexpected_eof = 0;
838
839     if (!r->internal) {
840         switch (r->headers_in.connection_type) {
841         case 0:
842             r->keepalive = (r->http_version > NGX_HTTP_VERSION_10);
843             break;
844
845         case NGX_HTTP_CONNECTION_CLOSE:
846             r->keepalive = 0;
847             break;
848
849         case NGX_HTTP_CONNECTION_KEEP_ALIVE:
850             r->keepalive = 1;
851             break;
852         }
853
854         r->lingering_close = (r->headers_in.content_length_n > 0
855                               || r->headers_in.chunked);
856         r->phase_handler = 0;
857
858     } else {
859         cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
860         r->phase_handler = cmcf->phase_engine.server_rewrite_index;
861     }
862
863     r->valid_location = 1;
864 #if (NGX_HTTP_GZIP)
865     r->gzip_tested = 0;
866     r->gzip_ok = 0;
867     r->gzip_vary = 0;
868 #endif
869
870     r->write_event_handler = ngx_http_core_run_phases;
871     ngx_http_core_run_phases(r);
872 }
873
874
875 void
876 ngx_http_core_run_phases(ngx_http_request_t *r)
877 {
878     ngx_int_t                   rc;
879     ngx_http_phase_handler_t   *ph;
880     ngx_http_core_main_conf_t  *cmcf;
881
882     cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
883
884     ph = cmcf->phase_engine.handlers;
885
886     while (ph[r->phase_handler].checker) {
887
888         rc = ph[r->phase_handler].checker(r, &ph[r->phase_handler]);
889
890         if (rc == NGX_OK) {
891             return;
892         }
893     }
894 }
895
896
897 ngx_int_t
898 ngx_http_core_generic_phase(ngx_http_request_t *r, ngx_http_phase_handler_t *ph)
899 {
900     ngx_int_t  rc;
901
902     /*
903      * generic phase checker,
904      * used by the post read and pre-access phases
905      */
906
907     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
908                    "generic phase: %ui", r->phase_handler);
909
```

```
    rc = ph->handler(r);

    if (rc == NGX_OK) {
        r->phase_handler = ph->next;
        return NGX_AGAIN;
    }

    if (rc == NGX_DECLINED) {
        r->phase_handler++;
        return NGX_AGAIN;
    }

    if (rc == NGX_AGAIN || rc == NGX_DONE) {
        return NGX_OK;
    }

    /* rc == NGX_ERROR || rc == NGX_HTTP_...  */

    ngx_http_finalize_request(r, rc);

    return NGX_OK;
}


ngx_int_t
ngx_http_core_rewrite_phase(ngx_http_request_t *r, ngx_http_phase_handler_t *ph)
{
    ngx_int_t  rc;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "rewrite phase: %ui", r->phase_handler);

    rc = ph->handler(r);

    if (rc == NGX_DECLINED) {
        r->phase_handler++;
        return NGX_AGAIN;
    }

    if (rc == NGX_DONE) {
        return NGX_OK;
    }

    /* NGX_OK, NGX_AGAIN, NGX_ERROR, NGX_HTTP_...  */

    ngx_http_finalize_request(r, rc);

    return NGX_OK;
}


ngx_int_t
ngx_http_core_find_config_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph)
{
    u_char                    *p;
    size_t                     len;
    ngx_int_t                  rc;
    ngx_http_core_loc_conf_t  *clcf;

    r->content_handler = NULL;
    r->uri_changed = 0;

    rc = ngx_http_core_find_location(r);

    if (rc == NGX_ERROR) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return NGX_OK;
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (!r->internal && clcf->internal) {
        ngx_http_finalize_request(r, NGX_HTTP_NOT_FOUND);
        return NGX_OK;
    }
```

```
986
987     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
988                    "using configuration \"%s%V\"",
989                    (clcf->noname ? "*" : (clcf->exact_match ? "=" : "")),
990                    &clcf->name);
991
992     ngx_http_update_location_config(r);
993
994     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
995                    "http cl:%O max:%O",
996                    r->headers_in.content_length_n, clcf->client_max_body_size);
997
998     if (r->headers_in.content_length_n != -1
999         && !r->discard_body
1000         && clcf->client_max_body_size
1001         && clcf->client_max_body_size < r->headers_in.content_length_n)
1002     {
1003         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1004                       "client intended to send too large body: %O bytes",
1005                       r->headers_in.content_length_n);
1006
1007         r->expect_tested = 1;
1008         (void) ngx_http_discard_request_body(r);
1009         ngx_http_finalize_request(r, NGX_HTTP_REQUEST_ENTITY_TOO_LARGE);
1010         return NGX_OK;
1011     }
1012
1013     if (rc == NGX_DONE) {
1014         ngx_http_clear_location(r);
1015
1016         r->headers_out.location = ngx_list_push(&r->headers_out.headers);
1017         if (r->headers_out.location == NULL) {
1018             ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
1019             return NGX_OK;
1020         }
1021
1022         /*
1023          * we do not need to set the r->headers_out.location->hash and
1024          * r->headers_out.location->key fields
1025          */
1026
1027         if (r->args.len == 0) {
1028             r->headers_out.location->value = clcf->name;
1029
1030         } else {
1031             len = clcf->name.len + 1 + r->args.len;
1032             p = ngx_pnalloc(r->pool, len);
1033
1034             if (p == NULL) {
1035                 ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
1036                 return NGX_OK;
1037             }
1038
1039             r->headers_out.location->value.len = len;
1040             r->headers_out.location->value.data = p;
1041
1042             p = ngx_cpymem(p, clcf->name.data, clcf->name.len);
1043             *p++ = '?';
1044             ngx_memcpy(p, r->args.data, r->args.len);
1045         }
1046
1047         ngx_http_finalize_request(r, NGX_HTTP_MOVED_PERMANENTLY);
1048         return NGX_OK;
1049     }
1050
1051     r->phase_handler++;
1052     return NGX_AGAIN;
1053 }
1054
1055
1056 ngx_int_t
1057 ngx_http_core_post_rewrite_phase(ngx_http_request_t *r,
1058     ngx_http_phase_handler_t *ph)
1059 {
1060     ngx_http_core_srv_conf_t  *cscf;
1061
```

```
1062      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1063                     "post rewrite phase: %ui", r->phase_handler);
1064
1065      if (!r->uri_changed) {
1066          r->phase_handler++;
1067          return NGX_AGAIN;
1068      }
1069
1070      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1071                     "uri changes: %d", r->uri_changes);
1072
1073      /*
1074       * gcc before 3.3 compiles the broken code for
1075       *     if (r->uri_changes-- == 0)
1076       * if the r->uri_changes is defined as
1077       *     unsigned  uri_changes:4
1078       */
1079
1080      r->uri_changes--;
1081
1082      if (r->uri_changes == 0) {
1083          ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1084                        "rewrite or internal redirection cycle "
1085                        "while processing \"%V\"", &r->uri);
1086
1087          ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
1088          return NGX_OK;
1089      }
1090
1091      r->phase_handler = ph->next;
1092
1093      cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
1094      r->loc_conf = cscf->ctx->loc_conf;
1095
1096      return NGX_AGAIN;
1097  }
1098
1099
1100  ngx_int_t
1101  ngx_http_core_access_phase(ngx_http_request_t *r, ngx_http_phase_handler_t *ph)
1102  {
1103      ngx_int_t                  rc;
1104      ngx_http_core_loc_conf_t  *clcf;
1105
1106      if (r != r->main) {
1107          r->phase_handler = ph->next;
1108          return NGX_AGAIN;
1109      }
1110
1111      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1112                     "access phase: %ui", r->phase_handler);
1113
1114      rc = ph->handler(r);
1115
1116      if (rc == NGX_DECLINED) {
1117          r->phase_handler++;
1118          return NGX_AGAIN;
1119      }
1120
1121      if (rc == NGX_AGAIN || rc == NGX_DONE) {
1122          return NGX_OK;
1123      }
1124
1125      clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
1126
1127      if (clcf->satisfy == NGX_HTTP_SATISFY_ALL) {
1128
1129          if (rc == NGX_OK) {
1130              r->phase_handler++;
1131              return NGX_AGAIN;
1132          }
1133
1134      } else {
1135          if (rc == NGX_OK) {
1136              r->access_code = 0;
1137
```

```
            if (r->headers_out.www_authenticate) {
                r->headers_out.www_authenticate->hash = 0;
            }

            r->phase_handler = ph->next;
            return NGX_AGAIN;
        }

        if (rc == NGX_HTTP_FORBIDDEN || rc == NGX_HTTP_UNAUTHORIZED) {
            if (r->access_code != NGX_HTTP_UNAUTHORIZED) {
                r->access_code = rc;
            }

            r->phase_handler++;
            return NGX_AGAIN;
        }
    }

    /* rc == NGX_ERROR || rc == NGX_HTTP_...  */

    ngx_http_finalize_request(r, rc);
    return NGX_OK;
}


ngx_int_t
ngx_http_core_post_access_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph)
{
    ngx_int_t  access_code;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "post access phase: %ui", r->phase_handler);

    access_code = r->access_code;

    if (access_code) {
        if (access_code == NGX_HTTP_FORBIDDEN) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "access forbidden by rule");
        }

        r->access_code = 0;
        ngx_http_finalize_request(r, access_code);
        return NGX_OK;
    }

    r->phase_handler++;
    return NGX_AGAIN;
}


ngx_int_t
ngx_http_core_try_files_phase(ngx_http_request_t *r,
    ngx_http_phase_handler_t *ph)
{
    size_t                     len, root, alias, reserve, allocated;
    u_char                    *p, *name;
    ngx_str_t                  path, args;
    ngx_uint_t                 test_dir;
    ngx_http_try_file_t       *tf;
    ngx_open_file_info_t       of;
    ngx_http_script_code_pt    code;
    ngx_http_script_engine_t   e;
    ngx_http_core_loc_conf_t  *clcf;
    ngx_http_script_len_code_pt  lcode;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "try files phase: %ui", r->phase_handler);

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (clcf->try_files == NULL) {
        r->phase_handler++;
        return NGX_AGAIN;
    }
```

```
1214
1215        allocated = 0;
1216        root = 0;
1217        name = NULL;
1218        /* suppress MSVC warning */
1219        path.data = NULL;
1220
1221        tf = clcf->try_files;
1222
1223        alias = clcf->alias;
1224
1225        for ( ;; ) {
1226
1227            if (tf->lengths) {
1228                ngx_memzero(&e, sizeof(ngx_http_script_engine_t));
1229
1230                e.ip = tf->lengths->elts;
1231                e.request = r;
1232
1233                /* 1 is for terminating '\0' as in static names */
1234                len = 1;
1235
1236                while (*(uintptr_t *) e.ip) {
1237                    lcode = *(ngx_http_script_len_code_pt *) e.ip;
1238                    len += lcode(&e);
1239                }
1240
1241            } else {
1242                len = tf->name.len;
1243            }
1244
1245            if (!alias) {
1246                reserve = len > r->uri.len ? len - r->uri.len : 0;
1247
1248            } else if (alias == NGX_MAX_SIZE_T_VALUE) {
1249                reserve = len;
1250
1251            } else {
1252                reserve = len > r->uri.len - alias ? len - (r->uri.len - alias) : 0;
1253            }
1254
1255            if (reserve > allocated || !allocated) {
1256
1257                /* 16 bytes are preallocation */
1258                allocated = reserve + 16;
1259
1260                if (ngx_http_map_uri_to_path(r, &path, &root, allocated) == NULL) {
1261                    ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
1262                    return NGX_OK;
1263                }
1264
1265                name = path.data + root;
1266             }
1267
1268            if (tf->values == NULL) {
1269
1270                /* tf->name.len includes the terminating '\0' */
1271
1272                ngx_memcpy(name, tf->name.data, tf->name.len);
1273
1274                path.len = (name + tf->name.len - 1) - path.data;
1275
1276            } else {
1277                e.ip = tf->values->elts;
1278                e.pos = name;
1279                e.flushed = 1;
1280
1281                while (*(uintptr_t *) e.ip) {
1282                    code = *(ngx_http_script_code_pt *) e.ip;
1283                    code((ngx_http_script_engine_t *) &e);
1284                }
1285
1286                path.len = e.pos - path.data;
1287
1288                *e.pos = '\0';
1289
```

```
            if (alias && ngx_strncmp(name, clcf->name.data, alias) == 0) {
                ngx_memmove(name, name + alias, len - alias);
                path.len -= alias;
            }
        }

        test_dir = tf->test_dir;

        tf++;

        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "trying to use %s: \"%s\" \"%s\"",
                       test_dir ? "dir" : "file", name, path.data);

        if (tf->lengths == NULL && tf->name.len == 0) {

            if (tf->code) {
                ngx_http_finalize_request(r, tf->code);
                return NGX_OK;
            }

            path.len -= root;
            path.data += root;

            if (path.data[0] == '@') {
                (void) ngx_http_named_location(r, &path);

            } else {
                ngx_http_split_args(r, &path, &args);

                (void) ngx_http_internal_redirect(r, &path, &args);
            }

            ngx_http_finalize_request(r, NGX_DONE);
            return NGX_OK;
        }

        ngx_memzero(&of, sizeof(ngx_open_file_info_t));

        of.read_ahead = clcf->read_ahead;
        of.directio = clcf->directio;
        of.valid = clcf->open_file_cache_valid;
        of.min_uses = clcf->open_file_cache_min_uses;
        of.test_only = 1;
        of.errors = clcf->open_file_cache_errors;
        of.events = clcf->open_file_cache_events;

        if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
            ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return NGX_OK;
        }

        if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
            != NGX_OK)
        {
            if (of.err != NGX_ENOENT
                && of.err != NGX_ENOTDIR
                && of.err != NGX_ENAMETOOLONG)
            {
                ngx_log_error(NGX_LOG_CRIT, r->connection->log, of.err,
                              "%s \"%s\" failed", of.failed, path.data);
            }

            continue;
        }

        if (of.is_dir != test_dir) {
            continue;
        }

        path.len -= root;
        path.data += root;

        if (!alias) {
            r->uri = path;

```

```
1366        } else if (alias == NGX_MAX_SIZE_T_VALUE) {
1367            if (!test_dir) {
1368                r->uri = path;
1369                r->add_uri_to_alias = 1;
1370            }
1371
1372        } else {
1373            r->uri.len = alias + path.len;
1374            r->uri.data = ngx_pnalloc(r->pool, r->uri.len);
1375            if (r->uri.data == NULL) {
1376                ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
1377                return NGX_OK;
1378            }
1379
1380            p = ngx_copy(r->uri.data, clcf->name.data, alias);
1381            ngx_memcpy(p, name, path.len);
1382        }
1383
1384        ngx_http_set_exten(r);
1385
1386        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1387                       "try file uri: \"%V\"", &r->uri);
1388
1389        r->phase_handler++;
1390        return NGX_AGAIN;
1391    }
1392
1393    /* not reached */
1394 }
1395
1396
1397 ngx_int_t
1398 ngx_http_core_content_phase(ngx_http_request_t *r,
1399     ngx_http_phase_handler_t *ph)
1400 {
1401     size_t     root;
1402     ngx_int_t  rc;
1403     ngx_str_t  path;
1404
1405     if (r->content_handler) {
1406         r->write_event_handler = ngx_http_request_empty_handler;
1407         ngx_http_finalize_request(r, r->content_handler(r));
1408         return NGX_OK;
1409     }
1410
1411     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1412                    "content phase: %ui", r->phase_handler);
1413
1414     rc = ph->handler(r);
1415
1416     if (rc != NGX_DECLINED) {
1417         ngx_http_finalize_request(r, rc);
1418         return NGX_OK;
1419     }
1420
1421     /* rc == NGX_DECLINED */
1422
1423     ph++;
1424
1425     if (ph->checker) {
1426         r->phase_handler++;
1427         return NGX_AGAIN;
1428     }
1429
1430     /* no content handler was found */
1431
1432     if (r->uri.data[r->uri.len - 1] == '/') {
1433
1434         if (ngx_http_map_uri_to_path(r, &path, &root, 0) != NULL) {
1435             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1436                           "directory index of \"%s\" is forbidden", path.data);
1437         }
1438
1439         ngx_http_finalize_request(r, NGX_HTTP_FORBIDDEN);
1440         return NGX_OK;
1441     }
```

```
1442
1443        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0, "no handler found");
1444
1445        ngx_http_finalize_request(r, NGX_HTTP_NOT_FOUND);
1446        return NGX_OK;
1447    }
1448
1449
1450    void
1451    ngx_http_update_location_config(ngx_http_request_t *r)
1452    {
1453        ngx_http_core_loc_conf_t  *clcf;
1454
1455        clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
1456
1457        if (r->method & clcf->limit_except) {
1458            r->loc_conf = clcf->limit_except_loc_conf;
1459            clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
1460        }
1461
1462        if (r == r->main) {
1463            ngx_http_set_connection_log(r->connection, clcf->error_log);
1464        }
1465
1466        if ((ngx_io.flags & NGX_IO_SENDFILE) && clcf->sendfile) {
1467            r->connection->sendfile = 1;
1468
1469        } else {
1470            r->connection->sendfile = 0;
1471        }
1472
1473        if (clcf->client_body_in_file_only) {
1474            r->request_body_in_file_only = 1;
1475            r->request_body_in_persistent_file = 1;
1476            r->request_body_in_clean_file =
1477                clcf->client_body_in_file_only == NGX_HTTP_REQUEST_BODY_FILE_CLEAN;
1478            r->request_body_file_log_level = NGX_LOG_NOTICE;
1479
1480        } else {
1481            r->request_body_file_log_level = NGX_LOG_WARN;
1482        }
1483
1484        r->request_body_in_single_buf = clcf->client_body_in_single_buffer;
1485
1486        if (r->keepalive) {
1487            if (clcf->keepalive_timeout == 0) {
1488                r->keepalive = 0;
1489
1490            } else if (r->connection->requests >= clcf->keepalive_requests) {
1491                r->keepalive = 0;
1492
1493            } else if (r->headers_in.msie6
1494                       && r->method == NGX_HTTP_POST
1495                       && (clcf->keepalive_disable
1496                           & NGX_HTTP_KEEPALIVE_DISABLE_MSIE6))
1497            {
1498                /*
1499                 * MSIE may wait for some time if an response for
1500                 * a POST request was sent over a keepalive connection
1501                 */
1502                r->keepalive = 0;
1503
1504            } else if (r->headers_in.safari
1505                       && (clcf->keepalive_disable
1506                           & NGX_HTTP_KEEPALIVE_DISABLE_SAFARI))
1507            {
1508                /*
1509                 * Safari may send a POST request to a closed keepalive
1510                 * connection and may stall for some time, see
1511                 *     https://bugs.webkit.org/show_bug.cgi?id=5760
1512                 */
1513                r->keepalive = 0;
1514            }
1515        }
1516
1517        if (!clcf->tcp_nopush) {
```

```
        /* disable TCP_NOPUSH/TCP_CORK use */
        r->connection->tcp_nopush = NGX_TCP_NOPUSH_DISABLED;
    }

    if (r->limit_rate == 0) {
        r->limit_rate = clcf->limit_rate;
    }

    if (clcf->handler) {
        r->content_handler = clcf->handler;
    }
}


/*
* NGX_OK       - exact or regex match
* NGX_DONE     - auto redirect
* NGX_AGAIN    - inclusive match
* NGX_ERROR    - regex error
* NGX_DECLINED - no match
*/

static ngx_int_t
ngx_http_core_find_location(ngx_http_request_t *r)
{
    ngx_int_t                  rc;
    ngx_http_core_loc_conf_t  *pclcf;
#if (NGX_PCRE)
    ngx_int_t                  n;
    ngx_uint_t                 noregex;
    ngx_http_core_loc_conf_t  *clcf, **clcfp;

    noregex = 0;
#endif

    pclcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    rc = ngx_http_core_find_static_location(r, pclcf->static_locations);

    if (rc == NGX_AGAIN) {

#if (NGX_PCRE)
        clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

        noregex = clcf->noregex;
#endif

        /* look up nested locations */

        rc = ngx_http_core_find_location(r);
    }

    if (rc == NGX_OK || rc == NGX_DONE) {
        return rc;
    }

    /* rc == NGX_DECLINED or rc == NGX_AGAIN in nested location */

#if (NGX_PCRE)

    if (noregex == 0 && pclcf->regex_locations) {

        for (clcfp = pclcf->regex_locations; *clcfp; clcfp++) {

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "test location: ~ \"%V\"", &(*clcfp)->name);

            n = ngx_http_regex_exec(r, (*clcfp)->regex, &r->uri);

            if (n == NGX_OK) {
                r->loc_conf = (*clcfp)->loc_conf;

                /* look up nested locations */

                rc = ngx_http_core_find_location(r);

```

```
                return (rc == NGX_ERROR) ? rc : NGX_OK;
            }

            if (n == NGX_DECLINED) {
                continue;
            }

            return NGX_ERROR;
        }
    }
#endif

    return rc;
}


/*
* NGX_OK       - exact match
* NGX_DONE     - auto redirect
* NGX_AGAIN    - inclusive match
* NGX_DECLINED - no match
*/

static ngx_int_t
ngx_http_core_find_static_location(ngx_http_request_t *r,
    ngx_http_location_tree_node_t *node)
{
    u_char     *uri;
    size_t      len, n;
    ngx_int_t   rc, rv;

    len = r->uri.len;
    uri = r->uri.data;

    rv = NGX_DECLINED;

    for ( ;; ) {

        if (node == NULL) {
            return rv;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "test location: \"%*s\"", node->len, node->name);

        n = (len <= (size_t) node->len) ? len : node->len;

        rc = ngx_filename_cmp(uri, node->name, n);

        if (rc != 0) {
            node = (rc < 0) ? node->left : node->right;

            continue;
        }

        if (len > (size_t) node->len) {

            if (node->inclusive) {

                r->loc_conf = node->inclusive->loc_conf;
                rv = NGX_AGAIN;

                node = node->tree;
                uri += n;
                len -= n;

                continue;
            }

            /* exact only */

            node = node->right;

            continue;
        }

```

```
        if (len == (size_t) node->len) {

            if (node->exact) {
                r->loc_conf = node->exact->loc_conf;
                return NGX_OK;

            } else {
                r->loc_conf = node->inclusive->loc_conf;
                return NGX_AGAIN;
            }
        }

        /* len < node->len */

        if (len + 1 == (size_t) node->len && node->auto_redirect) {

            r->loc_conf = (node->exact) ? node->exact->loc_conf:
                                          node->inclusive->loc_conf;
            rv = NGX_DONE;
        }

        node = node->left;
    }
}


void *
ngx_http_test_content_type(ngx_http_request_t *r, ngx_hash_t *types_hash)
{
    u_char      c, *lowcase;
    size_t      len;
    ngx_uint_t  i, hash;

    if (types_hash->size == 0) {
        return (void *) 4;
    }

    if (r->headers_out.content_type.len == 0) {
        return NULL;
    }

    len = r->headers_out.content_type_len;

    if (r->headers_out.content_type_lowcase == NULL) {

        lowcase = ngx_pnalloc(r->pool, len);
        if (lowcase == NULL) {
            return NULL;
        }

        r->headers_out.content_type_lowcase = lowcase;

        hash = 0;

        for (i = 0; i < len; i++) {
            c = ngx_tolower(r->headers_out.content_type.data[i]);
            hash = ngx_hash(hash, c);
            lowcase[i] = c;
        }

        r->headers_out.content_type_hash = hash;
    }

    return ngx_hash_find(types_hash, r->headers_out.content_type_hash,
                         r->headers_out.content_type_lowcase, len);
}


ngx_int_t
ngx_http_set_content_type(ngx_http_request_t *r)
{
    u_char                     c, *exten;
    ngx_str_t                 *type;
    ngx_uint_t                 i, hash;
    ngx_http_core_loc_conf_t  *clcf;

```

```
    if (r->headers_out.content_type.len) {
        return NGX_OK;
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (r->exten.len) {

        hash = 0;

        for (i = 0; i < r->exten.len; i++) {
            c = r->exten.data[i];

            if (c >= 'A' && c <= 'Z') {

                exten = ngx_pnalloc(r->pool, r->exten.len);
                if (exten == NULL) {
                    return NGX_ERROR;
                }

                hash = ngx_hash_strlow(exten, r->exten.data, r->exten.len);

                r->exten.data = exten;

                break;
            }

            hash = ngx_hash(hash, c);
        }

        type = ngx_hash_find(&clcf->types_hash, hash,
                             r->exten.data, r->exten.len);

        if (type) {
            r->headers_out.content_type_len = type->len;
            r->headers_out.content_type = *type;

            return NGX_OK;
        }
    }

    r->headers_out.content_type_len = clcf->default_type.len;
    r->headers_out.content_type = clcf->default_type;

    return NGX_OK;
}


void
ngx_http_set_exten(ngx_http_request_t *r)
{
    ngx_int_t  i;

    ngx_str_null(&r->exten);

    for (i = r->uri.len - 1; i > 1; i--) {
        if (r->uri.data[i] == '.' && r->uri.data[i - 1] != '/') {

            r->exten.len = r->uri.len - i - 1;
            r->exten.data = &r->uri.data[i + 1];

            return;

        } else if (r->uri.data[i] == '/') {
            return;
        }
    }

    return;
}


ngx_int_t
ngx_http_set_etag(ngx_http_request_t *r)
{
    ngx_table_elt_t           *etag;
```

```
    ngx_http_core_loc_conf_t  *clcf;

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (!clcf->etag) {
        return NGX_OK;
    }

    etag = ngx_list_push(&r->headers_out.headers);
    if (etag == NULL) {
        return NGX_ERROR;
    }

    etag->hash = 1;
    ngx_str_set(&etag->key, "ETag");

    etag->value.data = ngx_pnalloc(r->pool, NGX_OFF_T_LEN + NGX_TIME_T_LEN + 3);
    if (etag->value.data == NULL) {
        etag->hash = 0;
        return NGX_ERROR;
    }

    etag->value.len = ngx_sprintf(etag->value.data, "\"%xT-%xO\"",
                                  r->headers_out.last_modified_time,
                                  r->headers_out.content_length_n)
                      - etag->value.data;

    r->headers_out.etag = etag;

    return NGX_OK;
}


void
ngx_http_weak_etag(ngx_http_request_t *r)
{
    size_t            len;
    u_char           *p;
    ngx_table_elt_t  *etag;

    etag = r->headers_out.etag;

    if (etag == NULL) {
        return;
    }

    if (etag->value.len > 2
        && etag->value.data[0] == 'W'
        && etag->value.data[1] == '/')
    {
        return;
    }

    if (etag->value.len < 1 || etag->value.data[0] != '"') {
        r->headers_out.etag->hash = 0;
        r->headers_out.etag = NULL;
        return;
    }

    p = ngx_pnalloc(r->pool, etag->value.len + 2);
    if (p == NULL) {
        r->headers_out.etag->hash = 0;
        r->headers_out.etag = NULL;
        return;
    }

    len = ngx_sprintf(p, "W/%V", &etag->value) - p;

    etag->value.data = p;
    etag->value.len = len;
}


ngx_int_t
ngx_http_send_response(ngx_http_request_t *r, ngx_uint_t status,
    ngx_str_t *ct, ngx_http_complex_value_t *cv)
```

```
{
    ngx_int_t     rc;
    ngx_str_t     val;
    ngx_buf_t    *b;
    ngx_chain_t   out;

    if (ngx_http_discard_request_body(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    r->headers_out.status = status;

    if (ngx_http_complex_value(r, cv, &val) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (status == NGX_HTTP_MOVED_PERMANENTLY
        || status == NGX_HTTP_MOVED_TEMPORARILY
        || status == NGX_HTTP_SEE_OTHER
        || status == NGX_HTTP_TEMPORARY_REDIRECT)
    {
        ngx_http_clear_location(r);

        r->headers_out.location = ngx_list_push(&r->headers_out.headers);
        if (r->headers_out.location == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        r->headers_out.location->hash = 1;
        ngx_str_set(&r->headers_out.location->key, "Location");
        r->headers_out.location->value = val;

        return status;
    }

    r->headers_out.content_length_n = val.len;

    if (ct) {
        r->headers_out.content_type_len = ct->len;
        r->headers_out.content_type = *ct;

    } else {
        if (ngx_http_set_content_type(r) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }
    }

    if (r->method == NGX_HTTP_HEAD || (r != r->main && val.len == 0)) {
        return ngx_http_send_header(r);
    }

    b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
    if (b == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    b->pos = val.data;
    b->last = val.data + val.len;
    b->memory = val.len ? 1 : 0;
    b->last_buf = (r == r->main) ? 1 : 0;
    b->last_in_chain = 1;

    out.buf = b;
    out.next = NULL;

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
        return rc;
    }

    return ngx_http_output_filter(r, &out);
}


ngx_int_t
```

```
ngx_http_send_header(ngx_http_request_t *r)
{
    if (r->post_action) {
        return NGX_OK;
    }

    if (r->header_sent) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      "header already sent");
        return NGX_ERROR;
    }

    if (r->err_status) {
        r->headers_out.status = r->err_status;
        r->headers_out.status_line.len = 0;
    }

    return ngx_http_top_header_filter(r);
}


ngx_int_t
ngx_http_output_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    ngx_int_t          rc;
    ngx_connection_t  *c;

    c = r->connection;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http output filter \"%V?%V\"", &r->uri, &r->args);

    rc = ngx_http_top_body_filter(r, in);

    if (rc == NGX_ERROR) {
        /* NGX_ERROR may be returned by any filter */
        c->error = 1;
    }

    return rc;
}


u_char *
ngx_http_map_uri_to_path(ngx_http_request_t *r, ngx_str_t *path,
    size_t *root_length, size_t reserved)
{
    u_char                     *last;
    size_t                      alias;
    ngx_http_core_loc_conf_t  *clcf;

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    alias = clcf->alias;

    if (alias && !r->valid_location) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      "\"alias\" cannot be used in location \"%V\" "
                      "where URI was rewritten", &clcf->name);
        return NULL;
    }

    if (clcf->root_lengths == NULL) {

        *root_length = clcf->root.len;

        path->len = clcf->root.len + reserved + r->uri.len - alias + 1;

        path->data = ngx_pnalloc(r->pool, path->len);
        if (path->data == NULL) {
            return NULL;
        }

        last = ngx_copy(path->data, clcf->root.data, clcf->root.len);

    } else {
```

```
2050
2051            if (alias == NGX_MAX_SIZE_T_VALUE) {
2052                reserved += r->add_uri_to_alias ? r->uri.len + 1 : 1;
2053
2054            } else {
2055                reserved += r->uri.len - alias + 1;
2056            }
2057
2058            if (ngx_http_script_run(r, path, clcf->root_lengths->elts, reserved,
2059                                    clcf->root_values->elts)
2060                == NULL)
2061            {
2062                return NULL;
2063            }
2064
2065            if (ngx_get_full_name(r->pool, (ngx_str_t *) &ngx_cycle->prefix, path)
2066                != NGX_OK)
2067            {
2068                return NULL;
2069            }
2070
2071            *root_length = path->len - reserved;
2072            last = path->data + *root_length;
2073
2074            if (alias == NGX_MAX_SIZE_T_VALUE) {
2075                if (!r->add_uri_to_alias) {
2076                    *last = '\0';
2077                    return last;
2078                }
2079
2080                alias = 0;
2081            }
2082        }
2083
2084        last = ngx_cpystrn(last, r->uri.data + alias, r->uri.len - alias + 1);
2085
2086        return last;
2087    }
2088
2089
2090    ngx_int_t
2091    ngx_http_auth_basic_user(ngx_http_request_t *r)
2092    {
2093        ngx_str_t   auth, encoded;
2094        ngx_uint_t  len;
2095
2096        if (r->headers_in.user.len == 0 && r->headers_in.user.data != NULL) {
2097            return NGX_DECLINED;
2098        }
2099
2100        if (r->headers_in.authorization == NULL) {
2101            r->headers_in.user.data = (u_char *) "";
2102            return NGX_DECLINED;
2103        }
2104
2105        encoded = r->headers_in.authorization->value;
2106
2107        if (encoded.len < sizeof("Basic ") - 1
2108            || ngx_strncasecmp(encoded.data, (u_char *) "Basic ",
2109                               sizeof("Basic ") - 1)
2110               != 0)
2111        {
2112            r->headers_in.user.data = (u_char *) "";
2113            return NGX_DECLINED;
2114        }
2115
2116        encoded.len -= sizeof("Basic ") - 1;
2117        encoded.data += sizeof("Basic ") - 1;
2118
2119        while (encoded.len && encoded.data[0] == ' ') {
2120            encoded.len--;
2121            encoded.data++;
2122        }
2123
2124        if (encoded.len == 0) {
2125            r->headers_in.user.data = (u_char *) "";
```

```
2126         return NGX_DECLINED;
2127     }
2128 
2129     auth.len = ngx_base64_decoded_length(encoded.len);
2130     auth.data = ngx_pnalloc(r->pool, auth.len + 1);
2131     if (auth.data == NULL) {
2132         return NGX_ERROR;
2133     }
2134 
2135     if (ngx_decode_base64(&auth, &encoded) != NGX_OK) {
2136         r->headers_in.user.data = (u_char *) "";
2137         return NGX_DECLINED;
2138     }
2139 
2140     auth.data[auth.len] = '\0';
2141 
2142     for (len = 0; len < auth.len; len++) {
2143         if (auth.data[len] == ':') {
2144             break;
2145         }
2146     }
2147 
2148     if (len == 0 || len == auth.len) {
2149         r->headers_in.user.data = (u_char *) "";
2150         return NGX_DECLINED;
2151     }
2152 
2153     r->headers_in.user.len = len;
2154     r->headers_in.user.data = auth.data;
2155     r->headers_in.passwd.len = auth.len - len - 1;
2156     r->headers_in.passwd.data = &auth.data[len + 1];
2157 
2158     return NGX_OK;
2159 }
2160 
2161 
2162 #if (NGX_HTTP_GZIP)
2163 
2164 ngx_int_t
2165 ngx_http_gzip_ok(ngx_http_request_t *r)
2166 {
2167     time_t                     date, expires;
2168     ngx_uint_t                 p;
2169     ngx_array_t               *cc;
2170     ngx_table_elt_t           *e, *d, *ae;
2171     ngx_http_core_loc_conf_t  *clcf;
2172 
2173     r->gzip_tested = 1;
2174 
2175     if (r != r->main) {
2176         return NGX_DECLINED;
2177     }
2178 
2179 #if (NGX_HTTP_SPDY)
2180     if (r->spdy_stream) {
2181         r->gzip_ok = 1;
2182         return NGX_OK;
2183     }
2184 #endif
2185 
2186     ae = r->headers_in.accept_encoding;
2187     if (ae == NULL) {
2188         return NGX_DECLINED;
2189     }
2190 
2191     if (ae->value.len < sizeof("gzip") - 1) {
2192         return NGX_DECLINED;
2193     }
2194 
2195     /*
2196      * test first for the most common case "gzip,...":
2197      *   MSIE:    "gzip, deflate"
2198      *   Firefox: "gzip,deflate"
2199      *   Chrome:  "gzip,deflate,sdch"
2200      *   Safari:  "gzip, deflate"
2201      *   Opera:   "gzip, deflate"
```

```
2202        */
2203
2204    if (ngx_memcmp(ae->value.data, "gzip,", 5) != 0
2205        && ngx_http_gzip_accept_encoding(&ae->value) != NGX_OK)
2206    {
2207        return NGX_DECLINED;
2208    }
2209
2210    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2211
2212    if (r->headers_in.msie6 && clcf->gzip_disable_msie6) {
2213        return NGX_DECLINED;
2214    }
2215
2216    if (r->http_version < clcf->gzip_http_version) {
2217        return NGX_DECLINED;
2218    }
2219
2220    if (r->headers_in.via == NULL) {
2221        goto ok;
2222    }
2223
2224    p = clcf->gzip_proxied;
2225
2226    if (p & NGX_HTTP_GZIP_PROXIED_OFF) {
2227        return NGX_DECLINED;
2228    }
2229
2230    if (p & NGX_HTTP_GZIP_PROXIED_ANY) {
2231        goto ok;
2232    }
2233
2234    if (r->headers_in.authorization && (p & NGX_HTTP_GZIP_PROXIED_AUTH)) {
2235        goto ok;
2236    }
2237
2238    e = r->headers_out.expires;
2239
2240    if (e) {
2241
2242        if (!(p & NGX_HTTP_GZIP_PROXIED_EXPIRED)) {
2243            return NGX_DECLINED;
2244        }
2245
2246        expires = ngx_http_parse_time(e->value.data, e->value.len);
2247        if (expires == NGX_ERROR) {
2248            return NGX_DECLINED;
2249        }
2250
2251        d = r->headers_out.date;
2252
2253        if (d) {
2254            date = ngx_http_parse_time(d->value.data, d->value.len);
2255            if (date == NGX_ERROR) {
2256                return NGX_DECLINED;
2257            }
2258
2259        } else {
2260            date = ngx_time();
2261        }
2262
2263        if (expires < date) {
2264            goto ok;
2265        }
2266
2267        return NGX_DECLINED;
2268    }
2269
2270    cc = &r->headers_out.cache_control;
2271
2272    if (cc->elts) {
2273
2274        if ((p & NGX_HTTP_GZIP_PROXIED_NO_CACHE)
2275            && ngx_http_parse_multi_header_lines(cc, &ngx_http_gzip_no_cache,
2276                                                 NULL)
2277                >= 0)
```

```
2278        {
2279            goto ok;
2280        }
2281
2282        if ((p & NGX_HTTP_GZIP_PROXIED_NO_STORE)
2283            && ngx_http_parse_multi_header_lines(cc, &ngx_http_gzip_no_store,
2284                                                 NULL)
2285               >= 0)
2286        {
2287            goto ok;
2288        }
2289
2290        if ((p & NGX_HTTP_GZIP_PROXIED_PRIVATE)
2291            && ngx_http_parse_multi_header_lines(cc, &ngx_http_gzip_private,
2292                                                 NULL)
2293               >= 0)
2294        {
2295            goto ok;
2296        }
2297
2298        return NGX_DECLINED;
2299    }
2300
2301    if ((p & NGX_HTTP_GZIP_PROXIED_NO_LM) && r->headers_out.last_modified) {
2302        return NGX_DECLINED;
2303    }
2304
2305    if ((p & NGX_HTTP_GZIP_PROXIED_NO_ETAG) && r->headers_out.etag) {
2306        return NGX_DECLINED;
2307    }
2308
2309ok:
2310
2311#if (NGX_PCRE)
2312
2313    if (clcf->gzip_disable && r->headers_in.user_agent) {
2314
2315        if (ngx_regex_exec_array(clcf->gzip_disable,
2316                                 &r->headers_in.user_agent->value,
2317                                 r->connection->log)
2318            != NGX_DECLINED)
2319        {
2320            return NGX_DECLINED;
2321        }
2322    }
2323
2324#endif
2325
2326    r->gzip_ok = 1;
2327
2328    return NGX_OK;
2329}
2330
2331
2332/*
2333* gzip is enabled for the following quantities:
2334*     "gzip; q=0.001" ... "gzip; q=1.000"
2335* gzip is disabled for the following quantities:
2336*     "gzip; q=0" ... "gzip; q=0.000", and for any invalid cases
2337*/
2338
2339static ngx_int_t
2340ngx_http_gzip_accept_encoding(ngx_str_t *ae)
2341{
2342    u_char  *p, *start, *last;
2343
2344    start = ae->data;
2345    last = start + ae->len;
2346
2347    for ( ;; ) {
2348        p = ngx_strcasestrn(start, "gzip", 4 - 1);
2349        if (p == NULL) {
2350            return NGX_DECLINED;
2351        }
2352
2353        if (p == start || (*(p - 1) == ',' || *(p - 1) == ' ')) {
```

```
            break;
        }

        start = p + 4;
    }

    p += 4;

    while (p < last) {
        switch (*p++) {
        case ',':
            return NGX_OK;
        case ';':
            goto quantity;
        case ' ':
            continue;
        default:
            return NGX_DECLINED;
        }
    }

    return NGX_OK;

quantity:

    while (p < last) {
        switch (*p++) {
        case 'q':
        case 'Q':
            goto equal;
        case ' ':
            continue;
        default:
            return NGX_DECLINED;
        }
    }

    return NGX_OK;

equal:

    if (p + 2 > last || *p++ != '=') {
        return NGX_DECLINED;
    }

    if (ngx_http_gzip_quantity(p, last) == 0) {
        return NGX_DECLINED;
    }

    return NGX_OK;
}


static ngx_uint_t
ngx_http_gzip_quantity(u_char *p, u_char *last)
{
    u_char      c;
    ngx_uint_t  n, q;

    c = *p++;

    if (c != '0' && c != '1') {
        return 0;
    }

    q = (c - '0') * 100;

    if (p == last) {
        return q;
    }

    c = *p++;

    if (c == ',' || c == ' ') {
        return q;
    }
```

```
2430
2431        if (c != '.') {
2432            return 0;
2433        }
2434
2435        n = 0;
2436
2437        while (p < last) {
2438            c = *p++;
2439
2440            if (c == ',' || c == ' ') {
2441                break;
2442            }
2443
2444            if (c >= '0' && c <= '9') {
2445                q += c - '0';
2446                n++;
2447                continue;
2448            }
2449
2450            return 0;
2451        }
2452
2453        if (q > 100 || n > 3) {
2454            return 0;
2455        }
2456
2457        return q;
2458    }
2459
2460    #endif
2461
2462
2463    ngx_int_t
2464    ngx_http_subrequest(ngx_http_request_t *r,
2465        ngx_str_t *uri, ngx_str_t *args, ngx_http_request_t **psr,
2466        ngx_http_post_subrequest_t *ps, ngx_uint_t flags)
2467    {
2468        ngx_time_t                     *tp;
2469        ngx_connection_t               *c;
2470        ngx_http_request_t             *sr;
2471        ngx_http_core_srv_conf_t       *cscf;
2472        ngx_http_postponed_request_t   *pr, *p;
2473
2474        r->main->subrequests--;
2475
2476        if (r->main->subrequests == 0) {
2477            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2478                          "subrequests cycle while processing \"%V\"", uri);
2479            r->main->subrequests = 1;
2480            return NGX_ERROR;
2481        }
2482
2483        sr = ngx_pcalloc(r->pool, sizeof(ngx_http_request_t));
2484        if (sr == NULL) {
2485            return NGX_ERROR;
2486        }
2487
2488        sr->signature = NGX_HTTP_MODULE;
2489
2490        c = r->connection;
2491        sr->connection = c;
2492
2493        sr->ctx = ngx_pcalloc(r->pool, sizeof(void *) * ngx_http_max_module);
2494        if (sr->ctx == NULL) {
2495            return NGX_ERROR;
2496        }
2497
2498        if (ngx_list_init(&sr->headers_out.headers, r->pool, 20,
2499                          sizeof(ngx_table_elt_t))
2500            != NGX_OK)
2501        {
2502            return NGX_ERROR;
2503        }
2504
2505        cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
```

```
2506      sr->main_conf = cscf->ctx->main_conf;
2507      sr->srv_conf = cscf->ctx->srv_conf;
2508      sr->loc_conf = cscf->ctx->loc_conf;
2509
2510      sr->pool = r->pool;
2511
2512      sr->headers_in = r->headers_in;
2513
2514      ngx_http_clear_content_length(sr);
2515      ngx_http_clear_accept_ranges(sr);
2516      ngx_http_clear_last_modified(sr);
2517
2518      sr->request_body = r->request_body;
2519
2520  #if (NGX_HTTP_SPDY)
2521      sr->spdy_stream = r->spdy_stream;
2522  #endif
2523
2524      sr->method = NGX_HTTP_GET;
2525      sr->http_version = r->http_version;
2526
2527      sr->request_line = r->request_line;
2528      sr->uri = *uri;
2529
2530      if (args) {
2531          sr->args = *args;
2532      }
2533
2534      ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
2535                     "http subrequest \"%V?%V\"", uri, &sr->args);
2536
2537      sr->subrequest_in_memory = (flags & NGX_HTTP_SUBREQUEST_IN_MEMORY) != 0;
2538      sr->waited = (flags & NGX_HTTP_SUBREQUEST_WAITED) != 0;
2539
2540      sr->unparsed_uri = r->unparsed_uri;
2541      sr->method_name = ngx_http_core_get_method;
2542      sr->http_protocol = r->http_protocol;
2543
2544      ngx_http_set_exten(sr);
2545
2546      sr->main = r->main;
2547      sr->parent = r;
2548      sr->post_subrequest = ps;
2549      sr->read_event_handler = ngx_http_request_empty_handler;
2550      sr->write_event_handler = ngx_http_handler;
2551
2552      if (c->data == r && r->postponed == NULL) {
2553          c->data = sr;
2554      }
2555
2556      sr->variables = r->variables;
2557
2558      sr->log_handler = r->log_handler;
2559
2560      pr = ngx_palloc(r->pool, sizeof(ngx_http_postponed_request_t));
2561      if (pr == NULL) {
2562          return NGX_ERROR;
2563      }
2564
2565      pr->request = sr;
2566      pr->out = NULL;
2567      pr->next = NULL;
2568
2569      if (r->postponed) {
2570          for (p = r->postponed; p->next; p = p->next) { /* void */ }
2571          p->next = pr;
2572
2573      } else {
2574          r->postponed = pr;
2575      }
2576
2577      sr->internal = 1;
2578
2579      sr->discard_body = r->discard_body;
2580      sr->expect_tested = 1;
2581      sr->main_filter_need_in_memory = r->main_filter_need_in_memory;
```

```
2582
2583        sr->uri_changes = NGX_HTTP_MAX_URI_CHANGES + 1;
2584
2585        tp = ngx_timeofday();
2586        sr->start_sec = tp->sec;
2587        sr->start_msec = tp->msec;
2588
2589        r->main->count++;
2590
2591        *psr = sr;
2592
2593        return ngx_http_post_request(sr, NULL);
2594    }
2595
2596
2597    ngx_int_t
2598    ngx_http_internal_redirect(ngx_http_request_t *r,
2599        ngx_str_t *uri, ngx_str_t *args)
2600    {
2601        ngx_http_core_srv_conf_t  *cscf;
2602
2603        r->uri_changes--;
2604
2605        if (r->uri_changes == 0) {
2606            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2607                          "rewrite or internal redirection cycle "
2608                          "while internally redirecting to \"%V\"", uri);
2609
2610            r->main->count++;
2611            ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
2612            return NGX_DONE;
2613        }
2614
2615        r->uri = *uri;
2616
2617        if (args) {
2618            r->args = *args;
2619
2620        } else {
2621            ngx_str_null(&r->args);
2622        }
2623
2624        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2625                       "internal redirect: \"%V?%V\"", uri, &r->args);
2626
2627        ngx_http_set_exten(r);
2628
2629        /* clear the modules contexts */
2630        ngx_memzero(r->ctx, sizeof(void *) * ngx_http_max_module);
2631
2632        cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
2633        r->loc_conf = cscf->ctx->loc_conf;
2634
2635        ngx_http_update_location_config(r);
2636
2637    #if (NGX_HTTP_CACHE)
2638        r->cache = NULL;
2639    #endif
2640
2641        r->internal = 1;
2642        r->valid_unparsed_uri = 0;
2643        r->add_uri_to_alias = 0;
2644        r->main->count++;
2645
2646        ngx_http_handler(r);
2647
2648        return NGX_DONE;
2649    }
2650
2651
2652    ngx_int_t
2653    ngx_http_named_location(ngx_http_request_t *r, ngx_str_t *name)
2654    {
2655        ngx_http_core_srv_conf_t    *cscf;
2656        ngx_http_core_loc_conf_t   **clcfp;
2657        ngx_http_core_main_conf_t   *cmcf;
```

```
2658
2659     r->main->count++;
2660     r->uri_changes--;
2661
2662     if (r->uri_changes == 0) {
2663         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2664                       "rewrite or internal redirection cycle "
2665                       "while redirect to named location \"%V\"", name);
2666
2667         ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
2668         return NGX_DONE;
2669     }
2670
2671     if (r->uri.len == 0) {
2672         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2673                       "empty URI in redirect to named location \"%V\"", name);
2674
2675         ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
2676         return NGX_DONE;
2677     }
2678
2679     cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
2680
2681     if (cscf->named_locations) {
2682
2683         for (clcfp = cscf->named_locations; *clcfp; clcfp++) {
2684
2685             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2686                            "test location: \"%V\"", &(*clcfp)->name);
2687
2688             if (name->len != (*clcfp)->name.len
2689                 || ngx_strncmp(name->data, (*clcfp)->name.data, name->len) != 0)
2690             {
2691                 continue;
2692             }
2693
2694             ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2695                            "using location: %V \"%V?%V\"",
2696                            name, &r->uri, &r->args);
2697
2698             r->internal = 1;
2699             r->content_handler = NULL;
2700             r->uri_changed = 0;
2701             r->loc_conf = (*clcfp)->loc_conf;
2702
2703             /* clear the modules contexts */
2704             ngx_memzero(r->ctx, sizeof(void *) * ngx_http_max_module);
2705
2706             ngx_http_update_location_config(r);
2707
2708             cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
2709
2710             r->phase_handler = cmcf->phase_engine.location_rewrite_index;
2711
2712             r->write_event_handler = ngx_http_core_run_phases;
2713             ngx_http_core_run_phases(r);
2714
2715             return NGX_DONE;
2716         }
2717     }
2718
2719     ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2720                   "could not find named location \"%V\"", name);
2721
2722     ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
2723
2724     return NGX_DONE;
2725 }
2726
2727
2728 ngx_http_cleanup_t *
2729 ngx_http_cleanup_add(ngx_http_request_t *r, size_t size)
2730 {
2731     ngx_http_cleanup_t  *cln;
2732
2733     r = r->main;
```

```
2734
2735      cln = ngx_palloc(r->pool, sizeof(ngx_http_cleanup_t));
2736      if (cln == NULL) {
2737          return NULL;
2738      }
2739
2740      if (size) {
2741          cln->data = ngx_palloc(r->pool, size);
2742          if (cln->data == NULL) {
2743              return NULL;
2744          }
2745
2746      } else {
2747          cln->data = NULL;
2748      }
2749
2750      cln->handler = NULL;
2751      cln->next = r->cleanup;
2752
2753      r->cleanup = cln;
2754
2755      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2756                     "http cleanup add: %p", cln);
2757
2758      return cln;
2759  }
2760
2761
2762  ngx_int_t
2763  ngx_http_set_disable_symlinks(ngx_http_request_t *r,
2764      ngx_http_core_loc_conf_t *clcf, ngx_str_t *path, ngx_open_file_info_t *of)
2765  {
2766  #if (NGX_HAVE_OPENAT)
2767      u_char     *p;
2768      ngx_str_t   from;
2769
2770      of->disable_symlinks = clcf->disable_symlinks;
2771
2772      if (clcf->disable_symlinks_from == NULL) {
2773          return NGX_OK;
2774      }
2775
2776      if (ngx_http_complex_value(r, clcf->disable_symlinks_from, &from)
2777          != NGX_OK)
2778      {
2779          return NGX_ERROR;
2780      }
2781
2782      if (from.len == 0
2783          || from.len > path->len
2784          || ngx_memcmp(path->data, from.data, from.len) != 0)
2785      {
2786          return NGX_OK;
2787      }
2788
2789      if (from.len == path->len) {
2790          of->disable_symlinks = NGX_DISABLE_SYMLINKS_OFF;
2791          return NGX_OK;
2792      }
2793
2794      p = path->data + from.len;
2795
2796      if (*p == '/') {
2797          of->disable_symlinks_from = from.len;
2798          return NGX_OK;
2799      }
2800
2801      p--;
2802
2803      if (*p == '/') {
2804          of->disable_symlinks_from = from.len - 1;
2805      }
2806  #endif
2807
2808      return NGX_OK;
2809  }
```

```
2810
2811
2812 ngx_int_t
2813 ngx_http_get_forwarded_addr(ngx_http_request_t *r, ngx_addr_t *addr,
2814     ngx_array_t *headers, ngx_str_t *value, ngx_array_t *proxies,
2815     int recursive)
2816 {
2817     ngx_int_t          rc;
2818     ngx_uint_t         i, found;
2819     ngx_table_elt_t  **h;
2820
2821     if (headers == NULL) {
2822         return ngx_http_get_forwarded_addr_internal(r, addr, value->data,
2823                                                     value->len, proxies,
2824                                                     recursive);
2825     }
2826
2827     i = headers->nelts;
2828     h = headers->elts;
2829
2830     rc = NGX_DECLINED;
2831
2832     found = 0;
2833
2834     while (i-- > 0) {
2835         rc = ngx_http_get_forwarded_addr_internal(r, addr, h[i]->value.data,
2836                                                   h[i]->value.len, proxies,
2837                                                   recursive);
2838
2839         if (!recursive) {
2840             break;
2841         }
2842
2843         if (rc == NGX_DECLINED && found) {
2844             rc = NGX_DONE;
2845             break;
2846         }
2847
2848         if (rc != NGX_OK) {
2849             break;
2850         }
2851
2852         found = 1;
2853     }
2854
2855     return rc;
2856 }
2857
2858
2859 static ngx_int_t
2860 ngx_http_get_forwarded_addr_internal(ngx_http_request_t *r, ngx_addr_t *addr,
2861     u_char *xff, size_t xfflen, ngx_array_t *proxies, int recursive)
2862 {
2863     u_char           *p;
2864     in_addr_t         inaddr;
2865     ngx_int_t         rc;
2866     ngx_addr_t        paddr;
2867     ngx_cidr_t       *cidr;
2868     ngx_uint_t        family, i;
2869 #if (NGX_HAVE_INET6)
2870     ngx_uint_t        n;
2871     struct in6_addr  *inaddr6;
2872 #endif
2873
2874 #if (NGX_SUPPRESS_WARN)
2875     inaddr = 0;
2876 #if (NGX_HAVE_INET6)
2877     inaddr6 = NULL;
2878 #endif
2879 #endif
2880
2881     family = addr->sockaddr->sa_family;
2882
2883     if (family == AF_INET) {
2884         inaddr = ((struct sockaddr_in *) addr->sockaddr)->sin_addr.s_addr;
2885     }
```

```
2886
2887 #if (NGX_HAVE_INET6)
2888     else if (family == AF_INET6) {
2889         inaddr6 = &((struct sockaddr_in6 *) addr->sockaddr)->sin6_addr;
2890
2891         if (IN6_IS_ADDR_V4MAPPED(inaddr6)) {
2892             family = AF_INET;
2893
2894             p = inaddr6->s6_addr;
2895
2896             inaddr = p[12] << 24;
2897             inaddr += p[13] << 16;
2898             inaddr += p[14] << 8;
2899             inaddr += p[15];
2900
2901             inaddr = htonl(inaddr);
2902         }
2903     }
2904 #endif
2905
2906     for (cidr = proxies->elts, i = 0; i < proxies->nelts; i++) {
2907         if (cidr[i].family != family) {
2908             goto next;
2909         }
2910
2911         switch (family) {
2912
2913 #if (NGX_HAVE_INET6)
2914         case AF_INET6:
2915             for (n = 0; n < 16; n++) {
2916                 if ((inaddr6->s6_addr[n] & cidr[i].u.in6.mask.s6_addr[n])
2917                     != cidr[i].u.in6.addr.s6_addr[n])
2918                 {
2919                     goto next;
2920                 }
2921             }
2922             break;
2923 #endif
2924
2925 #if (NGX_HAVE_UNIX_DOMAIN)
2926         case AF_UNIX:
2927             break;
2928 #endif
2929
2930         default: /* AF_INET */
2931             if ((inaddr & cidr[i].u.in.mask) != cidr[i].u.in.addr) {
2932                 goto next;
2933             }
2934             break;
2935         }
2936
2937         for (p = xff + xfflen - 1; p > xff; p--, xfflen--) {
2938             if (*p != ' ' && *p != ',') {
2939                 break;
2940             }
2941         }
2942
2943         for ( /* void */ ; p > xff; p--) {
2944             if (*p == ' ' || *p == ',') {
2945                 p++;
2946                 break;
2947             }
2948         }
2949
2950         if (ngx_parse_addr(r->pool, &paddr, p, xfflen - (p - xff)) != NGX_OK) {
2951             return NGX_DECLINED;
2952         }
2953
2954         *addr = paddr;
2955
2956         if (recursive && p > xff) {
2957             rc = ngx_http_get_forwarded_addr_internal(r, addr, xff, p - 1 - xff,
2958                                                       proxies, 1);
2959
2960             if (rc == NGX_DECLINED) {
2961                 return NGX_DONE;
```

```
            }

            /* rc == NGX_OK || rc == NGX_DONE  */
            return rc;
        }

        return NGX_OK;

    next:
        continue;
    }

    return NGX_DECLINED;
}


static char *
ngx_http_core_server(ngx_conf_t *cf, ngx_command_t *cmd, void *dummy)
{
    char                        *rv;
    void                        *mconf;
    ngx_uint_t                   i;
    ngx_conf_t                   pcf;
    ngx_http_module_t           *module;
    struct sockaddr_in          *sin;
    ngx_http_conf_ctx_t         *ctx, *http_ctx;
    ngx_http_listen_opt_t        lsopt;
    ngx_http_core_srv_conf_t    *cscf, **cscfp;
    ngx_http_core_main_conf_t   *cmcf;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    http_ctx = cf->ctx;
    ctx->main_conf = http_ctx->main_conf;

    /* the server{}'s srv_conf */

    ctx->srv_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->srv_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    /* the server{}'s loc_conf */

    ctx->loc_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->loc_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_HTTP_MODULE) {
            continue;
        }

        module = ngx_modules[i]->ctx;

        if (module->create_srv_conf) {
            mconf = module->create_srv_conf(cf);
            if (mconf == NULL) {
                return NGX_CONF_ERROR;
            }

            ctx->srv_conf[ngx_modules[i]->ctx_index] = mconf;
        }

        if (module->create_loc_conf) {
            mconf = module->create_loc_conf(cf);
            if (mconf == NULL) {
                return NGX_CONF_ERROR;
            }

            ctx->loc_conf[ngx_modules[i]->ctx_index] = mconf;
        }
```

```
3038     }
3039
3040
3041     /* the server configuration context */
3042
3043     cscf = ctx->srv_conf[ngx_http_core_module.ctx_index];
3044     cscf->ctx = ctx;
3045
3046
3047     cmcf = ctx->main_conf[ngx_http_core_module.ctx_index];
3048
3049     cscfp = ngx_array_push(&cmcf->servers);
3050     if (cscfp == NULL) {
3051         return NGX_CONF_ERROR;
3052     }
3053
3054     *cscfp = cscf;
3055
3056
3057     /* parse inside server{} */
3058
3059     pcf = *cf;
3060     cf->ctx = ctx;
3061     cf->cmd_type = NGX_HTTP_SRV_CONF;
3062
3063     rv = ngx_conf_parse(cf, NULL);
3064
3065     *cf = pcf;
3066
3067     if (rv == NGX_CONF_OK && !cscf->listen) {
3068         ngx_memzero(&lsopt, sizeof(ngx_http_listen_opt_t));
3069
3070         sin = &lsopt.u.sockaddr_in;
3071
3072         sin->sin_family = AF_INET;
3073 #if (NGX_WIN32)
3074         sin->sin_port = htons(80);
3075 #else
3076         sin->sin_port = htons((getuid() == 0) ? 80 : 8000);
3077 #endif
3078         sin->sin_addr.s_addr = INADDR_ANY;
3079
3080         lsopt.socklen = sizeof(struct sockaddr_in);
3081
3082         lsopt.backlog = NGX_LISTEN_BACKLOG;
3083         lsopt.rcvbuf = -1;
3084         lsopt.sndbuf = -1;
3085 #if (NGX_HAVE_SETFIB)
3086         lsopt.setfib = -1;
3087 #endif
3088 #if (NGX_HAVE_TCP_FASTOPEN)
3089         lsopt.fastopen = -1;
3090 #endif
3091         lsopt.wildcard = 1;
3092
3093         (void) ngx_sock_ntop(&lsopt.u.sockaddr, lsopt.socklen, lsopt.addr,
3094                              NGX_SOCKADDR_STRLEN, 1);
3095
3096         if (ngx_http_add_listen(cf, cscf, &lsopt) != NGX_OK) {
3097             return NGX_CONF_ERROR;
3098         }
3099     }
3100
3101     return rv;
3102 }
3103
3104
3105 static char *
3106 ngx_http_core_location(ngx_conf_t *cf, ngx_command_t *cmd, void *dummy)
3107 {
3108     char                      *rv;
3109     u_char                    *mod;
3110     size_t                     len;
3111     ngx_str_t                 *value, *name;
3112     ngx_uint_t                 i;
3113     ngx_conf_t                 save;
```

```
    ngx_http_module_t          *module;
    ngx_http_conf_ctx_t        *ctx, *pctx;
    ngx_http_core_loc_conf_t   *clcf, *pclcf;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    pctx = cf->ctx;
    ctx->main_conf = pctx->main_conf;
    ctx->srv_conf = pctx->srv_conf;

    ctx->loc_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->loc_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_HTTP_MODULE) {
            continue;
        }

        module = ngx_modules[i]->ctx;

        if (module->create_loc_conf) {
            ctx->loc_conf[ngx_modules[i]->ctx_index] =
                                                    module->create_loc_conf(cf);
            if (ctx->loc_conf[ngx_modules[i]->ctx_index] == NULL) {
                 return NGX_CONF_ERROR;
            }
        }
    }

    clcf = ctx->loc_conf[ngx_http_core_module.ctx_index];
    clcf->loc_conf = ctx->loc_conf;

    value = cf->args->elts;

    if (cf->args->nelts == 3) {

        len = value[1].len;
        mod = value[1].data;
        name = &value[2];

        if (len == 1 && mod[0] == '=') {

            clcf->name = *name;
            clcf->exact_match = 1;

        } else if (len == 2 && mod[0] == '^' && mod[1] == '~') {

            clcf->name = *name;
            clcf->noregex = 1;

        } else if (len == 1 && mod[0] == '~') {

            if (ngx_http_core_regex_location(cf, clcf, name, 0) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

        } else if (len == 2 && mod[0] == '~' && mod[1] == '*') {

            if (ngx_http_core_regex_location(cf, clcf, name, 1) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

        } else {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid location modifier \"%V\"", &value[1]);
            return NGX_CONF_ERROR;
        }

    } else {

        name = &value[1];
```

```
3190
3191        if (name->data[0] == '=') {
3192
3193            clcf->name.len = name->len - 1;
3194            clcf->name.data = name->data + 1;
3195            clcf->exact_match = 1;
3196
3197        } else if (name->data[0] == '^' && name->data[1] == '~') {
3198
3199            clcf->name.len = name->len - 2;
3200            clcf->name.data = name->data + 2;
3201            clcf->noregex = 1;
3202
3203        } else if (name->data[0] == '~') {
3204
3205            name->len--;
3206            name->data++;
3207
3208            if (name->data[0] == '*') {
3209
3210                name->len--;
3211                name->data++;
3212
3213                if (ngx_http_core_regex_location(cf, clcf, name, 1) != NGX_OK) {
3214                    return NGX_CONF_ERROR;
3215                }
3216
3217            } else {
3218                if (ngx_http_core_regex_location(cf, clcf, name, 0) != NGX_OK) {
3219                    return NGX_CONF_ERROR;
3220                }
3221            }
3222
3223        } else {
3224
3225            clcf->name = *name;
3226
3227            if (name->data[0] == '@') {
3228                clcf->named = 1;
3229            }
3230        }
3231    }
3232
3233    pclcf = pctx->loc_conf[ngx_http_core_module.ctx_index];
3234
3235    if (pclcf->name.len) {
3236
3237        /* nested location */
3238
3239#if 0
3240        clcf->prev_location = pclcf;
3241#endif
3242
3243        if (pclcf->exact_match) {
3244            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3245                               "location \"%V\" cannot be inside "
3246                               "the exact location \"%V\"",
3247                               &clcf->name, &pclcf->name);
3248            return NGX_CONF_ERROR;
3249        }
3250
3251        if (pclcf->named) {
3252            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3253                               "location \"%V\" cannot be inside "
3254                               "the named location \"%V\"",
3255                               &clcf->name, &pclcf->name);
3256            return NGX_CONF_ERROR;
3257        }
3258
3259        if (clcf->named) {
3260            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3261                               "named location \"%V\" can be "
3262                               "on the server level only",
3263                               &clcf->name);
3264            return NGX_CONF_ERROR;
3265        }
```

```
3266
3267            len = pclcf->name.len;
3268
3269    #if (NGX_PCRE)
3270            if (clcf->regex == NULL
3271                && ngx_filename_cmp(clcf->name.data, pclcf->name.data, len) != 0)
3272    #else
3273            if (ngx_filename_cmp(clcf->name.data, pclcf->name.data, len) != 0)
3274    #endif
3275            {
3276                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3277                                   "location \"%V\" is outside location \"%V\"",
3278                                   &clcf->name, &pclcf->name);
3279                return NGX_CONF_ERROR;
3280            }
3281        }
3282
3283        if (ngx_http_add_location(cf, &pclcf->locations, clcf) != NGX_OK) {
3284            return NGX_CONF_ERROR;
3285        }
3286
3287        save = *cf;
3288        cf->ctx = ctx;
3289        cf->cmd_type = NGX_HTTP_LOC_CONF;
3290
3291        rv = ngx_conf_parse(cf, NULL);
3292
3293        *cf = save;
3294
3295        return rv;
3296    }
3297
3298
3299    static ngx_int_t
3300    ngx_http_core_regex_location(ngx_conf_t *cf, ngx_http_core_loc_conf_t *clcf,
3301        ngx_str_t *regex, ngx_uint_t caseless)
3302    {
3303    #if (NGX_PCRE)
3304        ngx_regex_compile_t  rc;
3305        u_char               errstr[NGX_MAX_CONF_ERRSTR];
3306
3307        ngx_memzero(&rc, sizeof(ngx_regex_compile_t));
3308
3309        rc.pattern = *regex;
3310        rc.err.len = NGX_MAX_CONF_ERRSTR;
3311        rc.err.data = errstr;
3312
3313    #if (NGX_HAVE_CASELESS_FILESYSTEM)
3314        rc.options = NGX_REGEX_CASELESS;
3315    #else
3316        rc.options = caseless ? NGX_REGEX_CASELESS : 0;
3317    #endif
3318
3319        clcf->regex = ngx_http_regex_compile(cf, &rc);
3320        if (clcf->regex == NULL) {
3321            return NGX_ERROR;
3322        }
3323
3324        clcf->name = *regex;
3325
3326        return NGX_OK;
3327
3328    #else
3329
3330        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3331                           "using regex \"%V\" requires PCRE library",
3332                           regex);
3333        return NGX_ERROR;
3334
3335    #endif
3336    }
3337
3338
3339    static char *
3340    ngx_http_core_types(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3341    {
```

```
3342      ngx_http_core_loc_conf_t *clcf = conf;
3343
3344      char        *rv;
3345      ngx_conf_t   save;
3346
3347      if (clcf->types == NULL) {
3348          clcf->types = ngx_array_create(cf->pool, 64, sizeof(ngx_hash_key_t));
3349          if (clcf->types == NULL) {
3350              return NGX_CONF_ERROR;
3351          }
3352      }
3353
3354      save = *cf;
3355      cf->handler = ngx_http_core_type;
3356      cf->handler_conf = conf;
3357
3358      rv = ngx_conf_parse(cf, NULL);
3359
3360      *cf = save;
3361
3362      return rv;
3363  }
3364
3365
3366  static char *
3367  ngx_http_core_type(ngx_conf_t *cf, ngx_command_t *dummy, void *conf)
3368  {
3369      ngx_http_core_loc_conf_t *clcf = conf;
3370
3371      ngx_str_t       *value, *content_type, *old;
3372      ngx_uint_t       i, n, hash;
3373      ngx_hash_key_t  *type;
3374
3375      value = cf->args->elts;
3376
3377      if (ngx_strcmp(value[0].data, "include") == 0) {
3378          if (cf->args->nelts != 2) {
3379              ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3380                                 "invalid number of arguments"
3381                                 " in \"include\" directive");
3382              return NGX_CONF_ERROR;
3383          }
3384
3385          return ngx_conf_include(cf, dummy, conf);
3386      }
3387
3388      content_type = ngx_palloc(cf->pool, sizeof(ngx_str_t));
3389      if (content_type == NULL) {
3390          return NGX_CONF_ERROR;
3391      }
3392
3393      *content_type = value[0];
3394
3395      for (i = 1; i < cf->args->nelts; i++) {
3396
3397          hash = ngx_hash_strlow(value[i].data, value[i].data, value[i].len);
3398
3399          type = clcf->types->elts;
3400          for (n = 0; n < clcf->types->nelts; n++) {
3401              if (ngx_strcmp(value[i].data, type[n].key.data) == 0) {
3402                  old = type[n].value;
3403                  type[n].value = content_type;
3404
3405                  ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
3406                                     "duplicate extension \"%V\", "
3407                                     "content type: \"%V\", "
3408                                     "previous content type: \"%V\"",
3409                                     &value[i], content_type, old);
3410                  goto next;
3411              }
3412          }
3413
3414
3415          type = ngx_array_push(clcf->types);
3416          if (type == NULL) {
3417              return NGX_CONF_ERROR;
```

```
3418        }
3419
3420        type->key = value[i];
3421        type->key_hash = hash;
3422        type->value = content_type;
3423
3424    next:
3425        continue;
3426    }
3427
3428    return NGX_CONF_OK;
3429 }
3430
3431
3432 static ngx_int_t
3433 ngx_http_core_preconfiguration(ngx_conf_t *cf)
3434 {
3435    return ngx_http_variables_add_core_vars(cf);
3436 }
3437
3438
3439 static void *
3440 ngx_http_core_create_main_conf(ngx_conf_t *cf)
3441 {
3442    ngx_http_core_main_conf_t  *cmcf;
3443
3444    cmcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_core_main_conf_t));
3445    if (cmcf == NULL) {
3446        return NULL;
3447    }
3448
3449    if (ngx_array_init(&cmcf->servers, cf->pool, 4,
3450                       sizeof(ngx_http_core_srv_conf_t *))
3451        != NGX_OK)
3452    {
3453        return NULL;
3454    }
3455
3456    cmcf->server_names_hash_max_size = NGX_CONF_UNSET_UINT;
3457    cmcf->server_names_hash_bucket_size = NGX_CONF_UNSET_UINT;
3458
3459    cmcf->variables_hash_max_size = NGX_CONF_UNSET_UINT;
3460    cmcf->variables_hash_bucket_size = NGX_CONF_UNSET_UINT;
3461
3462    return cmcf;
3463 }
3464
3465
3466 static char *
3467 ngx_http_core_init_main_conf(ngx_conf_t *cf, void *conf)
3468 {
3469    ngx_http_core_main_conf_t *cmcf = conf;
3470
3471    ngx_conf_init_uint_value(cmcf->server_names_hash_max_size, 512);
3472    ngx_conf_init_uint_value(cmcf->server_names_hash_bucket_size,
3473                             ngx_cacheline_size);
3474
3475    cmcf->server_names_hash_bucket_size =
3476            ngx_align(cmcf->server_names_hash_bucket_size, ngx_cacheline_size);
3477
3478
3479    ngx_conf_init_uint_value(cmcf->variables_hash_max_size, 1024);
3480    ngx_conf_init_uint_value(cmcf->variables_hash_bucket_size, 64);
3481
3482    cmcf->variables_hash_bucket_size =
3483               ngx_align(cmcf->variables_hash_bucket_size, ngx_cacheline_size);
3484
3485    if (cmcf->ncaptures) {
3486        cmcf->ncaptures = (cmcf->ncaptures + 1) * 3;
3487    }
3488
3489    return NGX_CONF_OK;
3490 }
3491
3492
3493 static void *
```

```
3494 ngx_http_core_create_srv_conf(ngx_conf_t *cf)
3495 {
3496     ngx_http_core_srv_conf_t  *cscf;
3497
3498     cscf = ngx_pcalloc(cf->pool, sizeof(ngx_http_core_srv_conf_t));
3499     if (cscf == NULL) {
3500         return NULL;
3501     }
3502
3503     /*
3504      * set by ngx_pcalloc():
3505      *
3506      *     conf->client_large_buffers.num = 0;
3507      */
3508
3509     if (ngx_array_init(&cscf->server_names, cf->temp_pool, 4,
3510                        sizeof(ngx_http_server_name_t))
3511         != NGX_OK)
3512     {
3513         return NULL;
3514     }
3515
3516     cscf->connection_pool_size = NGX_CONF_UNSET_SIZE;
3517     cscf->request_pool_size = NGX_CONF_UNSET_SIZE;
3518     cscf->client_header_timeout = NGX_CONF_UNSET_MSEC;
3519     cscf->client_header_buffer_size = NGX_CONF_UNSET_SIZE;
3520     cscf->ignore_invalid_headers = NGX_CONF_UNSET;
3521     cscf->merge_slashes = NGX_CONF_UNSET;
3522     cscf->underscores_in_headers = NGX_CONF_UNSET;
3523
3524     return cscf;
3525 }
3526
3527
3528 static char *
3529 ngx_http_core_merge_srv_conf(ngx_conf_t *cf, void *parent, void *child)
3530 {
3531     ngx_http_core_srv_conf_t *prev = parent;
3532     ngx_http_core_srv_conf_t *conf = child;
3533
3534     ngx_str_t                name;
3535     ngx_http_server_name_t  *sn;
3536
3537     /* TODO: it does not merge, it inits only */
3538
3539     ngx_conf_merge_size_value(conf->connection_pool_size,
3540                               prev->connection_pool_size, 256);
3541     ngx_conf_merge_size_value(conf->request_pool_size,
3542                               prev->request_pool_size, 4096);
3543     ngx_conf_merge_msec_value(conf->client_header_timeout,
3544                               prev->client_header_timeout, 60000);
3545     ngx_conf_merge_size_value(conf->client_header_buffer_size,
3546                               prev->client_header_buffer_size, 1024);
3547     ngx_conf_merge_bufs_value(conf->large_client_header_buffers,
3548                               prev->large_client_header_buffers,
3549                               4, 8192);
3550
3551     if (conf->large_client_header_buffers.size < conf->connection_pool_size) {
3552         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3553                            "the \"large_client_header_buffers\" size must be "
3554                            "equal to or greater than \"connection_pool_size\"");
3555         return NGX_CONF_ERROR;
3556     }
3557
3558     ngx_conf_merge_value(conf->ignore_invalid_headers,
3559                               prev->ignore_invalid_headers, 1);
3560
3561     ngx_conf_merge_value(conf->merge_slashes, prev->merge_slashes, 1);
3562
3563     ngx_conf_merge_value(conf->underscores_in_headers,
3564                               prev->underscores_in_headers, 0);
3565
3566     if (conf->server_names.nelts == 0) {
3567         /* the array has 4 empty preallocated elements, so push cannot fail */
3568         sn = ngx_array_push(&conf->server_names);
3569 #if (NGX_PCRE)
```

```
3570            sn->regex = NULL;
3571    #endif
3572            sn->server = conf;
3573            ngx_str_set(&sn->name, "");
3574        }
3575
3576        sn = conf->server_names.elts;
3577        name = sn[0].name;
3578
3579    #if (NGX_PCRE)
3580        if (sn->regex) {
3581            name.len++;
3582            name.data--;
3583        } else
3584    #endif
3585
3586        if (name.data[0] == '.') {
3587            name.len--;
3588            name.data++;
3589        }
3590
3591        conf->server_name.len = name.len;
3592        conf->server_name.data = ngx_pstrdup(cf->pool, &name);
3593        if (conf->server_name.data == NULL) {
3594            return NGX_CONF_ERROR;
3595        }
3596
3597        return NGX_CONF_OK;
3598    }
3599
3600
3601    static void *
3602    ngx_http_core_create_loc_conf(ngx_conf_t *cf)
3603    {
3604        ngx_http_core_loc_conf_t  *clcf;
3605
3606        clcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_core_loc_conf_t));
3607        if (clcf == NULL) {
3608            return NULL;
3609        }
3610
3611        /*
3612         * set by ngx_pcalloc():
3613         *
3614         *     clcf->root = { 0, NULL };
3615         *     clcf->limit_except = 0;
3616         *     clcf->post_action = { 0, NULL };
3617         *     clcf->types = NULL;
3618         *     clcf->default_type = { 0, NULL };
3619         *     clcf->error_log = NULL;
3620         *     clcf->error_pages = NULL;
3621         *     clcf->try_files = NULL;
3622         *     clcf->client_body_path = NULL;
3623         *     clcf->regex = NULL;
3624         *     clcf->exact_match = 0;
3625         *     clcf->auto_redirect = 0;
3626         *     clcf->alias = 0;
3627         *     clcf->gzip_proxied = 0;
3628         *     clcf->keepalive_disable = 0;
3629         */
3630
3631        clcf->client_max_body_size = NGX_CONF_UNSET;
3632        clcf->client_body_buffer_size = NGX_CONF_UNSET_SIZE;
3633        clcf->client_body_timeout = NGX_CONF_UNSET_MSEC;
3634        clcf->satisfy = NGX_CONF_UNSET_UINT;
3635        clcf->if_modified_since = NGX_CONF_UNSET_UINT;
3636        clcf->max_ranges = NGX_CONF_UNSET_UINT;
3637        clcf->client_body_in_file_only = NGX_CONF_UNSET_UINT;
3638        clcf->client_body_in_single_buffer = NGX_CONF_UNSET;
3639        clcf->internal = NGX_CONF_UNSET;
3640        clcf->sendfile = NGX_CONF_UNSET;
3641        clcf->sendfile_max_chunk = NGX_CONF_UNSET_SIZE;
3642    #if (NGX_HAVE_FILE_AIO)
3643        clcf->aio = NGX_CONF_UNSET;
3644    #endif
3645        clcf->read_ahead = NGX_CONF_UNSET_SIZE;
```

```
3646      clcf->directio = NGX_CONF_UNSET;
3647      clcf->directio_alignment = NGX_CONF_UNSET;
3648      clcf->tcp_nopush = NGX_CONF_UNSET;
3649      clcf->tcp_nodelay = NGX_CONF_UNSET;
3650      clcf->send_timeout = NGX_CONF_UNSET_MSEC;
3651      clcf->send_lowat = NGX_CONF_UNSET_SIZE;
3652      clcf->postpone_output = NGX_CONF_UNSET_SIZE;
3653      clcf->limit_rate = NGX_CONF_UNSET_SIZE;
3654      clcf->limit_rate_after = NGX_CONF_UNSET_SIZE;
3655      clcf->keepalive_timeout = NGX_CONF_UNSET_MSEC;
3656      clcf->keepalive_header = NGX_CONF_UNSET;
3657      clcf->keepalive_requests = NGX_CONF_UNSET_UINT;
3658      clcf->lingering_close = NGX_CONF_UNSET_UINT;
3659      clcf->lingering_time = NGX_CONF_UNSET_MSEC;
3660      clcf->lingering_timeout = NGX_CONF_UNSET_MSEC;
3661      clcf->resolver_timeout = NGX_CONF_UNSET_MSEC;
3662      clcf->reset_timedout_connection = NGX_CONF_UNSET;
3663      clcf->server_name_in_redirect = NGX_CONF_UNSET;
3664      clcf->port_in_redirect = NGX_CONF_UNSET;
3665      clcf->msie_padding = NGX_CONF_UNSET;
3666      clcf->msie_refresh = NGX_CONF_UNSET;
3667      clcf->log_not_found = NGX_CONF_UNSET;
3668      clcf->log_subrequest = NGX_CONF_UNSET;
3669      clcf->recursive_error_pages = NGX_CONF_UNSET;
3670      clcf->server_tokens = NGX_CONF_UNSET;
3671      clcf->chunked_transfer_encoding = NGX_CONF_UNSET;
3672      clcf->etag = NGX_CONF_UNSET;
3673      clcf->types_hash_max_size = NGX_CONF_UNSET_UINT;
3674      clcf->types_hash_bucket_size = NGX_CONF_UNSET_UINT;
3675
3676      clcf->open_file_cache = NGX_CONF_UNSET_PTR;
3677      clcf->open_file_cache_valid = NGX_CONF_UNSET;
3678      clcf->open_file_cache_min_uses = NGX_CONF_UNSET_UINT;
3679      clcf->open_file_cache_errors = NGX_CONF_UNSET;
3680      clcf->open_file_cache_events = NGX_CONF_UNSET;
3681
3682  #if (NGX_HTTP_GZIP)
3683      clcf->gzip_vary = NGX_CONF_UNSET;
3684      clcf->gzip_http_version = NGX_CONF_UNSET_UINT;
3685  #if (NGX_PCRE)
3686      clcf->gzip_disable = NGX_CONF_UNSET_PTR;
3687  #endif
3688      clcf->gzip_disable_msie6 = 3;
3689  #if (NGX_HTTP_DEGRADATION)
3690      clcf->gzip_disable_degradation = 3;
3691  #endif
3692  #endif
3693
3694  #if (NGX_HAVE_OPENAT)
3695      clcf->disable_symlinks = NGX_CONF_UNSET_UINT;
3696      clcf->disable_symlinks_from = NGX_CONF_UNSET_PTR;
3697  #endif
3698
3699      return clcf;
3700  }
3701
3702
3703  static ngx_str_t  ngx_http_core_text_html_type = ngx_string("text/html");
3704  static ngx_str_t  ngx_http_core_image_gif_type = ngx_string("image/gif");
3705  static ngx_str_t  ngx_http_core_image_jpeg_type = ngx_string("image/jpeg");
3706
3707  static ngx_hash_key_t  ngx_http_core_default_types[] = {
3708      { ngx_string("html"), 0, &ngx_http_core_text_html_type },
3709      { ngx_string("gif"), 0, &ngx_http_core_image_gif_type },
3710      { ngx_string("jpg"), 0, &ngx_http_core_image_jpeg_type },
3711      { ngx_null_string, 0, NULL }
3712  };
3713
3714
3715  static char *
3716  ngx_http_core_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
3717  {
3718      ngx_http_core_loc_conf_t *prev = parent;
3719      ngx_http_core_loc_conf_t *conf = child;
3720
3721      ngx_uint_t        i;
```

```
3722      ngx_hash_key_t   *type;
3723      ngx_hash_init_t   types_hash;
3724
3725      if (conf->root.data == NULL) {
3726
3727          conf->alias = prev->alias;
3728          conf->root = prev->root;
3729          conf->root_lengths = prev->root_lengths;
3730          conf->root_values = prev->root_values;
3731
3732          if (prev->root.data == NULL) {
3733              ngx_str_set(&conf->root, "html");
3734
3735              if (ngx_conf_full_name(cf->cycle, &conf->root, 0) != NGX_OK) {
3736                  return NGX_CONF_ERROR;
3737              }
3738          }
3739      }
3740
3741      if (conf->post_action.data == NULL) {
3742          conf->post_action = prev->post_action;
3743      }
3744
3745      ngx_conf_merge_uint_value(conf->types_hash_max_size,
3746                                prev->types_hash_max_size, 1024);
3747
3748      ngx_conf_merge_uint_value(conf->types_hash_bucket_size,
3749                                prev->types_hash_bucket_size, 64);
3750
3751      conf->types_hash_bucket_size = ngx_align(conf->types_hash_bucket_size,
3752                                               ngx_cacheline_size);
3753
3754      /*
3755       * the special handling of the "types" directive in the "http" section
3756       * to inherit the http's conf->types_hash to all servers
3757       */
3758
3759      if (prev->types && prev->types_hash.buckets == NULL) {
3760
3761          types_hash.hash = &prev->types_hash;
3762          types_hash.key = ngx_hash_key_lc;
3763          types_hash.max_size = conf->types_hash_max_size;
3764          types_hash.bucket_size = conf->types_hash_bucket_size;
3765          types_hash.name = "types_hash";
3766          types_hash.pool = cf->pool;
3767          types_hash.temp_pool = NULL;
3768
3769          if (ngx_hash_init(&types_hash, prev->types->elts, prev->types->nelts)
3770              != NGX_OK)
3771          {
3772              return NGX_CONF_ERROR;
3773          }
3774      }
3775
3776      if (conf->types == NULL) {
3777          conf->types = prev->types;
3778          conf->types_hash = prev->types_hash;
3779      }
3780
3781      if (conf->types == NULL) {
3782          conf->types = ngx_array_create(cf->pool, 3, sizeof(ngx_hash_key_t));
3783          if (conf->types == NULL) {
3784              return NGX_CONF_ERROR;
3785          }
3786
3787          for (i = 0; ngx_http_core_default_types[i].key.len; i++) {
3788              type = ngx_array_push(conf->types);
3789              if (type == NULL) {
3790                  return NGX_CONF_ERROR;
3791              }
3792
3793              type->key = ngx_http_core_default_types[i].key;
3794              type->key_hash =
3795                        ngx_hash_key_lc(ngx_http_core_default_types[i].key.data,
3796                                        ngx_http_core_default_types[i].key.len);
3797              type->value = ngx_http_core_default_types[i].value;
```

```
3798        }
3799    }
3800
3801    if (conf->types_hash.buckets == NULL) {
3802
3803        types_hash.hash = &conf->types_hash;
3804        types_hash.key = ngx_hash_key_lc;
3805        types_hash.max_size = conf->types_hash_max_size;
3806        types_hash.bucket_size = conf->types_hash_bucket_size;
3807        types_hash.name = "types_hash";
3808        types_hash.pool = cf->pool;
3809        types_hash.temp_pool = NULL;
3810
3811        if (ngx_hash_init(&types_hash, conf->types->elts, conf->types->nelts)
3812            != NGX_OK)
3813        {
3814            return NGX_CONF_ERROR;
3815        }
3816    }
3817
3818    if (conf->error_log == NULL) {
3819        if (prev->error_log) {
3820            conf->error_log = prev->error_log;
3821        } else {
3822            conf->error_log = &cf->cycle->new_log;
3823        }
3824    }
3825
3826    if (conf->error_pages == NULL && prev->error_pages) {
3827        conf->error_pages = prev->error_pages;
3828    }
3829
3830    ngx_conf_merge_str_value(conf->default_type,
3831                             prev->default_type, "text/plain");
3832
3833    ngx_conf_merge_off_value(conf->client_max_body_size,
3834                             prev->client_max_body_size, 1 * 1024 * 1024);
3835    ngx_conf_merge_size_value(conf->client_body_buffer_size,
3836                             prev->client_body_buffer_size,
3837                             (size_t) 2 * ngx_pagesize);
3838    ngx_conf_merge_msec_value(conf->client_body_timeout,
3839                             prev->client_body_timeout, 60000);
3840
3841    ngx_conf_merge_bitmask_value(conf->keepalive_disable,
3842                             prev->keepalive_disable,
3843                             (NGX_CONF_BITMASK_SET
3844                              |NGX_HTTP_KEEPALIVE_DISABLE_MSIE6));
3845    ngx_conf_merge_uint_value(conf->satisfy, prev->satisfy,
3846                             NGX_HTTP_SATISFY_ALL);
3847    ngx_conf_merge_uint_value(conf->if_modified_since, prev->if_modified_since,
3848                             NGX_HTTP_IMS_EXACT);
3849    ngx_conf_merge_uint_value(conf->max_ranges, prev->max_ranges,
3850                             NGX_MAX_INT32_VALUE);
3851    ngx_conf_merge_uint_value(conf->client_body_in_file_only,
3852                             prev->client_body_in_file_only,
3853                             NGX_HTTP_REQUEST_BODY_FILE_OFF);
3854    ngx_conf_merge_value(conf->client_body_in_single_buffer,
3855                             prev->client_body_in_single_buffer, 0);
3856    ngx_conf_merge_value(conf->internal, prev->internal, 0);
3857    ngx_conf_merge_value(conf->sendfile, prev->sendfile, 0);
3858    ngx_conf_merge_size_value(conf->sendfile_max_chunk,
3859                             prev->sendfile_max_chunk, 0);
3860#if (NGX_HAVE_FILE_AIO)
3861    ngx_conf_merge_value(conf->aio, prev->aio, NGX_HTTP_AIO_OFF);
3862#endif
3863    ngx_conf_merge_size_value(conf->read_ahead, prev->read_ahead, 0);
3864    ngx_conf_merge_off_value(conf->directio, prev->directio,
3865                             NGX_OPEN_FILE_DIRECTIO_OFF);
3866    ngx_conf_merge_off_value(conf->directio_alignment, prev->directio_alignment,
3867                             512);
3868    ngx_conf_merge_value(conf->tcp_nopush, prev->tcp_nopush, 0);
3869    ngx_conf_merge_value(conf->tcp_nodelay, prev->tcp_nodelay, 1);
3870
3871    ngx_conf_merge_msec_value(conf->send_timeout, prev->send_timeout, 60000);
3872    ngx_conf_merge_size_value(conf->send_lowat, prev->send_lowat, 0);
3873    ngx_conf_merge_size_value(conf->postpone_output, prev->postpone_output,
```

```
                              1460);
    ngx_conf_merge_size_value(conf->limit_rate, prev->limit_rate, 0);
    ngx_conf_merge_size_value(conf->limit_rate_after, prev->limit_rate_after,
                              0);
    ngx_conf_merge_msec_value(conf->keepalive_timeout,
                              prev->keepalive_timeout, 75000);
    ngx_conf_merge_sec_value(conf->keepalive_header,
                              prev->keepalive_header, 0);
    ngx_conf_merge_uint_value(conf->keepalive_requests,
                              prev->keepalive_requests, 100);
    ngx_conf_merge_uint_value(conf->lingering_close,
                              prev->lingering_close, NGX_HTTP_LINGERING_ON);
    ngx_conf_merge_msec_value(conf->lingering_time,
                              prev->lingering_time, 30000);
    ngx_conf_merge_msec_value(conf->lingering_timeout,
                              prev->lingering_timeout, 5000);
    ngx_conf_merge_msec_value(conf->resolver_timeout,
                              prev->resolver_timeout, 30000);

    if (conf->resolver == NULL) {

        if (prev->resolver == NULL) {

            /*
             * create dummy resolver in http {} context
             * to inherit it in all servers
             */

            prev->resolver = ngx_resolver_create(cf, NULL, 0);
            if (prev->resolver == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        conf->resolver = prev->resolver;
    }

    if (ngx_conf_merge_path_value(cf, &conf->client_body_temp_path,
                              prev->client_body_temp_path,
                              &ngx_http_client_temp_path)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_value(conf->reset_timedout_connection,
                              prev->reset_timedout_connection, 0);
    ngx_conf_merge_value(conf->server_name_in_redirect,
                              prev->server_name_in_redirect, 0);
    ngx_conf_merge_value(conf->port_in_redirect, prev->port_in_redirect, 1);
    ngx_conf_merge_value(conf->msie_padding, prev->msie_padding, 1);
    ngx_conf_merge_value(conf->msie_refresh, prev->msie_refresh, 0);
    ngx_conf_merge_value(conf->log_not_found, prev->log_not_found, 1);
    ngx_conf_merge_value(conf->log_subrequest, prev->log_subrequest, 0);
    ngx_conf_merge_value(conf->recursive_error_pages,
                              prev->recursive_error_pages, 0);
    ngx_conf_merge_value(conf->server_tokens, prev->server_tokens, 1);
    ngx_conf_merge_value(conf->chunked_transfer_encoding,
                              prev->chunked_transfer_encoding, 1);
    ngx_conf_merge_value(conf->etag, prev->etag, 1);

    ngx_conf_merge_ptr_value(conf->open_file_cache,
                              prev->open_file_cache, NULL);

    ngx_conf_merge_sec_value(conf->open_file_cache_valid,
                              prev->open_file_cache_valid, 60);

    ngx_conf_merge_uint_value(conf->open_file_cache_min_uses,
                              prev->open_file_cache_min_uses, 1);

    ngx_conf_merge_sec_value(conf->open_file_cache_errors,
                              prev->open_file_cache_errors, 0);

    ngx_conf_merge_sec_value(conf->open_file_cache_events,
                              prev->open_file_cache_events, 0);
#if (NGX_HTTP_GZIP)
```

```
3950
3951        ngx_conf_merge_value(conf->gzip_vary, prev->gzip_vary, 0);
3952        ngx_conf_merge_uint_value(conf->gzip_http_version, prev->gzip_http_version,
3953                                  NGX_HTTP_VERSION_11);
3954        ngx_conf_merge_bitmask_value(conf->gzip_proxied, prev->gzip_proxied,
3955                                  (NGX_CONF_BITMASK_SET|NGX_HTTP_GZIP_PROXIED_OFF));
3956
3957    #if (NGX_PCRE)
3958        ngx_conf_merge_ptr_value(conf->gzip_disable, prev->gzip_disable, NULL);
3959    #endif
3960
3961        if (conf->gzip_disable_msie6 == 3) {
3962            conf->gzip_disable_msie6 =
3963                (prev->gzip_disable_msie6 == 3) ? 0 : prev->gzip_disable_msie6;
3964        }
3965
3966    #if (NGX_HTTP_DEGRADATION)
3967
3968        if (conf->gzip_disable_degradation == 3) {
3969            conf->gzip_disable_degradation =
3970                (prev->gzip_disable_degradation == 3) ?
3971                    0 : prev->gzip_disable_degradation;
3972        }
3973
3974    #endif
3975    #endif
3976
3977    #if (NGX_HAVE_OPENAT)
3978        ngx_conf_merge_uint_value(conf->disable_symlinks, prev->disable_symlinks,
3979                                  NGX_DISABLE_SYMLINKS_OFF);
3980        ngx_conf_merge_ptr_value(conf->disable_symlinks_from,
3981                                 prev->disable_symlinks_from, NULL);
3982    #endif
3983
3984        return NGX_CONF_OK;
3985    }
3986
3987
3988    static char *
3989    ngx_http_core_listen(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3990    {
3991        ngx_http_core_srv_conf_t *cscf = conf;
3992
3993        ngx_str_t              *value, size;
3994        ngx_url_t               u;
3995        ngx_uint_t              n;
3996        ngx_http_listen_opt_t   lsopt;
3997
3998        cscf->listen = 1;
3999
4000        value = cf->args->elts;
4001
4002        ngx_memzero(&u, sizeof(ngx_url_t));
4003
4004        u.url = value[1];
4005        u.listen = 1;
4006        u.default_port = 80;
4007
4008        if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
4009            if (u.err) {
4010                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4011                                   "%s in \"%V\" of the \"listen\" directive",
4012                                   u.err, &u.url);
4013            }
4014
4015            return NGX_CONF_ERROR;
4016        }
4017
4018        ngx_memzero(&lsopt, sizeof(ngx_http_listen_opt_t));
4019
4020        ngx_memcpy(&lsopt.u.sockaddr, u.sockaddr, u.socklen);
4021
4022        lsopt.socklen = u.socklen;
4023        lsopt.backlog = NGX_LISTEN_BACKLOG;
4024        lsopt.rcvbuf = -1;
4025        lsopt.sndbuf = -1;
```

```
#if (NGX_HAVE_SETFIB)
    lsopt.setfib = -1;
#endif
#if (NGX_HAVE_TCP_FASTOPEN)
    lsopt.fastopen = -1;
#endif
    lsopt.wildcard = u.wildcard;
#if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
    lsopt.ipv6only = 1;
#endif

    (void) ngx_sock_ntop(&lsopt.u.sockaddr, lsopt.socklen, lsopt.addr,
                         NGX_SOCKADDR_STRLEN, 1);

    for (n = 2; n < cf->args->nelts; n++) {

        if (ngx_strcmp(value[n].data, "default_server") == 0
            || ngx_strcmp(value[n].data, "default") == 0)
        {
            lsopt.default_server = 1;
            continue;
        }

        if (ngx_strcmp(value[n].data, "bind") == 0) {
            lsopt.set = 1;
            lsopt.bind = 1;
            continue;
        }

#if (NGX_HAVE_SETFIB)
        if (ngx_strncmp(value[n].data, "setfib=", 7) == 0) {
            lsopt.setfib = ngx_atoi(value[n].data + 7, value[n].len - 7);
            lsopt.set = 1;
            lsopt.bind = 1;

            if (lsopt.setfib == NGX_ERROR) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid setfib \"%V\"", &value[n]);
                return NGX_CONF_ERROR;
            }

            continue;
        }
#endif

#if (NGX_HAVE_TCP_FASTOPEN)
        if (ngx_strncmp(value[n].data, "fastopen=", 9) == 0) {
            lsopt.fastopen = ngx_atoi(value[n].data + 9, value[n].len - 9);
            lsopt.set = 1;
            lsopt.bind = 1;

            if (lsopt.fastopen == NGX_ERROR) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid fastopen \"%V\"", &value[n]);
                return NGX_CONF_ERROR;
            }

            continue;
        }
#endif

        if (ngx_strncmp(value[n].data, "backlog=", 8) == 0) {
            lsopt.backlog = ngx_atoi(value[n].data + 8, value[n].len - 8);
            lsopt.set = 1;
            lsopt.bind = 1;

            if (lsopt.backlog == NGX_ERROR || lsopt.backlog == 0) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid backlog \"%V\"", &value[n]);
                return NGX_CONF_ERROR;
            }

            continue;
        }

        if (ngx_strncmp(value[n].data, "rcvbuf=", 7) == 0) {
```

```
4102            size.len = value[n].len - 7;
4103            size.data = value[n].data + 7;
4104
4105            lsopt.rcvbuf = ngx_parse_size(&size);
4106            lsopt.set = 1;
4107            lsopt.bind = 1;
4108
4109            if (lsopt.rcvbuf == NGX_ERROR) {
4110                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4111                                   "invalid rcvbuf \"%V\"", &value[n]);
4112                return NGX_CONF_ERROR;
4113            }
4114
4115            continue;
4116        }
4117
4118        if (ngx_strncmp(value[n].data, "sndbuf=", 7) == 0) {
4119            size.len = value[n].len - 7;
4120            size.data = value[n].data + 7;
4121
4122            lsopt.sndbuf = ngx_parse_size(&size);
4123            lsopt.set = 1;
4124            lsopt.bind = 1;
4125
4126            if (lsopt.sndbuf == NGX_ERROR) {
4127                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4128                                   "invalid sndbuf \"%V\"", &value[n]);
4129                return NGX_CONF_ERROR;
4130            }
4131
4132            continue;
4133        }
4134
4135        if (ngx_strncmp(value[n].data, "accept_filter=", 14) == 0) {
4136 #if (NGX_HAVE_DEFERRED_ACCEPT && defined SO_ACCEPTFILTER)
4137            lsopt.accept_filter = (char *) &value[n].data[14];
4138            lsopt.set = 1;
4139            lsopt.bind = 1;
4140 #else
4141            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4142                               "accept filters \"%V\" are not supported "
4143                               "on this platform, ignored",
4144                               &value[n]);
4145 #endif
4146            continue;
4147        }
4148
4149        if (ngx_strcmp(value[n].data, "deferred") == 0) {
4150 #if (NGX_HAVE_DEFERRED_ACCEPT && defined TCP_DEFER_ACCEPT)
4151            lsopt.deferred_accept = 1;
4152            lsopt.set = 1;
4153            lsopt.bind = 1;
4154 #else
4155            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4156                               "the deferred accept is not supported "
4157                               "on this platform, ignored");
4158 #endif
4159            continue;
4160        }
4161
4162        if (ngx_strncmp(value[n].data, "ipv6only=o", 10) == 0) {
4163 #if (NGX_HAVE_INET6 && defined IPV6_V6ONLY)
4164            struct sockaddr  *sa;
4165
4166            sa = &lsopt.u.sockaddr;
4167
4168            if (sa->sa_family == AF_INET6) {
4169
4170                if (ngx_strcmp(&value[n].data[10], "n") == 0) {
4171                    lsopt.ipv6only = 1;
4172
4173                } else if (ngx_strcmp(&value[n].data[10], "ff") == 0) {
4174                    lsopt.ipv6only = 0;
4175
4176                } else {
4177                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
```

```
                                      "invalid ipv6only flags \"%s\"",
                                      &value[n].data[9]);
                    return NGX_CONF_ERROR;
                }

                lsopt.set = 1;
                lsopt.bind = 1;

            } else {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "ipv6only is not supported "
                                   "on addr \"%s\", ignored", lsopt.addr);
            }

            continue;
#else
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "ipv6only is not supported "
                               "on this platform");
            return NGX_CONF_ERROR;
#endif
        }

        if (ngx_strcmp(value[n].data, "ssl") == 0) {
#if (NGX_HTTP_SSL)
            lsopt.ssl = 1;
            continue;
#else
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "the \"ssl\" parameter requires "
                               "ngx_http_ssl_module");
            return NGX_CONF_ERROR;
#endif
        }

        if (ngx_strcmp(value[n].data, "spdy") == 0) {
#if (NGX_HTTP_SPDY)
            lsopt.spdy = 1;
            continue;
#else
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "the \"spdy\" parameter requires "
                               "ngx_http_spdy_module");
            return NGX_CONF_ERROR;
#endif
        }

        if (ngx_strncmp(value[n].data, "so_keepalive=", 13) == 0) {

            if (ngx_strcmp(&value[n].data[13], "on") == 0) {
                lsopt.so_keepalive = 1;

            } else if (ngx_strcmp(&value[n].data[13], "off") == 0) {
                lsopt.so_keepalive = 2;

            } else {

#if (NGX_HAVE_KEEPALIVE_TUNABLE)
                u_char     *p, *end;
                ngx_str_t   s;

                end = value[n].data + value[n].len;
                s.data = value[n].data + 13;

                p = ngx_strlchr(s.data, end, ':');
                if (p == NULL) {
                    p = end;
                }

                if (p > s.data) {
                    s.len = p - s.data;

                    lsopt.tcp_keepidle = ngx_parse_time(&s, 1);
                    if (lsopt.tcp_keepidle == (time_t) NGX_ERROR) {
                        goto invalid_so_keepalive;
                    }
```

```
4254                }
4255
4256                s.data = (p < end) ? (p + 1) : end;
4257
4258                p = ngx_strlchr(s.data, end, ':');
4259                if (p == NULL) {
4260                    p = end;
4261                }
4262
4263                if (p > s.data) {
4264                    s.len = p - s.data;
4265
4266                    lsopt.tcp_keepintvl = ngx_parse_time(&s, 1);
4267                    if (lsopt.tcp_keepintvl == (time_t) NGX_ERROR) {
4268                        goto invalid_so_keepalive;
4269                    }
4270                }
4271
4272                s.data = (p < end) ? (p + 1) : end;
4273
4274                if (s.data < end) {
4275                    s.len = end - s.data;
4276
4277                    lsopt.tcp_keepcnt = ngx_atoi(s.data, s.len);
4278                    if (lsopt.tcp_keepcnt == NGX_ERROR) {
4279                        goto invalid_so_keepalive;
4280                    }
4281                }
4282
4283                if (lsopt.tcp_keepidle == 0 && lsopt.tcp_keepintvl == 0
4284                    && lsopt.tcp_keepcnt == 0)
4285                {
4286                    goto invalid_so_keepalive;
4287                }
4288
4289                lsopt.so_keepalive = 1;
4290
4291 #else
4292
4293                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4294                                   "the \"so_keepalive\" parameter accepts "
4295                                   "only \"on\" or \"off\" on this platform");
4296                return NGX_CONF_ERROR;
4297
4298 #endif
4299            }
4300
4301            lsopt.set = 1;
4302            lsopt.bind = 1;
4303
4304            continue;
4305
4306 #if (NGX_HAVE_KEEPALIVE_TUNABLE)
4307        invalid_so_keepalive:
4308
4309            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4310                               "invalid so_keepalive value: \"%s\"",
4311                               &value[n].data[13]);
4312            return NGX_CONF_ERROR;
4313 #endif
4314        }
4315
4316        if (ngx_strcmp(value[n].data, "proxy_protocol") == 0) {
4317            lsopt.proxy_protocol = 1;
4318            continue;
4319        }
4320
4321        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4322                           "invalid parameter \"%V\"", &value[n]);
4323        return NGX_CONF_ERROR;
4324    }
4325
4326    if (ngx_http_add_listen(cf, cscf, &lsopt) == NGX_OK) {
4327        return NGX_CONF_OK;
4328    }
4329
```

```
4330      return NGX_CONF_ERROR;
4331  }
4332
4333
4334  static char *
4335  ngx_http_core_server_name(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
4336  {
4337      ngx_http_core_srv_conf_t *cscf = conf;
4338
4339      u_char                   ch;
4340      ngx_str_t               *value;
4341      ngx_uint_t               i;
4342      ngx_http_server_name_t  *sn;
4343
4344      value = cf->args->elts;
4345
4346      for (i = 1; i < cf->args->nelts; i++) {
4347
4348          ch = value[i].data[0];
4349
4350          if ((ch == '*' && (value[i].len < 3 || value[i].data[1] != '.'))
4351              || (ch == '.' && value[i].len < 2))
4352          {
4353              ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4354                                 "server name \"%V\" is invalid", &value[i]);
4355              return NGX_CONF_ERROR;
4356          }
4357
4358          if (ngx_strchr(value[i].data, '/')) {
4359              ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
4360                                 "server name \"%V\" has suspicious symbols",
4361                                 &value[i]);
4362          }
4363
4364          sn = ngx_array_push(&cscf->server_names);
4365          if (sn == NULL) {
4366              return NGX_CONF_ERROR;
4367          }
4368
4369  #if (NGX_PCRE)
4370          sn->regex = NULL;
4371  #endif
4372          sn->server = cscf;
4373
4374          if (ngx_strcasecmp(value[i].data, (u_char *) "$hostname") == 0) {
4375              sn->name = cf->cycle->hostname;
4376
4377          } else {
4378              sn->name = value[i];
4379          }
4380
4381          if (value[i].data[0] != '~') {
4382              ngx_strlow(sn->name.data, sn->name.data, sn->name.len);
4383              continue;
4384          }
4385
4386  #if (NGX_PCRE)
4387          {
4388          u_char               *p;
4389          ngx_regex_compile_t   rc;
4390          u_char                errstr[NGX_MAX_CONF_ERRSTR];
4391
4392          if (value[i].len == 1) {
4393              ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4394                                 "empty regex in server name \"%V\"", &value[i]);
4395              return NGX_CONF_ERROR;
4396          }
4397
4398          value[i].len--;
4399          value[i].data++;
4400
4401          ngx_memzero(&rc, sizeof(ngx_regex_compile_t));
4402
4403          rc.pattern = value[i];
4404          rc.err.len = NGX_MAX_CONF_ERRSTR;
4405          rc.err.data = errstr;
```

```
4406
4407        for (p = value[i].data; p < value[i].data + value[i].len; p++) {
4408            if (*p >= 'A' && *p <= 'Z') {
4409                rc.options = NGX_REGEX_CASELESS;
4410                break;
4411            }
4412        }
4413
4414        sn->regex = ngx_http_regex_compile(cf, &rc);
4415        if (sn->regex == NULL) {
4416            return NGX_CONF_ERROR;
4417        }
4418
4419        sn->name = value[i];
4420        cscf->captures = (rc.captures > 0);
4421        }
4422 #else
4423        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4424                           "using regex \"%V\" "
4425                           "requires PCRE library", &value[i]);
4426
4427        return NGX_CONF_ERROR;
4428 #endif
4429    }
4430
4431    return NGX_CONF_OK;
4432 }
4433
4434
4435 static char *
4436 ngx_http_core_root(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
4437 {
4438    ngx_http_core_loc_conf_t *clcf = conf;
4439
4440    ngx_str_t                  *value;
4441    ngx_int_t                   alias;
4442    ngx_uint_t                  n;
4443    ngx_http_script_compile_t   sc;
4444
4445    alias = (cmd->name.len == sizeof("alias") - 1) ? 1 : 0;
4446
4447    if (clcf->root.data) {
4448
4449        if ((clcf->alias != 0) == alias) {
4450            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4451                               "\"%V\" directive is duplicate",
4452                               &cmd->name);
4453        } else {
4454            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4455                               "\"%V\" directive is duplicate, "
4456                               "\"%s\" directive was specified earlier",
4457                               &cmd->name, clcf->alias ? "alias" : "root");
4458        }
4459
4460        return NGX_CONF_ERROR;
4461    }
4462
4463    if (clcf->named && alias) {
4464        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4465                           "the \"alias\" directive cannot be used "
4466                           "inside the named location");
4467
4468        return NGX_CONF_ERROR;
4469    }
4470
4471    value = cf->args->elts;
4472
4473    if (ngx_strstr(value[1].data, "$document_root")
4474        || ngx_strstr(value[1].data, "${document_root}"))
4475    {
4476        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4477                           "the $document_root variable cannot be used "
4478                           "in the \"%V\" directive",
4479                           &cmd->name);
4480
4481        return NGX_CONF_ERROR;
```

```
    }

    if (ngx_strstr(value[1].data, "$realpath_root")
        || ngx_strstr(value[1].data, "${realpath_root}"))
    {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "the $realpath_root variable cannot be used "
                           "in the \"%V\" directive",
                           &cmd->name);

        return NGX_CONF_ERROR;
    }

    clcf->alias = alias ? clcf->name.len : 0;
    clcf->root = value[1];

    if (!alias && clcf->root.data[clcf->root.len - 1] == '/') {
        clcf->root.len--;
    }

    if (clcf->root.data[0] != '$') {
        if (ngx_conf_full_name(cf->cycle, &clcf->root, 0) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

    n = ngx_http_script_variables_count(&clcf->root);

    ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
    sc.variables = n;

#if (NGX_PCRE)
    if (alias && clcf->regex) {
        clcf->alias = NGX_MAX_SIZE_T_VALUE;
        n = 1;
    }
#endif

    if (n) {
        sc.cf = cf;
        sc.source = &clcf->root;
        sc.lengths = &clcf->root_lengths;
        sc.values = &clcf->root_values;
        sc.complete_lengths = 1;
        sc.complete_values = 1;

        if (ngx_http_script_compile(&sc) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

    return NGX_CONF_OK;
}


static ngx_http_method_name_t  ngx_methods_names[] = {
   { (u_char *) "GET",       (uint32_t) ~NGX_HTTP_GET },
   { (u_char *) "HEAD",      (uint32_t) ~NGX_HTTP_HEAD },
   { (u_char *) "POST",      (uint32_t) ~NGX_HTTP_POST },
   { (u_char *) "PUT",       (uint32_t) ~NGX_HTTP_PUT },
   { (u_char *) "DELETE",    (uint32_t) ~NGX_HTTP_DELETE },
   { (u_char *) "MKCOL",     (uint32_t) ~NGX_HTTP_MKCOL },
   { (u_char *) "COPY",      (uint32_t) ~NGX_HTTP_COPY },
   { (u_char *) "MOVE",      (uint32_t) ~NGX_HTTP_MOVE },
   { (u_char *) "OPTIONS",   (uint32_t) ~NGX_HTTP_OPTIONS },
   { (u_char *) "PROPFIND",  (uint32_t) ~NGX_HTTP_PROPFIND },
   { (u_char *) "PROPPATCH", (uint32_t) ~NGX_HTTP_PROPPATCH },
   { (u_char *) "LOCK",      (uint32_t) ~NGX_HTTP_LOCK },
   { (u_char *) "UNLOCK",    (uint32_t) ~NGX_HTTP_UNLOCK },
   { (u_char *) "PATCH",     (uint32_t) ~NGX_HTTP_PATCH },
   { NULL, 0 }
};


static char *
ngx_http_core_limit_except(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
```

```
{
    ngx_http_core_loc_conf_t *pclcf = conf;

    char                     *rv;
    void                     *mconf;
    ngx_str_t                *value;
    ngx_uint_t                i;
    ngx_conf_t                save;
    ngx_http_module_t        *module;
    ngx_http_conf_ctx_t      *ctx, *pctx;
    ngx_http_method_name_t   *name;
    ngx_http_core_loc_conf_t  *clcf;

    if (pclcf->limit_except) {
        return "duplicate";
    }

    pclcf->limit_except = 0xffffffff;

    value = cf->args->elts;

    for (i = 1; i < cf->args->nelts; i++) {
        for (name = ngx_methods_names; name->name; name++) {

            if (ngx_strcasecmp(value[i].data, name->name) == 0) {
                pclcf->limit_except &= name->method;
                goto next;
            }
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid method \"%V\"", &value[i]);
        return NGX_CONF_ERROR;

    next:
        continue;
    }

    if (!(pclcf->limit_except & NGX_HTTP_GET)) {
        pclcf->limit_except &= (uint32_t) ~NGX_HTTP_HEAD;
    }

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    pctx = cf->ctx;
    ctx->main_conf = pctx->main_conf;
    ctx->srv_conf = pctx->srv_conf;

    ctx->loc_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->loc_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_HTTP_MODULE) {
            continue;
        }

        module = ngx_modules[i]->ctx;

        if (module->create_loc_conf) {

            mconf = module->create_loc_conf(cf);
            if (mconf == NULL) {
                 return NGX_CONF_ERROR;
            }

            ctx->loc_conf[ngx_modules[i]->ctx_index] = mconf;
        }
    }


    clcf = ctx->loc_conf[ngx_http_core_module.ctx_index];
```

```
    pclcf->limit_except_loc_conf = ctx->loc_conf;
    clcf->loc_conf = ctx->loc_conf;
    clcf->name = pclcf->name;
    clcf->noname = 1;
    clcf->lmt_excpt = 1;

    if (ngx_http_add_location(cf, &pclcf->locations, clcf) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    save = *cf;
    cf->ctx = ctx;
    cf->cmd_type = NGX_HTTP_LMT_CONF;

    rv = ngx_conf_parse(cf, NULL);

    *cf = save;

    return rv;
}


static char *
ngx_http_core_directio(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_core_loc_conf_t *clcf = conf;

    ngx_str_t  *value;

    if (clcf->directio != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    if (ngx_strcmp(value[1].data, "off") == 0) {
        clcf->directio = NGX_OPEN_FILE_DIRECTIO_OFF;
        return NGX_CONF_OK;
    }

    clcf->directio = ngx_parse_offset(&value[1]);
    if (clcf->directio == (off_t) NGX_ERROR) {
        return "invalid value";
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_core_error_page(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_core_loc_conf_t *clcf = conf;

    u_char                           *p;
    ngx_int_t                         overwrite;
    ngx_str_t                        *value, uri, args;
    ngx_uint_t                        i, n;
    ngx_http_err_page_t              *err;
    ngx_http_complex_value_t          cv;
    ngx_http_compile_complex_value_t  ccv;

    if (clcf->error_pages == NULL) {
        clcf->error_pages = ngx_array_create(cf->pool, 4,
                                             sizeof(ngx_http_err_page_t));
        if (clcf->error_pages == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    value = cf->args->elts;

    i = cf->args->nelts - 2;

    if (value[i].data[0] == '=') {
        if (i == 1) {
```

```
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid value \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        if (value[i].len > 1) {
            overwrite = ngx_atoi(&value[i].data[1], value[i].len - 1);

            if (overwrite == NGX_ERROR) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid value \"%V\"", &value[i]);
                return NGX_CONF_ERROR;
            }

        } else {
            overwrite = 0;
        }

        n = 2;

    } else {
        overwrite = -1;
        n = 1;
    }

    uri = value[cf->args->nelts - 1];

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &uri;
    ccv.complex_value = &cv;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    ngx_str_null(&args);

    if (cv.lengths == NULL && uri.len && uri.data[0] == '/') {
        p = (u_char *) ngx_strchr(uri.data, '?');

        if (p) {
            cv.value.len = p - uri.data;
            cv.value.data = uri.data;
            p++;
            args.len = (uri.data + uri.len) - p;
            args.data = p;
        }
    }

    for (i = 1; i < cf->args->nelts - n; i++) {
        err = ngx_array_push(clcf->error_pages);
        if (err == NULL) {
            return NGX_CONF_ERROR;
        }

        err->status = ngx_atoi(value[i].data, value[i].len);

        if (err->status == NGX_ERROR || err->status == 499) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid value \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        if (err->status < 300 || err->status > 599) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "value \"%V\" must be between 300 and 599",
                               &value[i]);
            return NGX_CONF_ERROR;
        }

        err->overwrite = overwrite;

        if (overwrite == -1) {
            switch (err->status) {
```

```
                case NGX_HTTP_TO_HTTPS:
                case NGX_HTTPS_CERT_ERROR:
                case NGX_HTTPS_NO_CERT:
                    err->overwrite = NGX_HTTP_BAD_REQUEST;
                default:
                    break;
            }
        }

        err->value = cv;
        err->args = args;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_core_try_files(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_core_loc_conf_t *clcf = conf;

    ngx_str_t                  *value;
    ngx_int_t                   code;
    ngx_uint_t                  i, n;
    ngx_http_try_file_t        *tf;
    ngx_http_script_compile_t   sc;
    ngx_http_core_main_conf_t  *cmcf;

    if (clcf->try_files) {
        return "is duplicate";
    }

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    cmcf->try_files = 1;

    tf = ngx_pcalloc(cf->pool, cf->args->nelts * sizeof(ngx_http_try_file_t));
    if (tf == NULL) {
        return NGX_CONF_ERROR;
    }

    clcf->try_files = tf;

    value = cf->args->elts;

    for (i = 0; i < cf->args->nelts - 1; i++) {

        tf[i].name = value[i + 1];

        if (tf[i].name.len > 0
            && tf[i].name.data[tf[i].name.len - 1] == '/'
            && i + 2 < cf->args->nelts)
        {
            tf[i].test_dir = 1;
            tf[i].name.len--;
            tf[i].name.data[tf[i].name.len] = '\0';
        }

        n = ngx_http_script_variables_count(&tf[i].name);

        if (n) {
            ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

            sc.cf = cf;
            sc.source = &tf[i].name;
            sc.lengths = &tf[i].lengths;
            sc.values = &tf[i].values;
            sc.variables = n;
            sc.complete_lengths = 1;
            sc.complete_values = 1;

            if (ngx_http_script_compile(&sc) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

```

```
        } else {
            /* add trailing '\0' to length */
            tf[i].name.len++;
        }
    }

    if (tf[i - 1].name.data[0] == '=') {

        code = ngx_atoi(tf[i - 1].name.data + 1, tf[i - 1].name.len - 2);

        if (code == NGX_ERROR || code > 999) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid code \"%*s\"",
                               tf[i - 1].name.len - 1, tf[i - 1].name.data);
            return NGX_CONF_ERROR;
        }

        tf[i].code = code;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_core_open_file_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_core_loc_conf_t *clcf = conf;

    time_t       inactive;
    ngx_str_t   *value, s;
    ngx_int_t    max;
    ngx_uint_t   i;

    if (clcf->open_file_cache != NGX_CONF_UNSET_PTR) {
        return "is duplicate";
    }

    value = cf->args->elts;

    max = 0;
    inactive = 60;

    for (i = 1; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "max=", 4) == 0) {

            max = ngx_atoi(value[i].data + 4, value[i].len - 4);
            if (max <= 0) {
                goto failed;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "inactive=", 9) == 0) {

            s.len = value[i].len - 9;
            s.data = value[i].data + 9;

            inactive = ngx_parse_time(&s, 1);
            if (inactive == (time_t) NGX_ERROR) {
                goto failed;
            }

            continue;
        }

        if (ngx_strcmp(value[i].data, "off") == 0) {

            clcf->open_file_cache = NULL;

            continue;
        }

    failed:
```

```
4938
4939            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4940                               "invalid \"open_file_cache\" parameter \"%V\"",
4941                               &value[i]);
4942            return NGX_CONF_ERROR;
4943        }
4944
4945        if (clcf->open_file_cache == NULL) {
4946            return NGX_CONF_OK;
4947        }
4948
4949        if (max == 0) {
4950            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
4951                               "\"open_file_cache\" must have the \"max\" parameter");
4952            return NGX_CONF_ERROR;
4953        }
4954
4955        clcf->open_file_cache = ngx_open_file_cache_init(cf->pool, max, inactive);
4956        if (clcf->open_file_cache) {
4957            return NGX_CONF_OK;
4958        }
4959
4960        return NGX_CONF_ERROR;
4961    }
4962
4963
4964    static char *
4965    ngx_http_core_error_log(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
4966    {
4967        ngx_http_core_loc_conf_t *clcf = conf;
4968
4969        return ngx_log_set_log(cf, &clcf->error_log);
4970    }
4971
4972
4973    static char *
4974    ngx_http_core_keepalive(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
4975    {
4976        ngx_http_core_loc_conf_t *clcf = conf;
4977
4978        ngx_str_t  *value;
4979
4980        if (clcf->keepalive_timeout != NGX_CONF_UNSET_MSEC) {
4981            return "is duplicate";
4982        }
4983
4984        value = cf->args->elts;
4985
4986        clcf->keepalive_timeout = ngx_parse_time(&value[1], 0);
4987
4988        if (clcf->keepalive_timeout == (ngx_msec_t) NGX_ERROR) {
4989            return "invalid value";
4990        }
4991
4992        if (cf->args->nelts == 2) {
4993            return NGX_CONF_OK;
4994        }
4995
4996        clcf->keepalive_header = ngx_parse_time(&value[2], 1);
4997
4998        if (clcf->keepalive_header == (time_t) NGX_ERROR) {
4999            return "invalid value";
5000        }
5001
5002        return NGX_CONF_OK;
5003    }
5004
5005
5006    static char *
5007    ngx_http_core_internal(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
5008    {
5009        ngx_http_core_loc_conf_t *clcf = conf;
5010
5011        if (clcf->internal != NGX_CONF_UNSET) {
5012            return "is duplicate";
5013        }
```

```
5014
5015        clcf->internal = 1;
5016
5017        return NGX_CONF_OK;
5018    }
5019
5020
5021    static char *
5022    ngx_http_core_resolver(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
5023    {
5024        ngx_http_core_loc_conf_t  *clcf = conf;
5025
5026        ngx_str_t  *value;
5027
5028        if (clcf->resolver) {
5029            return "is duplicate";
5030        }
5031
5032        value = cf->args->elts;
5033
5034        clcf->resolver = ngx_resolver_create(cf, &value[1], cf->args->nelts - 1);
5035        if (clcf->resolver == NULL) {
5036            return NGX_CONF_ERROR;
5037        }
5038
5039        return NGX_CONF_OK;
5040    }
5041
5042
5043    #if (NGX_HTTP_GZIP)
5044
5045    static char *
5046    ngx_http_gzip_disable(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
5047    {
5048        ngx_http_core_loc_conf_t  *clcf = conf;
5049
5050    #if (NGX_PCRE)
5051
5052        ngx_str_t            *value;
5053        ngx_uint_t            i;
5054        ngx_regex_elt_t      *re;
5055        ngx_regex_compile_t   rc;
5056        u_char                errstr[NGX_MAX_CONF_ERRSTR];
5057
5058        if (clcf->gzip_disable == NGX_CONF_UNSET_PTR) {
5059            clcf->gzip_disable = ngx_array_create(cf->pool, 2,
5060                                                  sizeof(ngx_regex_elt_t));
5061            if (clcf->gzip_disable == NULL) {
5062                return NGX_CONF_ERROR;
5063            }
5064        }
5065
5066        value = cf->args->elts;
5067
5068        ngx_memzero(&rc, sizeof(ngx_regex_compile_t));
5069
5070        rc.pool = cf->pool;
5071        rc.err.len = NGX_MAX_CONF_ERRSTR;
5072        rc.err.data = errstr;
5073
5074        for (i = 1; i < cf->args->nelts; i++) {
5075
5076            if (ngx_strcmp(value[i].data, "msie6") == 0) {
5077                clcf->gzip_disable_msie6 = 1;
5078                continue;
5079            }
5080
5081    #if (NGX_HTTP_DEGRADATION)
5082
5083            if (ngx_strcmp(value[i].data, "degradation") == 0) {
5084                clcf->gzip_disable_degradation = 1;
5085                continue;
5086            }
5087
5088    #endif
5089
```

```
5090         re = ngx_array_push(clcf->gzip_disable);
5091         if (re == NULL) {
5092             return NGX_CONF_ERROR;
5093         }
5094
5095         rc.pattern = value[i];
5096         rc.options = NGX_REGEX_CASELESS;
5097
5098         if (ngx_regex_compile(&rc) != NGX_OK) {
5099             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "%V", &rc.err);
5100             return NGX_CONF_ERROR;
5101         }
5102
5103         re->regex = rc.regex;
5104         re->name = value[i].data;
5105     }
5106
5107     return NGX_CONF_OK;
5108
5109 #else
5110     ngx_str_t   *value;
5111     ngx_uint_t   i;
5112
5113     value = cf->args->elts;
5114
5115     for (i = 1; i < cf->args->nelts; i++) {
5116         if (ngx_strcmp(value[i].data, "msie6") == 0) {
5117             clcf->gzip_disable_msie6 = 1;
5118             continue;
5119         }
5120
5121 #if (NGX_HTTP_DEGRADATION)
5122
5123         if (ngx_strcmp(value[i].data, "degradation") == 0) {
5124             clcf->gzip_disable_degradation = 1;
5125             continue;
5126         }
5127
5128 #endif
5129
5130         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5131                            "without PCRE library \"gzip_disable\" supports "
5132                            "builtin \"msie6\" and \"degradation\" mask only");
5133
5134         return NGX_CONF_ERROR;
5135     }
5136
5137     return NGX_CONF_OK;
5138
5139 #endif
5140 }
5141
5142 #endif
5143
5144
5145 #if (NGX_HAVE_OPENAT)
5146
5147 static char *
5148 ngx_http_disable_symlinks(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
5149 {
5150     ngx_http_core_loc_conf_t *clcf = conf;
5151
5152     ngx_str_t                         *value;
5153     ngx_uint_t                         i;
5154     ngx_http_compile_complex_value_t   ccv;
5155
5156     if (clcf->disable_symlinks != NGX_CONF_UNSET_UINT) {
5157         return "is duplicate";
5158     }
5159
5160     value = cf->args->elts;
5161
5162     for (i = 1; i < cf->args->nelts; i++) {
5163
5164         if (ngx_strcmp(value[i].data, "off") == 0) {
5165             clcf->disable_symlinks = NGX_DISABLE_SYMLINKS_OFF;
```

```
            continue;
        }

        if (ngx_strcmp(value[i].data, "if_not_owner") == 0) {
            clcf->disable_symlinks = NGX_DISABLE_SYMLINKS_NOTOWNER;
            continue;
        }

        if (ngx_strcmp(value[i].data, "on") == 0) {
            clcf->disable_symlinks = NGX_DISABLE_SYMLINKS_ON;
            continue;
        }

        if (ngx_strncmp(value[i].data, "from=", 5) == 0) {
            value[i].len -= 5;
            value[i].data += 5;

            ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

            ccv.cf = cf;
            ccv.value = &value[i];
            ccv.complex_value = ngx_palloc(cf->pool,
                                           sizeof(ngx_http_complex_value_t));
            if (ccv.complex_value == NULL) {
                return NGX_CONF_ERROR;
            }

            if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

            clcf->disable_symlinks_from = ccv.complex_value;

            continue;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid parameter \"%V\"", &value[i]);
        return NGX_CONF_ERROR;
    }

    if (clcf->disable_symlinks == NGX_CONF_UNSET_UINT) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"%V\" must have \"off\", \"on\" "
                           "or \"if_not_owner\" parameter",
                           &cmd->name);
        return NGX_CONF_ERROR;
    }

    if (cf->args->nelts == 2) {
        clcf->disable_symlinks_from = NULL;
        return NGX_CONF_OK;
    }

    if (clcf->disable_symlinks_from == NGX_CONF_UNSET_PTR) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "duplicate parameters \"%V %V\"",
                           &value[1], &value[2]);
        return NGX_CONF_ERROR;
    }

    if (clcf->disable_symlinks == NGX_DISABLE_SYMLINKS_OFF) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"from=\" cannot be used with \"off\" parameter");
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}

#endif


static char *
ngx_http_core_lowat_check(ngx_conf_t *cf, void *post, void *data)
{
```

```
5242 #if (NGX_FREEBSD)
5243     ssize_t *np = data;
5244 
5245     if ((u_long) *np >= ngx_freebsd_net_inet_tcp_sendspace) {
5246         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5247                            "\"send_lowat\" must be less than %d "
5248                            "(sysctl net.inet.tcp.sendspace)",
5249                            ngx_freebsd_net_inet_tcp_sendspace);
5250 
5251         return NGX_CONF_ERROR;
5252     }
5253 
5254 #elif !(NGX_HAVE_SO_SNDLOWAT)
5255     ssize_t *np = data;
5256 
5257     ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
5258                        "\"send_lowat\" is not supported, ignored");
5259 
5260     *np = 0;
5261 
5262 #endif
5263 
5264     return NGX_CONF_OK;
5265 }
5266 
5267 
5268 static char *
5269 ngx_http_core_pool_size(ngx_conf_t *cf, void *post, void *data)
5270 {
5271     size_t *sp = data;
5272 
5273     if (*sp < NGX_MIN_POOL_SIZE) {
5274         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5275                            "the pool size must be no less than %uz",
5276                            NGX_MIN_POOL_SIZE);
5277         return NGX_CONF_ERROR;
5278     }
5279 
5280     if (*sp % NGX_POOL_ALIGNMENT) {
5281         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5282                            "the pool size must be a multiple of %uz",
5283                            NGX_POOL_ALIGNMENT);
5284         return NGX_CONF_ERROR;
5285     }
5286 
5287     return NGX_CONF_OK;
5288 }
```

One Level Up Top Level

src/http/ngx_http_request_body.c - nginx-1.7.10

Functions defined

- ngx_http_discard_request_body
- ngx_http_discard_request_body_filter
- ngx_http_discarded_request_body_handler
- ngx_http_do_read_client_request_body
- ngx_http_read_client_request_body
- ngx_http_read_client_request_body_handler
- ngx_http_read_discarded_request_body
- ngx_http_request_body_chunked_filter
- ngx_http_request_body_filter
- ngx_http_request_body_length_filter
- ngx_http_request_body_save_filter
- ngx_http_test_expect
- ngx_http_write_request_body

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static void ngx_http_read_client_request_body_handler(ngx_http_request_t *r);
14  static ngx_int_t ngx_http_do_read_client_request_body(ngx_http_request_t *r);
15  static ngx_int_t ngx_http_write_request_body(ngx_http_request_t *r);
16  static ngx_int_t ngx_http_read_discarded_request_body(ngx_http_request_t *r);
17  static ngx_int_t ngx_http_discard_request_body_filter(ngx_http_request_t *r,
18      ngx_buf_t *b);
19  static ngx_int_t ngx_http_test_expect(ngx_http_request_t *r);
20
21  static ngx_int_t ngx_http_request_body_filter(ngx_http_request_t *r,
22      ngx_chain_t *in);
23  static ngx_int_t ngx_http_request_body_length_filter(ngx_http_request_t *r,
24      ngx_chain_t *in);
25  static ngx_int_t ngx_http_request_body_chunked_filter(ngx_http_request_t *r,
26      ngx_chain_t *in);
27  static ngx_int_t ngx_http_request_body_save_filter(ngx_http_request_t *r,
28      ngx_chain_t *in);
29
30
31  ngx_int_t
32  ngx_http_read_client_request_body(ngx_http_request_t *r,
33      ngx_http_client_body_handler_pt post_handler)
34  {
35      size_t                     preread;
```

```
    ssize_t                   size;
    ngx_int_t                 rc;
    ngx_buf_t                *b;
    ngx_chain_t               out, *cl;
    ngx_http_request_body_t   *rb;
    ngx_http_core_loc_conf_t  *clcf;

    r->main->count++;

#if (NGX_HTTP_SPDY)
    if (r->spdy_stream && r == r->main) {
        rc = ngx_http_spdy_read_request_body(r, post_handler);
        goto done;
    }
#endif

    if (r != r->main || r->request_body || r->discard_body) {
        post_handler(r);
        return NGX_OK;
    }

    if (ngx_http_test_expect(r) != NGX_OK) {
        rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        goto done;
    }

    rb = ngx_pcalloc(r->pool, sizeof(ngx_http_request_body_t));
    if (rb == NULL) {
        rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        goto done;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     rb->bufs = NULL;
     *     rb->buf = NULL;
     *     rb->free = NULL;
     *     rb->busy = NULL;
     *     rb->chunked = NULL;
     */

    rb->rest = -1;
    rb->post_handler = post_handler;

    r->request_body = rb;

    if (r->headers_in.content_length_n < 0 && !r->headers_in.chunked) {
        post_handler(r);
        return NGX_OK;
    }

    preread = r->header_in->last - r->header_in->pos;

    if (preread) {

        /* there is the pre-read part of the request body */

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http client request body preread %uz", preread);

        out.buf = r->header_in;
        out.next = NULL;

         rc = ngx_http_request_body_filter(r, &out);

         if (rc != NGX_OK) {
             goto done;
         }

         r->request_length += preread - (r->header_in->last - r->header_in->pos);

         if (!r->headers_in.chunked
             && rb->rest > 0
             && rb->rest <= (off_t) (r->header_in->end - r->header_in->last))
         {
```

```
112             /* the whole request body may be placed in r->header_in */
113
114             b = ngx_calloc_buf(r->pool);
115             if (b == NULL) {
116                 rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
117                 goto done;
118             }
119
120             b->temporary = 1;
121             b->start = r->header_in->pos;
122             b->pos = r->header_in->pos;
123             b->last = r->header_in->last;
124             b->end = r->header_in->end;
125
126             rb->buf = b;
127
128             r->read_event_handler = ngx_http_read_client_request_body_handler;
129             r->write_event_handler = ngx_http_request_empty_handler;
130
131             rc = ngx_http_do_read_client_request_body(r);
132             goto done;
133         }
134
135     } else {
136         /* set rb->rest */
137
138         if (ngx_http_request_body_filter(r, NULL) != NGX_OK) {
139             rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
140             goto done;
141         }
142     }
143
144     if (rb->rest == 0) {
145         /* the whole request body was pre-read */
146
147         if (r->request_body_in_file_only) {
148             if (ngx_http_write_request_body(r) != NGX_OK) {
149                 rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
150                 goto done;
151             }
152
153             if (rb->temp_file->file.offset != 0) {
154
155                 cl = ngx_chain_get_free_buf(r->pool, &rb->free);
156                 if (cl == NULL) {
157                     rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
158                     goto done;
159                 }
160
161                 b = cl->buf;
162
163                 ngx_memzero(b, sizeof(ngx_buf_t));
164
165                 b->in_file = 1;
166                 b->file_last = rb->temp_file->file.offset;
167                 b->file = &rb->temp_file->file;
168
169                 rb->bufs = cl;
170
171             } else {
172                 rb->bufs = NULL;
173             }
174         }
175
176         post_handler(r);
177
178         return NGX_OK;
179     }
180
181     if (rb->rest < 0) {
182         ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
183                       "negative request body rest");
184         rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
185         goto done;
186     }
187
```

```
    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    size = clcf->client_body_buffer_size;
    size += size >> 2;

    /* TODO: honor r->request_body_in_single_buf */

    if (!r->headers_in.chunked && rb->rest < size) {
        size = (ssize_t) rb->rest;

        if (r->request_body_in_single_buf) {
            size += preread;
        }

    } else {
        size = clcf->client_body_buffer_size;
    }

    rb->buf = ngx_create_temp_buf(r->pool, size);
    if (rb->buf == NULL) {
        rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        goto done;
    }

    r->read_event_handler = ngx_http_read_client_request_body_handler;
    r->write_event_handler = ngx_http_request_empty_handler;

    rc = ngx_http_do_read_client_request_body(r);

done:

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        r->main->count--;
    }

    return rc;
}


static void
ngx_http_read_client_request_body_handler(ngx_http_request_t *r)
{
    ngx_int_t  rc;

    if (r->connection->read->timedout) {
        r->connection->timedout = 1;
        ngx_http_finalize_request(r, NGX_HTTP_REQUEST_TIME_OUT);
        return;
    }

    rc = ngx_http_do_read_client_request_body(r);

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        ngx_http_finalize_request(r, rc);
    }
}


static ngx_int_t
ngx_http_do_read_client_request_body(ngx_http_request_t *r)
{
    off_t                     rest;
    size_t                    size;
    ssize_t                   n;
    ngx_int_t                 rc;
    ngx_buf_t                *b;
    ngx_chain_t              *cl, out;
    ngx_connection_t         *c;
    ngx_http_request_body_t  *rb;
    ngx_http_core_loc_conf_t  *clcf;

    c = r->connection;
    rb = r->request_body;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http read client request body");
```

```
264
265     for ( ;; ) {
266         for ( ;; ) {
267             if (rb->buf->last == rb->buf->end) {
268
269                 /* pass buffer to request body filter chain */
270
271                 out.buf = rb->buf;
272                 out.next = NULL;
273
274                 rc = ngx_http_request_body_filter(r, &out);
275
276                 if (rc != NGX_OK) {
277                     return rc;
278                 }
279
280                 /* write to file */
281
282                 if (ngx_http_write_request_body(r) != NGX_OK) {
283                     return NGX_HTTP_INTERNAL_SERVER_ERROR;
284                 }
285
286                 /* update chains */
287
288                 rc = ngx_http_request_body_filter(r, NULL);
289
290                 if (rc != NGX_OK) {
291                     return rc;
292                 }
293
294                 if (rb->busy != NULL) {
295                     return NGX_HTTP_INTERNAL_SERVER_ERROR;
296                 }
297
298                 rb->buf->pos = rb->buf->start;
299                 rb->buf->last = rb->buf->start;
300             }
301
302             size = rb->buf->end - rb->buf->last;
303             rest = rb->rest - (rb->buf->last - rb->buf->pos);
304
305             if ((off_t) size > rest) {
306                 size = (size_t) rest;
307             }
308
309             n = c->recv(c, rb->buf->last, size);
310
311             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
312                            "http client request body recv %z", n);
313
314             if (n == NGX_AGAIN) {
315                 break;
316             }
317
318             if (n == 0) {
319                 ngx_log_error(NGX_LOG_INFO, c->log, 0,
320                               "client prematurely closed connection");
321             }
322
323             if (n == 0 || n == NGX_ERROR) {
324                 c->error = 1;
325                 return NGX_HTTP_BAD_REQUEST;
326             }
327
328             rb->buf->last += n;
329             r->request_length += n;
330
331             if (n == rest) {
332                 /* pass buffer to request body filter chain */
333
334                 out.buf = rb->buf;
335                 out.next = NULL;
336
337                 rc = ngx_http_request_body_filter(r, &out);
338
339                 if (rc != NGX_OK) {
```

```
                    return rc;
                }
            }

            if (rb->rest == 0) {
                break;
            }

            if (rb->buf->last < rb->buf->end) {
                break;
            }
        }

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                       "http client request body rest %O", rb->rest);

        if (rb->rest == 0) {
            break;
        }

        if (!c->read->ready) {
            clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
            ngx_add_timer(c->read, clcf->client_body_timeout);

            if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
                return NGX_HTTP_INTERNAL_SERVER_ERROR;
            }

            return NGX_AGAIN;
        }
    }

    if (c->read->timer_set) {
        ngx_del_timer(c->read);
    }

    if (rb->temp_file || r->request_body_in_file_only) {

        /* save the last part */

        if (ngx_http_write_request_body(r) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        if (rb->temp_file->file.offset != 0) {

            cl = ngx_chain_get_free_buf(r->pool, &rb->free);
            if (cl == NULL) {
                return NGX_HTTP_INTERNAL_SERVER_ERROR;
            }

            b = cl->buf;

            ngx_memzero(b, sizeof(ngx_buf_t));

            b->in_file = 1;
            b->file_last = rb->temp_file->file.offset;
            b->file = &rb->temp_file->file;

            rb->bufs = cl;

        } else {
            rb->bufs = NULL;
        }
    }

    r->read_event_handler = ngx_http_block_reading;

    rb->post_handler(r);

    return NGX_OK;
}


static ngx_int_t
ngx_http_write_request_body(ngx_http_request_t *r)
```

```
{
    ssize_t                   n;
    ngx_chain_t              *cl;
    ngx_temp_file_t          *tf;
    ngx_http_request_body_t  *rb;
    ngx_http_core_loc_conf_t  *clcf;

    rb = r->request_body;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http write client request body, bufs %p", rb->bufs);

    if (rb->temp_file == NULL) {
        tf = ngx_pcalloc(r->pool, sizeof(ngx_temp_file_t));
        if (tf == NULL) {
            return NGX_ERROR;
        }

        clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

        tf->file.fd = NGX_INVALID_FILE;
        tf->file.log = r->connection->log;
        tf->path = clcf->client_body_temp_path;
        tf->pool = r->pool;
        tf->warn = "a client request body is buffered to a temporary file";
        tf->log_level = r->request_body_file_log_level;
        tf->persistent = r->request_body_in_persistent_file;
        tf->clean = r->request_body_in_clean_file;

        if (r->request_body_file_group_access) {
            tf->access = 0660;
        }

        rb->temp_file = tf;

        if (rb->bufs == NULL) {
            /* empty body with r->request_body_in_file_only */

            if (ngx_create_temp_file(&tf->file, tf->path, tf->pool,
                                     tf->persistent, tf->clean, tf->access)
                != NGX_OK)
            {
                return NGX_ERROR;
            }

            return NGX_OK;
        }
    }

    if (rb->bufs == NULL) {
        return NGX_OK;
    }

    n = ngx_write_chain_to_temp_file(rb->temp_file, rb->bufs);

    /* TODO: n == 0 or not complete and level event */

    if (n == NGX_ERROR) {
        return NGX_ERROR;
    }

    rb->temp_file->offset += n;

    /* mark all buffers as written */

    for (cl = rb->bufs; cl; cl = cl->next) {
        cl->buf->pos = cl->buf->last;
    }

    rb->bufs = NULL;

    return NGX_OK;
}


ngx_int_t
```

```
ngx_http_discard_request_body(ngx_http_request_t *r)
{
    ssize_t       size;
    ngx_int_t     rc;
    ngx_event_t  *rev;

#if (NGX_HTTP_SPDY)
    if (r->spdy_stream && r == r->main) {
        r->spdy_stream->skip_data = NGX_SPDY_DATA_DISCARD;
        return NGX_OK;
    }
#endif

    if (r != r->main || r->discard_body || r->request_body) {
        return NGX_OK;
    }

    if (ngx_http_test_expect(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    rev = r->connection->read;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, rev->log, 0, "http set discard body");

    if (rev->timer_set) {
        ngx_del_timer(rev);
    }

    if (r->headers_in.content_length_n <= 0 && !r->headers_in.chunked) {
        return NGX_OK;
    }

    size = r->header_in->last - r->header_in->pos;

    if (size || r->headers_in.chunked) {
        rc = ngx_http_discard_request_body_filter(r, r->header_in);

        if (rc != NGX_OK) {
            return rc;
        }

        if (r->headers_in.content_length_n == 0) {
            return NGX_OK;
        }
    }

    rc = ngx_http_read_discarded_request_body(r);

    if (rc == NGX_OK) {
        r->lingering_close = 0;
        return NGX_OK;
    }

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        return rc;
    }

    /* rc == NGX_AGAIN */

    r->read_event_handler = ngx_http_discarded_request_body_handler;

    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    r->count++;
    r->discard_body = 1;

    return NGX_OK;
}


void
ngx_http_discarded_request_body_handler(ngx_http_request_t *r)
{
```

```
    ngx_int_t                  rc;
    ngx_msec_t                 timer;
    ngx_event_t               *rev;
    ngx_connection_t          *c;
    ngx_http_core_loc_conf_t  *clcf;

    c = r->connection;
    rev = c->read;

    if (rev->timedout) {
        c->timedout = 1;
        c->error = 1;
        ngx_http_finalize_request(r, NGX_ERROR);
        return;
    }

    if (r->lingering_time) {
        timer = (ngx_msec_t) r->lingering_time - (ngx_msec_t) ngx_time();

        if ((ngx_msec_int_t) timer <= 0) {
            r->discard_body = 0;
            r->lingering_close = 0;
            ngx_http_finalize_request(r, NGX_ERROR);
            return;
        }

    } else {
        timer = 0;
    }

    rc = ngx_http_read_discarded_request_body(r);

    if (rc == NGX_OK) {
        r->discard_body = 0;
        r->lingering_close = 0;
        ngx_http_finalize_request(r, NGX_DONE);
        return;
    }

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        c->error = 1;
        ngx_http_finalize_request(r, NGX_ERROR);
        return;
    }

    /* rc == NGX_AGAIN */

    if (ngx_handle_read_event(rev, 0) != NGX_OK) {
        c->error = 1;
        ngx_http_finalize_request(r, NGX_ERROR);
        return;
    }

    if (timer) {

        clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

        timer *= 1000;

        if (timer > clcf->lingering_timeout) {
            timer = clcf->lingering_timeout;
        }

        ngx_add_timer(rev, timer);
    }
}


static ngx_int_t
ngx_http_read_discarded_request_body(ngx_http_request_t *r)
{
    size_t     size;
    ssize_t    n;
    ngx_int_t  rc;
    ngx_buf_t  b;
    u_char     buffer[NGX_HTTP_DISCARD_BUFFER_SIZE];
```

```
644
645     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
646                    "http read discarded body");
647
648     ngx_memzero(&b, sizeof(ngx_buf_t));
649
650     b.temporary = 1;
651
652     for ( ;; ) {
653         if (r->headers_in.content_length_n == 0) {
654             r->read_event_handler = ngx_http_block_reading;
655             return NGX_OK;
656         }
657
658         if (!r->connection->read->ready) {
659             return NGX_AGAIN;
660         }
661
662         size = (size_t) ngx_min(r->headers_in.content_length_n,
663                                 NGX_HTTP_DISCARD_BUFFER_SIZE);
664
665         n = r->connection->recv(r->connection, buffer, size);
666
667         if (n == NGX_ERROR) {
668             r->connection->error = 1;
669             return NGX_OK;
670         }
671
672         if (n == NGX_AGAIN) {
673             return NGX_AGAIN;
674         }
675
676         if (n == 0) {
677             return NGX_OK;
678         }
679
680         b.pos = buffer;
681         b.last = buffer + n;
682
683         rc = ngx_http_discard_request_body_filter(r, &b);
684
685         if (rc != NGX_OK) {
686             return rc;
687         }
688     }
689 }
690
691
692 static ngx_int_t
693 ngx_http_discard_request_body_filter(ngx_http_request_t *r, ngx_buf_t *b)
694 {
695     size_t                     size;
696     ngx_int_t                  rc;
697     ngx_http_request_body_t  *rb;
698
699     if (r->headers_in.chunked) {
700
701         rb = r->request_body;
702
703         if (rb == NULL) {
704
705             rb = ngx_pcalloc(r->pool, sizeof(ngx_http_request_body_t));
706             if (rb == NULL) {
707                 return NGX_HTTP_INTERNAL_SERVER_ERROR;
708             }
709
710             rb->chunked = ngx_pcalloc(r->pool, sizeof(ngx_http_chunked_t));
711             if (rb->chunked == NULL) {
712                 return NGX_HTTP_INTERNAL_SERVER_ERROR;
713             }
714
715             r->request_body = rb;
716         }
717
718         for ( ;; ) {
719
```

```
            rc = ngx_http_parse_chunked(r, b, rb->chunked);

            if (rc == NGX_OK) {

                /* a chunk has been parsed successfully */

                size = b->last - b->pos;

                if ((off_t) size > rb->chunked->size) {
                    b->pos += (size_t) rb->chunked->size;
                    rb->chunked->size = 0;

                } else {
                    rb->chunked->size -= size;
                    b->pos = b->last;
                }

                continue;
            }

            if (rc == NGX_DONE) {

                /* a whole response has been parsed successfully */

                r->headers_in.content_length_n = 0;
                break;
            }

            if (rc == NGX_AGAIN) {

                /* set amount of data we want to see next time */

                r->headers_in.content_length_n = rb->chunked->length;
                break;
            }

            /* invalid */

            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "client sent invalid chunked body");

            return NGX_HTTP_BAD_REQUEST;
        }

    } else {
        size = b->last - b->pos;

        if ((off_t) size > r->headers_in.content_length_n) {
            b->pos += (size_t) r->headers_in.content_length_n;
            r->headers_in.content_length_n = 0;

        } else {
            b->pos = b->last;
            r->headers_in.content_length_n -= size;
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_test_expect(ngx_http_request_t *r)
{
    ngx_int_t   n;
    ngx_str_t  *expect;

    if (r->expect_tested
        || r->headers_in.expect == NULL
        || r->http_version < NGX_HTTP_VERSION_11)
    {
        return NGX_OK;
    }

    r->expect_tested = 1;

```

```
    expect = &r->headers_in.expect->value;

    if (expect->len != sizeof("100-continue") - 1
        || ngx_strncasecmp(expect->data, (u_char *) "100-continue",
                           sizeof("100-continue") - 1)
           != 0)
    {
        return NGX_OK;
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "send 100 Continue");

    n = r->connection->send(r->connection,
                            (u_char *) "HTTP/1.1 100 Continue" CRLF CRLF,
                            sizeof("HTTP/1.1 100 Continue" CRLF CRLF) - 1);

    if (n == sizeof("HTTP/1.1 100 Continue" CRLF CRLF) - 1) {
        return NGX_OK;
    }

    /* we assume that such small packet should be send successfully */

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_request_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    if (r->headers_in.chunked) {
        return ngx_http_request_body_chunked_filter(r, in);

    } else {
        return ngx_http_request_body_length_filter(r, in);
    }
}


static ngx_int_t
ngx_http_request_body_length_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    size_t                    size;
    ngx_int_t                 rc;
    ngx_buf_t                 *b;
    ngx_chain_t               *cl, *tl, *out, **ll;
    ngx_http_request_body_t   *rb;

    rb = r->request_body;

    if (rb->rest == -1) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http request body content length filter");

        rb->rest = r->headers_in.content_length_n;
    }

    out = NULL;
    ll = &out;

    for (cl = in; cl; cl = cl->next) {

        if (rb->rest == 0) {
            break;
        }

        tl = ngx_chain_get_free_buf(r->pool, &rb->free);
        if (tl == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        b = tl->buf;

        ngx_memzero(b, sizeof(ngx_buf_t));

        b->temporary = 1;
```

```
        b->tag = (ngx_buf_tag_t) &ngx_http_read_client_request_body;
        b->start = cl->buf->pos;
        b->pos = cl->buf->pos;
        b->last = cl->buf->last;
        b->end = cl->buf->end;

        size = cl->buf->last - cl->buf->pos;

        if ((off_t) size < rb->rest) {
            cl->buf->pos = cl->buf->last;
            rb->rest -= size;

        } else {
            cl->buf->pos += (size_t) rb->rest;
            rb->rest = 0;
            b->last = cl->buf->pos;
            b->last_buf = 1;
        }

        *ll = tl;
        ll = &tl->next;
    }

    rc = ngx_http_request_body_save_filter(r, out);

    ngx_chain_update_chains(r->pool, &rb->free, &rb->busy, &out,
                            (ngx_buf_tag_t) &ngx_http_read_client_request_body);

    return rc;
}


static ngx_int_t
ngx_http_request_body_chunked_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    size_t                     size;
    ngx_int_t                  rc;
    ngx_buf_t                 *b;
    ngx_chain_t               *cl, *out, *tl, **ll;
    ngx_http_request_body_t   *rb;
    ngx_http_core_loc_conf_t  *clcf;

    rb = r->request_body;

    if (rb->rest == -1) {

        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http request body chunked filter");

        rb->chunked = ngx_pcalloc(r->pool, sizeof(ngx_http_chunked_t));
        if (rb->chunked == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        r->headers_in.content_length_n = 0;
        rb->rest = 3;
    }

    out = NULL;
    ll = &out;

    for (cl = in; cl; cl = cl->next) {

        for ( ;; ) {

            ngx_log_debug7(NGX_LOG_DEBUG_EVENT, r->connection->log, 0,
                           "http body chunked buf "
                           "t:%d f:%d %p, pos %p, size: %z file: %O, size: %z",
                           cl->buf->temporary, cl->buf->in_file,
                           cl->buf->start, cl->buf->pos,
                           cl->buf->last - cl->buf->pos,
                           cl->buf->file_pos,
                           cl->buf->file_last - cl->buf->file_pos);

            rc = ngx_http_parse_chunked(r, cl->buf, rb->chunked);

```

```
948             if (rc == NGX_OK) {
949
950                 /* a chunk has been parsed successfully */
951
952                 clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
953
954                 if (clcf->client_max_body_size
955                     && clcf->client_max_body_size
956                        - r->headers_in.content_length_n < rb->chunked->size)
957                 {
958                     ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
959                                   "client intended to send too large chunked "
960                                   "body: %O+%O bytes",
961                                   r->headers_in.content_length_n,
962                                   rb->chunked->size);
963
964                     r->lingering_close = 1;
965
966                     return NGX_HTTP_REQUEST_ENTITY_TOO_LARGE;
967                 }
968
969                 tl = ngx_chain_get_free_buf(r->pool, &rb->free);
970                 if (tl == NULL) {
971                     return NGX_HTTP_INTERNAL_SERVER_ERROR;
972                 }
973
974                 b = tl->buf;
975
976                 ngx_memzero(b, sizeof(ngx_buf_t));
977
978                 b->temporary = 1;
979                 b->tag = (ngx_buf_tag_t) &ngx_http_read_client_request_body;
980                 b->start = cl->buf->pos;
981                 b->pos = cl->buf->pos;
982                 b->last = cl->buf->last;
983                 b->end = cl->buf->end;
984
985                 *ll = tl;
986                 ll = &tl->next;
987
988                 size = cl->buf->last - cl->buf->pos;
989
990                 if ((off_t) size > rb->chunked->size) {
991                     cl->buf->pos += (size_t) rb->chunked->size;
992                     r->headers_in.content_length_n += rb->chunked->size;
993                     rb->chunked->size = 0;
994
995                 } else {
996                     rb->chunked->size -= size;
997                     r->headers_in.content_length_n += size;
998                     cl->buf->pos = cl->buf->last;
999                 }
1000
1001                 b->last = cl->buf->pos;
1002
1003                 continue;
1004             }
1005
1006             if (rc == NGX_DONE) {
1007
1008                 /* a whole response has been parsed successfully */
1009
1010                 rb->rest = 0;
1011
1012                 tl = ngx_chain_get_free_buf(r->pool, &rb->free);
1013                 if (tl == NULL) {
1014                     return NGX_HTTP_INTERNAL_SERVER_ERROR;
1015                 }
1016
1017                 b = tl->buf;
1018
1019                 ngx_memzero(b, sizeof(ngx_buf_t));
1020
1021                 b->last_buf = 1;
1022
1023                 *ll = tl;
```

```
                ll = &tl->next;

                break;
            }

            if (rc == NGX_AGAIN) {

                /* set rb->rest, amount of data we want to see next time */

                rb->rest = rb->chunked->length;

                break;
            }

            /* invalid */

            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "client sent invalid chunked body");

            return NGX_HTTP_BAD_REQUEST;
        }
    }

    rc = ngx_http_request_body_save_filter(r, out);

    ngx_chain_update_chains(r->pool, &rb->free, &rb->busy, &out,
                            (ngx_buf_tag_t) &ngx_http_read_client_request_body);

    return rc;
}


static ngx_int_t
ngx_http_request_body_save_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
#if (NGX_DEBUG)
    ngx_chain_t               *cl;
#endif
    ngx_http_request_body_t   *rb;

    rb = r->request_body;

#if (NGX_DEBUG)

    for (cl = rb->bufs; cl; cl = cl->next) {
        ngx_log_debug7(NGX_LOG_DEBUG_EVENT, r->connection->log, 0,
                       "http body old buf t:%d f:%d %p, pos %p, size: %z "
                       "file: %O, size: %z",
                       cl->buf->temporary, cl->buf->in_file,
                       cl->buf->start, cl->buf->pos,
                       cl->buf->last - cl->buf->pos,
                       cl->buf->file_pos,
                       cl->buf->file_last - cl->buf->file_pos);
    }

    for (cl = in; cl; cl = cl->next) {
        ngx_log_debug7(NGX_LOG_DEBUG_EVENT, r->connection->log, 0,
                       "http body new buf t:%d f:%d %p, pos %p, size: %z "
                       "file: %O, size: %z",
                       cl->buf->temporary, cl->buf->in_file,
                       cl->buf->start, cl->buf->pos,
                       cl->buf->last - cl->buf->pos,
                       cl->buf->file_pos,
                       cl->buf->file_last - cl->buf->file_pos);
    }

#endif

    /* TODO: coalesce neighbouring buffers */

    if (ngx_chain_add_copy(r->pool, &rb->bufs, in) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    return NGX_OK;
}
```

src/core/ngx_open_file_cache.h - nginx-1.7.10

Data types defined

- ngx_cached_open_file_s
- ngx_cached_open_file_t
- ngx_open_file_cache_cleanup_t
- ngx_open_file_cache_event_t
- ngx_open_file_cache_t
- ngx_open_file_info_t

Macros defined

- NGX_OPEN_FILE_DIRECTIO_OFF
- _NGX_OPEN_FILE_CACHE_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  #ifndef _NGX_OPEN_FILE_CACHE_H_INCLUDED_
13  #define _NGX_OPEN_FILE_CACHE_H_INCLUDED_
14
15
16  #define NGX_OPEN_FILE_DIRECTIO_OFF  NGX_MAX_OFF_T_VALUE
17
18
19  typedef struct {
20      ngx_fd_t                 fd;
21      ngx_file_uniq_t          uniq;
22      time_t                   mtime;
23      off_t                    size;
24      off_t                    fs_size;
25      off_t                    directio;
26      size_t                   read_ahead;
27
28      ngx_err_t                err;
29      char                    *failed;
30
31      time_t                   valid;
32
33      ngx_uint_t               min_uses;
34
35  #if (NGX_HAVE_OPENAT)
36      size_t                   disable_symlinks_from;
37      unsigned                 disable_symlinks:2;
38  #endif
39
40      unsigned                 test_dir:1;
41      unsigned                 test_only:1;
42      unsigned                 log:1;
43      unsigned                 errors:1;
```

```
    unsigned                 events:1;

    unsigned                 is_dir:1;
    unsigned                 is_file:1;
    unsigned                 is_link:1;
    unsigned                 is_exec:1;
    unsigned                 is_directio:1;
} ngx_open_file_info_t;


typedef struct ngx_cached_open_file_s  ngx_cached_open_file_t;

struct ngx_cached_open_file_s {
    ngx_rbtree_node_t        node;
    ngx_queue_t              queue;

    u_char                  *name;
    time_t                   created;
    time_t                   accessed;

    ngx_fd_t                 fd;
    ngx_file_uniq_t          uniq;
    time_t                   mtime;
    off_t                    size;
    ngx_err_t                err;

    uint32_t                 uses;

#if (NGX_HAVE_OPENAT)
    size_t                   disable_symlinks_from;
    unsigned                 disable_symlinks:2;
#endif

    unsigned                 count:24;
    unsigned                 close:1;
    unsigned                 use_event:1;

    unsigned                 is_dir:1;
    unsigned                 is_file:1;
    unsigned                 is_link:1;
    unsigned                 is_exec:1;
    unsigned                 is_directio:1;

    ngx_event_t             *event;
};


typedef struct {
    ngx_rbtree_t             rbtree;
    ngx_rbtree_node_t        sentinel;
    ngx_queue_t              expire_queue;

    ngx_uint_t               current;
    ngx_uint_t               max;
    time_t                   inactive;
} ngx_open_file_cache_t;


 typedef struct {
     ngx_open_file_cache_t   *cache;
     ngx_cached_open_file_t  *file;
     ngx_uint_t               min_uses;
     ngx_log_t               *log;
 } ngx_open_file_cache_cleanup_t;


 typedef struct {

     /* ngx_connection_t stub to allow use c->fd as event ident */
     void                    *data;
     ngx_event_t             *read;
     ngx_event_t             *write;
     ngx_fd_t                 fd;

     ngx_cached_open_file_t  *file;
     ngx_open_file_cache_t   *cache;
```

```
} ngx_open_file_cache_event_t;


ngx_open_file_cache_t *ngx_open_file_cache_init(ngx_pool_t *pool,
    ngx_uint_t max, time_t inactive);
ngx_int_t ngx_open_cached_file(ngx_open_file_cache_t *cache, ngx_str_t *name,
    ngx_open_file_info_t *of, ngx_pool_t *pool);


#endif /* _NGX_OPEN_FILE_CACHE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_open_file_cache.c - nginx-1.7.10

Functions defined

- ngx_close_cached_file
- ngx_expire_old_cached_files
- ngx_file_info_wrapper
- ngx_file_o_path_info
- ngx_open_and_stat_file
- ngx_open_cached_file
- ngx_open_file_add_event
- ngx_open_file_cache_cleanup
- ngx_open_file_cache_init
- ngx_open_file_cache_rbtree_insert_value
- ngx_open_file_cache_remove
- ngx_open_file_cleanup
- ngx_open_file_del_event
- ngx_open_file_lookup
- ngx_open_file_wrapper
- ngx_openat_file_owner

Macros defined

- NGX_MIN_READ_AHEAD

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  /*
14   * open file cache caches
15   *    open file handles with stat() info;
16   *    directories stat() info;
17   *    files and directories errors: not found, access denied, etc.
18   */
19
20
21  #define NGX_MIN_READ_AHEAD  (128 * 1024)
22
23
```

```c
static void ngx_open_file_cache_cleanup(void *data);
#if (NGX_HAVE_OPENAT)
static ngx_fd_t ngx_openat_file_owner(ngx_fd_t at_fd, const u_char *name,
    ngx_int_t mode, ngx_int_t create, ngx_int_t access, ngx_log_t *log);
#if (NGX_HAVE_O_PATH)
static ngx_int_t ngx_file_o_path_info(ngx_fd_t fd, ngx_file_info_t *fi,
    ngx_log_t *log);
#endif
#endif
static ngx_fd_t ngx_open_file_wrapper(ngx_str_t *name,
    ngx_open_file_info_t *of, ngx_int_t mode, ngx_int_t create,
    ngx_int_t access, ngx_log_t *log);
static ngx_int_t ngx_file_info_wrapper(ngx_str_t *name,
    ngx_open_file_info_t *of, ngx_file_info_t *fi, ngx_log_t *log);
static ngx_int_t ngx_open_and_stat_file(ngx_str_t *name,
    ngx_open_file_info_t *of, ngx_log_t *log);
static void ngx_open_file_add_event(ngx_open_file_cache_t *cache,
    ngx_cached_open_file_t *file, ngx_open_file_info_t *of, ngx_log_t *log);
static void ngx_open_file_cleanup(void *data);
static void ngx_close_cached_file(ngx_open_file_cache_t *cache,
    ngx_cached_open_file_t *file, ngx_uint_t min_uses, ngx_log_t *log);
static void ngx_open_file_del_event(ngx_cached_open_file_t *file);
static void ngx_expire_old_cached_files(ngx_open_file_cache_t *cache,
    ngx_uint_t n, ngx_log_t *log);
static void ngx_open_file_cache_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
static ngx_cached_open_file_t *
    ngx_open_file_lookup(ngx_open_file_cache_t *cache, ngx_str_t *name,
    uint32_t hash);
static void ngx_open_file_cache_remove(ngx_event_t *ev);


ngx_open_file_cache_t *
ngx_open_file_cache_init(ngx_pool_t *pool, ngx_uint_t max, time_t inactive)
{
    ngx_pool_cleanup_t     *cln;
    ngx_open_file_cache_t  *cache;

    cache = ngx_palloc(pool, sizeof(ngx_open_file_cache_t));
    if (cache == NULL) {
        return NULL;
    }

    ngx_rbtree_init(&cache->rbtree, &cache->sentinel,
                    ngx_open_file_cache_rbtree_insert_value);

    ngx_queue_init(&cache->expire_queue);

    cache->current = 0;
    cache->max = max;
    cache->inactive = inactive;

    cln = ngx_pool_cleanup_add(pool, 0);
    if (cln == NULL) {
        return NULL;
    }

    cln->handler = ngx_open_file_cache_cleanup;
    cln->data = cache;

    return cache;
}


static void
ngx_open_file_cache_cleanup(void *data)
{
    ngx_open_file_cache_t  *cache = data;

    ngx_queue_t             *q;
    ngx_cached_open_file_t  *file;

    ngx_log_debug0(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
                   "open file cache cleanup");

    for ( ;; ) {
```

```
100
101         if (ngx_queue_empty(&cache->expire_queue)) {
102             break;
103         }
104
105         q = ngx_queue_last(&cache->expire_queue);
106
107         file = ngx_queue_data(q, ngx_cached_open_file_t, queue);
108
109         ngx_queue_remove(q);
110
111         ngx_rbtree_delete(&cache->rbtree, &file->node);
112
113         cache->current--;
114
115         ngx_log_debug1(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
116                        "delete cached open file: %s", file->name);
117
118         if (!file->err && !file->is_dir) {
119             file->close = 1;
120             file->count = 0;
121             ngx_close_cached_file(cache, file, 0, ngx_cycle->log);
122
123         } else {
124             ngx_free(file->name);
125             ngx_free(file);
126         }
127     }
128
129     if (cache->current) {
130         ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
131                       "%ui items still leave in open file cache",
132                       cache->current);
133     }
134
135     if (cache->rbtree.root != cache->rbtree.sentinel) {
136         ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
137                       "rbtree still is not empty in open file cache");
138
139     }
140 }
141
142
143 ngx_int_t
144 ngx_open_cached_file(ngx_open_file_cache_t *cache, ngx_str_t *name,
145     ngx_open_file_info_t *of, ngx_pool_t *pool)
146 {
147     time_t                        now;
148     uint32_t                      hash;
149     ngx_int_t                     rc;
150     ngx_file_info_t               fi;
151     ngx_pool_cleanup_t           *cln;
152     ngx_cached_open_file_t       *file;
153     ngx_pool_cleanup_file_t      *clnf;
154     ngx_open_file_cache_cleanup_t  *ofcln;
155
156     of->fd = NGX_INVALID_FILE;
157     of->err = 0;
158
159     if (cache == NULL) {
160
161         if (of->test_only) {
162
163             if (ngx_file_info_wrapper(name, of, &fi, pool->log)
164                 == NGX_FILE_ERROR)
165             {
166                 return NGX_ERROR;
167             }
168
169             of->uniq = ngx_file_uniq(&fi);
170             of->mtime = ngx_file_mtime(&fi);
171             of->size = ngx_file_size(&fi);
172             of->fs_size = ngx_file_fs_size(&fi);
173             of->is_dir = ngx_is_dir(&fi);
174             of->is_file = ngx_is_file(&fi);
175             of->is_link = ngx_is_link(&fi);
```

```
            of->is_exec = ngx_is_exec(&fi);

            return NGX_OK;
        }

        cln = ngx_pool_cleanup_add(pool, sizeof(ngx_pool_cleanup_file_t));
        if (cln == NULL) {
            return NGX_ERROR;
        }

        rc = ngx_open_and_stat_file(name, of, pool->log);

        if (rc == NGX_OK && !of->is_dir) {
            cln->handler = ngx_pool_cleanup_file;
            clnf = cln->data;

            clnf->fd = of->fd;
            clnf->name = name->data;
            clnf->log = pool->log;
        }

        return rc;
    }

    cln = ngx_pool_cleanup_add(pool, sizeof(ngx_open_file_cache_cleanup_t));
    if (cln == NULL) {
        return NGX_ERROR;
    }

    now = ngx_time();

    hash = ngx_crc32_long(name->data, name->len);

    file = ngx_open_file_lookup(cache, name, hash);

    if (file) {

        file->uses++;

        ngx_queue_remove(&file->queue);

        if (file->fd == NGX_INVALID_FILE && file->err == 0 && !file->is_dir) {

            /* file was not used often enough to keep open */

            rc = ngx_open_and_stat_file(name, of, pool->log);

            if (rc != NGX_OK && (of->err == 0 || !of->errors)) {
                goto failed;
            }

            goto add_event;
        }

        if (file->use_event
            || (file->event == NULL
                && (of->uniq == 0 || of->uniq == file->uniq)
                && now - file->created < of->valid
#if (NGX_HAVE_OPENAT)
                && of->disable_symlinks == file->disable_symlinks
                && of->disable_symlinks_from == file->disable_symlinks_from
#endif
            ))
        {
            if (file->err == 0) {

                of->fd = file->fd;
                of->uniq = file->uniq;
                of->mtime = file->mtime;
                of->size = file->size;

                of->is_dir = file->is_dir;
                of->is_file = file->is_file;
                of->is_link = file->is_link;
                of->is_exec = file->is_exec;
                of->is_directio = file->is_directio;
```

```
252
253                 if (!file->is_dir) {
254                     file->count++;
255                     ngx_open_file_add_event(cache, file, of, pool->log);
256                 }
257
258             } else {
259                 of->err = file->err;
260 #if (NGX_HAVE_OPENAT)
261                 of->failed = file->disable_symlinks ? ngx_openat_file_n
262                                                     : ngx_open_file_n;
263 #else
264                 of->failed = ngx_open_file_n;
265 #endif
266             }
267
268             goto found;
269         }
270
271         ngx_log_debug4(NGX_LOG_DEBUG_CORE, pool->log, 0,
272                        "retest open file: %s, fd:%d, c:%d, e:%d",
273                        file->name, file->fd, file->count, file->err);
274
275         if (file->is_dir) {
276
277             /*
278              * chances that directory became file are very small
279              * so test_dir flag allows to use a single syscall
280              * in ngx_file_info() instead of three syscalls
281              */
282
283             of->test_dir = 1;
284         }
285
286         of->fd = file->fd;
287         of->uniq = file->uniq;
288
289         rc = ngx_open_and_stat_file(name, of, pool->log);
290
291         if (rc != NGX_OK && (of->err == 0 || !of->errors)) {
292             goto failed;
293         }
294
295         if (of->is_dir) {
296
297             if (file->is_dir || file->err) {
298                 goto update;
299             }
300
301             /* file became directory */
302
303         } else if (of->err == 0) {  /* file */
304
305             if (file->is_dir || file->err) {
306                 goto add_event;
307             }
308
309             if (of->uniq == file->uniq) {
310
311                 if (file->event) {
312                     file->use_event = 1;
313                 }
314
315                 of->is_directio = file->is_directio;
316
317                 goto update;
318             }
319
320             /* file was changed */
321
322         } else { /* error to cache */
323
324             if (file->err || file->is_dir) {
325                 goto update;
326             }
327
```

```
            /* file was removed, etc. */
        }

        if (file->count == 0) {

            ngx_open_file_del_event(file);

            if (ngx_close_file(file->fd) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_ALERT, pool->log, ngx_errno,
                              ngx_close_file_n " \"%V\" failed", name);
            }

            goto add_event;
        }

        ngx_rbtree_delete(&cache->rbtree, &file->node);

        cache->current--;

        file->close = 1;

        goto create;
    }

    /* not found */

    rc = ngx_open_and_stat_file(name, of, pool->log);

    if (rc != NGX_OK && (of->err == 0 || !of->errors)) {
        goto failed;
    }

create:

    if (cache->current >= cache->max) {
        ngx_expire_old_cached_files(cache, 0, pool->log);
    }

    file = ngx_alloc(sizeof(ngx_cached_open_file_t), pool->log);

    if (file == NULL) {
        goto failed;
    }

    file->name = ngx_alloc(name->len + 1, pool->log);

    if (file->name == NULL) {
        ngx_free(file);
        file = NULL;
        goto failed;
    }

    ngx_cpystrn(file->name, name->data, name->len + 1);

    file->node.key = hash;

    ngx_rbtree_insert(&cache->rbtree, &file->node);

    cache->current++;

    file->uses = 1;
    file->count = 0;
    file->use_event = 0;
    file->event = NULL;

add_event:

    ngx_open_file_add_event(cache, file, of, pool->log);

update:

    file->fd = of->fd;
    file->err = of->err;
#if (NGX_HAVE_OPENAT)
    file->disable_symlinks = of->disable_symlinks;
    file->disable_symlinks_from = of->disable_symlinks_from;
```

```
#endif

    if (of->err == 0) {
        file->uniq = of->uniq;
        file->mtime = of->mtime;
        file->size = of->size;

        file->close = 0;

        file->is_dir = of->is_dir;
        file->is_file = of->is_file;
        file->is_link = of->is_link;
        file->is_exec = of->is_exec;
        file->is_directio = of->is_directio;

        if (!of->is_dir) {
            file->count++;
        }
    }

    file->created = now;

found:

    file->accessed = now;

    ngx_queue_insert_head(&cache->expire_queue, &file->queue);

    ngx_log_debug5(NGX_LOG_DEBUG_CORE, pool->log, 0,
                   "cached open file: %s, fd:%d, c:%d, e:%d, u:%d",
                   file->name, file->fd, file->count, file->err, file->uses);

    if (of->err == 0) {

        if (!of->is_dir) {
            cln->handler = ngx_open_file_cleanup;
            ofcln = cln->data;

            ofcln->cache = cache;
            ofcln->file = file;
            ofcln->min_uses = of->min_uses;
            ofcln->log = pool->log;
        }

        return NGX_OK;
    }

    return NGX_ERROR;

failed:

    if (file) {
        ngx_rbtree_delete(&cache->rbtree, &file->node);

        cache->current--;

        if (file->count == 0) {

            if (file->fd != NGX_INVALID_FILE) {
                if (ngx_close_file(file->fd) == NGX_FILE_ERROR) {
                    ngx_log_error(NGX_LOG_ALERT, pool->log, ngx_errno,
                                  ngx_close_file_n " \"%s\" failed",
                                  file->name);
                }
            }

            ngx_free(file->name);
            ngx_free(file);

        } else {
            file->close = 1;
        }
    }

    if (of->fd != NGX_INVALID_FILE) {
        if (ngx_close_file(of->fd) == NGX_FILE_ERROR) {
```

```
            ngx_log_error(NGX_LOG_ALERT, pool->log, ngx_errno,
                          ngx_close_file_n " \"%V\" failed", name);
        }
    }

    return NGX_ERROR;
}


#if (NGX_HAVE_OPENAT)

static ngx_fd_t
ngx_openat_file_owner(ngx_fd_t at_fd, const u_char *name,
    ngx_int_t mode, ngx_int_t create, ngx_int_t access, ngx_log_t *log)
{
    ngx_fd_t         fd;
    ngx_err_t        err;
    ngx_file_info_t  fi, atfi;

    /*
     * To allow symlinks with the same owner, use openat() (followed
     * by fstat()) and fstatat(AT_SYMLINK_NOFOLLOW), and then compare
     * uids between fstat() and fstatat().
     *
     * As there is a race between openat() and fstatat() we don't
     * know if openat() in fact opened symlink or not.  Therefore,
     * we have to compare uids even if fstatat() reports the opened
     * component isn't a symlink (as we don't know whether it was
     * symlink during openat() or not).
     */

    fd = ngx_openat_file(at_fd, name, mode, create, access);

    if (fd == NGX_INVALID_FILE) {
        return NGX_INVALID_FILE;
    }

    if (ngx_file_at_info(at_fd, name, &atfi, AT_SYMLINK_NOFOLLOW)
        == NGX_FILE_ERROR)
    {
        err = ngx_errno;
        goto failed;
    }

#if (NGX_HAVE_O_PATH)
    if (ngx_file_o_path_info(fd, &fi, log) == NGX_ERROR) {
        err = ngx_errno;
        goto failed;
    }
#else
    if (ngx_fd_info(fd, &fi) == NGX_FILE_ERROR) {
        err = ngx_errno;
        goto failed;
    }
#endif

    if (fi.st_uid != atfi.st_uid) {
        err = NGX_ELOOP;
        goto failed;
    }

    return fd;

failed:

    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      ngx_close_file_n " \"%V\" failed", name);
    }

    ngx_set_errno(err);

    return NGX_INVALID_FILE;
}
```

```c
#if (NGX_HAVE_O_PATH)

static ngx_int_t
ngx_file_o_path_info(ngx_fd_t fd, ngx_file_info_t *fi, ngx_log_t *log)
{
    static ngx_uint_t  use_fstat = 1;

    /*
     * In Linux 2.6.39 the O_PATH flag was introduced that allows to obtain
     * a descriptor without actually opening file or directory.  It requires
     * less permissions for path components, but till Linux 3.6 fstat() returns
     * EBADF on such descriptors, and fstatat() with the AT_EMPTY_PATH flag
     * should be used instead.
     *
     * Three scenarios are handled in this function:
     *
     * 1) The kernel is newer than 3.6 or fstat() with O_PATH support was
     *    backported by vendor.  Then fstat() is used.
     *
     * 2) The kernel is newer than 2.6.39 but older than 3.6.  In this case
     *    the first call of fstat() returns EBADF and we fallback to fstatat()
     *    with AT_EMPTY_PATH which was introduced at the same time as O_PATH.
     *
     * 3) The kernel is older than 2.6.39 but nginx was build with O_PATH
     *    support.  Since descriptors are opened with O_PATH|O_RDONLY flags
     *    and O_PATH is ignored by the kernel then the O_RDONLY flag is
     *    actually used.  In this case fstat() just works.
     */

    if (use_fstat) {
        if (ngx_fd_info(fd, fi) != NGX_FILE_ERROR) {
            return NGX_OK;
        }

        if (ngx_errno != NGX_EBADF) {
            return NGX_ERROR;
        }

        ngx_log_error(NGX_LOG_NOTICE, log, 0,
                      "fstat(O_PATH) failed with EBADF, "
                      "switching to fstatat(AT_EMPTY_PATH)");

        use_fstat = 0;
    }

    if (ngx_file_at_info(fd, "", fi, AT_EMPTY_PATH) != NGX_FILE_ERROR) {
        return NGX_OK;
    }

    return NGX_ERROR;
}

#endif

#endif /* NGX_HAVE_OPENAT */


static ngx_fd_t
ngx_open_file_wrapper(ngx_str_t *name, ngx_open_file_info_t *of,
    ngx_int_t mode, ngx_int_t create, ngx_int_t access, ngx_log_t *log)
{
    ngx_fd_t  fd;

#if !(NGX_HAVE_OPENAT)

    fd = ngx_open_file(name->data, mode, create, access);

    if (fd == NGX_INVALID_FILE) {
        of->err = ngx_errno;
        of->failed = ngx_open_file_n;
        return NGX_INVALID_FILE;
    }

    return fd;

#else
```

```
632
633     u_char           *p, *cp, *end;
634     ngx_fd_t          at_fd;
635     ngx_str_t         at_name;
636
637     if (of->disable_symlinks == NGX_DISABLE_SYMLINKS_OFF) {
638         fd = ngx_open_file(name->data, mode, create, access);
639
640         if (fd == NGX_INVALID_FILE) {
641             of->err = ngx_errno;
642             of->failed = ngx_open_file_n;
643             return NGX_INVALID_FILE;
644         }
645
646         return fd;
647     }
648
649     p = name->data;
650     end = p + name->len;
651
652     at_name = *name;
653
654     if (of->disable_symlinks_from) {
655
656         cp = p + of->disable_symlinks_from;
657
658         *cp = '\0';
659
660         at_fd = ngx_open_file(p, NGX_FILE_SEARCH|NGX_FILE_NONBLOCK,
661                               NGX_FILE_OPEN, 0);
662
663         *cp = '/';
664
665         if (at_fd == NGX_INVALID_FILE) {
666             of->err = ngx_errno;
667             of->failed = ngx_open_file_n;
668             return NGX_INVALID_FILE;
669         }
670
671         at_name.len = of->disable_symlinks_from;
672         p = cp + 1;
673
674     } else if (*p == '/') {
675
676         at_fd = ngx_open_file("/",
677                               NGX_FILE_SEARCH|NGX_FILE_NONBLOCK,
678                               NGX_FILE_OPEN, 0);
679
680         if (at_fd == NGX_INVALID_FILE) {
681             of->err = ngx_errno;
682             of->failed = ngx_openat_file_n;
683             return NGX_INVALID_FILE;
684         }
685
686         at_name.len = 1;
687         p++;
688
689     } else {
690         at_fd = NGX_AT_FDCWD;
691     }
692
693     for ( ;; ) {
694         cp = ngx_strlchr(p, end, '/');
695         if (cp == NULL) {
696             break;
697         }
698
699         if (cp == p) {
700             p++;
701             continue;
702         }
703
704         *cp = '\0';
705
706         if (of->disable_symlinks == NGX_DISABLE_SYMLINKS_NOTOWNER) {
707             fd = ngx_openat_file_owner(at_fd, p,
```

```
                                      NGX_FILE_SEARCH|NGX_FILE_NONBLOCK,
                                      NGX_FILE_OPEN, 0, log);

        } else {
            fd = ngx_openat_file(at_fd, p,
                         NGX_FILE_SEARCH|NGX_FILE_NONBLOCK|NGX_FILE_NOFOLLOW,
                         NGX_FILE_OPEN, 0);
        }

        *cp = '/';

        if (fd == NGX_INVALID_FILE) {
            of->err = ngx_errno;
            of->failed = ngx_openat_file_n;
            goto failed;
        }

        if (at_fd != NGX_AT_FDCWD && ngx_close_file(at_fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                          ngx_close_file_n " \"%V\" failed", &at_name);
        }

        p = cp + 1;
        at_fd = fd;
        at_name.len = cp - at_name.data;
    }

    if (p == end) {

        /*
         * If pathname ends with a trailing slash, assume the last path
         * component is a directory and reopen it with requested flags;
         * if not, fail with ENOTDIR as per POSIX.
         *
         * We cannot rely on O_DIRECTORY in the loop above to check
         * that the last path component is a directory because
         * O_DIRECTORY doesn't work on FreeBSD 8.  Fortunately, by
         * reopening a directory, we don't depend on it at all.
         */

        fd = ngx_openat_file(at_fd, ".", mode, create, access);
        goto done;
    }

    if (of->disable_symlinks == NGX_DISABLE_SYMLINKS_NOTOWNER
        && !(create & (NGX_FILE_CREATE_OR_OPEN|NGX_FILE_TRUNCATE)))
    {
        fd = ngx_openat_file_owner(at_fd, p, mode, create, access, log);

    } else {
        fd = ngx_openat_file(at_fd, p, mode|NGX_FILE_NOFOLLOW, create, access);
    }

done:

    if (fd == NGX_INVALID_FILE) {
        of->err = ngx_errno;
        of->failed = ngx_openat_file_n;
    }

failed:

    if (at_fd != NGX_AT_FDCWD && ngx_close_file(at_fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      ngx_close_file_n " \"%V\" failed", &at_name);
    }

    return fd;
#endif
}


static ngx_int_t
ngx_file_info_wrapper(ngx_str_t *name, ngx_open_file_info_t *of,
    ngx_file_info_t *fi, ngx_log_t *log)
{
```

```
    ngx_int_t  rc;

#if !(NGX_HAVE_OPENAT)

    rc = ngx_file_info(name->data, fi);

    if (rc == NGX_FILE_ERROR) {
        of->err = ngx_errno;
        of->failed = ngx_file_info_n;
        return NGX_FILE_ERROR;
    }

    return rc;

#else

    ngx_fd_t  fd;

    if (of->disable_symlinks == NGX_DISABLE_SYMLINKS_OFF) {

        rc = ngx_file_info(name->data, fi);

        if (rc == NGX_FILE_ERROR) {
            of->err = ngx_errno;
            of->failed = ngx_file_info_n;
            return NGX_FILE_ERROR;
        }

        return rc;
    }

    fd = ngx_open_file_wrapper(name, of, NGX_FILE_RDONLY|NGX_FILE_NONBLOCK,
                               NGX_FILE_OPEN, 0, log);

    if (fd == NGX_INVALID_FILE) {
        return NGX_FILE_ERROR;
    }

    rc = ngx_fd_info(fd, fi);

    if (rc == NGX_FILE_ERROR) {
        of->err = ngx_errno;
        of->failed = ngx_fd_info_n;
    }

    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      ngx_close_file_n " \"%V\" failed", name);
    }

    return rc;
#endif
}


static ngx_int_t
ngx_open_and_stat_file(ngx_str_t *name, ngx_open_file_info_t *of,
    ngx_log_t *log)
{
    ngx_fd_t         fd;
    ngx_file_info_t  fi;

    if (of->fd != NGX_INVALID_FILE) {

        if (ngx_file_info_wrapper(name, of, &fi, log) == NGX_FILE_ERROR) {
            of->fd = NGX_INVALID_FILE;
            return NGX_ERROR;
        }

        if (of->uniq == ngx_file_uniq(&fi)) {
            goto done;
        }

    } else if (of->test_dir) {

        if (ngx_file_info_wrapper(name, of, &fi, log) == NGX_FILE_ERROR) {
```

```
            of->fd = NGX_INVALID_FILE;
            return NGX_ERROR;
        }

        if (ngx_is_dir(&fi)) {
            goto done;
        }
    }

    if (!of->log) {

        /*
         * Use non-blocking open() not to hang on FIFO files, etc.
         * This flag has no effect on a regular files.
         */

        fd = ngx_open_file_wrapper(name, of, NGX_FILE_RDONLY|NGX_FILE_NONBLOCK,
                                   NGX_FILE_OPEN, 0, log);

    } else {
        fd = ngx_open_file_wrapper(name, of, NGX_FILE_APPEND,
                                   NGX_FILE_CREATE_OR_OPEN,
                                   NGX_FILE_DEFAULT_ACCESS, log);
    }

    if (fd == NGX_INVALID_FILE) {
        of->fd = NGX_INVALID_FILE;
        return NGX_ERROR;
    }

    if (ngx_fd_info(fd, &fi) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_CRIT, log, ngx_errno,
                      ngx_fd_info_n " \"%V\" failed", name);

        if (ngx_close_file(fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                          ngx_close_file_n " \"%V\" failed", name);
        }

        of->fd = NGX_INVALID_FILE;

        return NGX_ERROR;
    }

    if (ngx_is_dir(&fi)) {
        if (ngx_close_file(fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                          ngx_close_file_n " \"%V\" failed", name);
        }

        of->fd = NGX_INVALID_FILE;

    } else {
        of->fd = fd;

        if (of->read_ahead && ngx_file_size(&fi) > NGX_MIN_READ_AHEAD) {
            if (ngx_read_ahead(fd, of->read_ahead) == NGX_ERROR) {
                ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                              ngx_read_ahead_n " \"%V\" failed", name);
            }
        }

        if (of->directio <= ngx_file_size(&fi)) {
            if (ngx_directio_on(fd) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                              ngx_directio_on_n " \"%V\" failed", name);

            } else {
                of->is_directio = 1;
            }
        }
    }

done:

    of->uniq = ngx_file_uniq(&fi);
```

```
936     of->mtime = ngx_file_mtime(&fi);
937     of->size = ngx_file_size(&fi);
938     of->fs_size = ngx_file_fs_size(&fi);
939     of->is_dir = ngx_is_dir(&fi);
940     of->is_file = ngx_is_file(&fi);
941     of->is_link = ngx_is_link(&fi);
942     of->is_exec = ngx_is_exec(&fi);
943
944     return NGX_OK;
945 }
946
947
948 /*
949 * we ignore any possible event setting error and
950 * fallback to usual periodic file retests
951 */
952
953 static void
954 ngx_open_file_add_event(ngx_open_file_cache_t *cache,
955     ngx_cached_open_file_t *file, ngx_open_file_info_t *of, ngx_log_t *log)
956 {
957     ngx_open_file_cache_event_t  *fev;
958
959     if (!(ngx_event_flags & NGX_USE_VNODE_EVENT)
960         || !of->events
961         || file->event
962         || of->fd == NGX_INVALID_FILE
963         || file->uses < of->min_uses)
964     {
965         return;
966     }
967
968     file->use_event = 0;
969
970     file->event = ngx_calloc(sizeof(ngx_event_t), log);
971     if (file->event== NULL) {
972         return;
973     }
974
975     fev = ngx_alloc(sizeof(ngx_open_file_cache_event_t), log);
976     if (fev == NULL) {
977         ngx_free(file->event);
978         file->event = NULL;
979         return;
980     }
981
982     fev->fd = of->fd;
983     fev->file = file;
984     fev->cache = cache;
985
986     file->event->handler = ngx_open_file_cache_remove;
987     file->event->data = fev;
988
989     /*
990      * although vnode event may be called while ngx_cycle->poll
991      * destruction, however, cleanup procedures are run before any
992      * memory freeing and events will be canceled.
993      */
994
995     file->event->log = ngx_cycle->log;
996
997     if (ngx_add_event(file->event, NGX_VNODE_EVENT, NGX_ONESHOT_EVENT)
998         != NGX_OK)
999     {
1000         ngx_free(file->event->data);
1001         ngx_free(file->event);
1002         file->event = NULL;
1003         return;
1004     }
1005
1006     /*
1007      * we do not set file->use_event here because there may be a race
1008      * condition: a file may be deleted between opening the file and
1009      * adding event, so we rely upon event notification only after
1010      * one file revalidation on next file access
1011      */
```

```
1012
1013        return;
1014    }
1015
1016
1017    static void
1018    ngx_open_file_cleanup(void *data)
1019    {
1020        ngx_open_file_cache_cleanup_t  *c = data;
1021
1022        c->file->count--;
1023
1024        ngx_close_cached_file(c->cache, c->file, c->min_uses, c->log);
1025
1026        /* drop one or two expired open files */
1027        ngx_expire_old_cached_files(c->cache, 1, c->log);
1028    }
1029
1030
1031    static void
1032    ngx_close_cached_file(ngx_open_file_cache_t *cache,
1033        ngx_cached_open_file_t *file, ngx_uint_t min_uses, ngx_log_t *log)
1034    {
1035        ngx_log_debug5(NGX_LOG_DEBUG_CORE, log, 0,
1036                       "close cached open file: %s, fd:%d, c:%d, u:%d, %d",
1037                       file->name, file->fd, file->count, file->uses, file->close);
1038
1039        if (!file->close) {
1040
1041            file->accessed = ngx_time();
1042
1043            ngx_queue_remove(&file->queue);
1044
1045            ngx_queue_insert_head(&cache->expire_queue, &file->queue);
1046
1047            if (file->uses >= min_uses || file->count) {
1048                return;
1049            }
1050        }
1051
1052        ngx_open_file_del_event(file);
1053
1054        if (file->count) {
1055            return;
1056        }
1057
1058        if (file->fd != NGX_INVALID_FILE) {
1059
1060            if (ngx_close_file(file->fd) == NGX_FILE_ERROR) {
1061                ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
1062                              ngx_close_file_n " \"%s\" failed", file->name);
1063            }
1064
1065            file->fd = NGX_INVALID_FILE;
1066        }
1067
1068        if (!file->close) {
1069            return;
1070        }
1071
1072        ngx_free(file->name);
1073        ngx_free(file);
1074    }
1075
1076
1077    static void
1078    ngx_open_file_del_event(ngx_cached_open_file_t *file)
1079    {
1080        if (file->event == NULL) {
1081            return;
1082        }
1083
1084        (void) ngx_del_event(file->event, NGX_VNODE_EVENT,
1085                             file->count ? NGX_FLUSH_EVENT : NGX_CLOSE_EVENT);
1086
1087        ngx_free(file->event->data);
```

```
    ngx_free(file->event);
    file->event = NULL;
    file->use_event = 0;
}


static void
ngx_expire_old_cached_files(ngx_open_file_cache_t *cache, ngx_uint_t n,
    ngx_log_t *log)
{
    time_t                   now;
    ngx_queue_t             *q;
    ngx_cached_open_file_t  *file;

    now = ngx_time();

    /*
     * n == 1 deletes one or two inactive files
     * n == 0 deletes least recently used file by force
     *        and one or two inactive files
     */

    while (n < 3) {

        if (ngx_queue_empty(&cache->expire_queue)) {
            return;
        }

        q = ngx_queue_last(&cache->expire_queue);

        file = ngx_queue_data(q, ngx_cached_open_file_t, queue);

        if (n++ != 0 && now - file->accessed <= cache->inactive) {
            return;
        }

        ngx_queue_remove(q);

        ngx_rbtree_delete(&cache->rbtree, &file->node);

        cache->current--;

        ngx_log_debug1(NGX_LOG_DEBUG_CORE, log, 0,
                       "expire cached open file: %s", file->name);

        if (!file->err && !file->is_dir) {
            file->close = 1;
            ngx_close_cached_file(cache, file, 0, log);

        } else {
            ngx_free(file->name);
            ngx_free(file);
        }
    }
}


static void
ngx_open_file_cache_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
    ngx_rbtree_node_t       **p;
    ngx_cached_open_file_t    *file, *file_temp;

    for ( ;; ) {

        if (node->key < temp->key) {

            p = &temp->left;

        } else if (node->key > temp->key) {

            p = &temp->right;

        } else { /* node->key == temp->key */
```

```
            file = (ngx_cached_open_file_t *) node;
            file_temp = (ngx_cached_open_file_t *) temp;

            p = (ngx_strcmp(file->name, file_temp->name) < 0)
                    ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
    ngx_rbt_red(node);
}


static ngx_cached_open_file_t *
ngx_open_file_lookup(ngx_open_file_cache_t *cache, ngx_str_t *name,
    uint32_t hash)
{
    ngx_int_t                rc;
    ngx_rbtree_node_t       *node, *sentinel;
    ngx_cached_open_file_t  *file;

    node = cache->rbtree.root;
    sentinel = cache->rbtree.sentinel;

    while (node != sentinel) {

        if (hash < node->key) {
            node = node->left;
            continue;
        }

        if (hash > node->key) {
            node = node->right;
            continue;
        }

        /* hash == node->key */

        file = (ngx_cached_open_file_t *) node;

        rc = ngx_strcmp(name->data, file->name);

        if (rc == 0) {
            return file;
        }

        node = (rc < 0) ? node->left : node->right;
    }

    return NULL;
}


static void
ngx_open_file_cache_remove(ngx_event_t *ev)
{
    ngx_cached_open_file_t       *file;
    ngx_open_file_cache_event_t  *fev;

    fev = ev->data;
    file = fev->file;

    ngx_queue_remove(&file->queue);

    ngx_rbtree_delete(&fev->cache->rbtree, &file->node);

    fev->cache->current--;
```

```
1240
1241        /* NGX_ONESHOT_EVENT was already deleted */
1242        file->event = NULL;
1243        file->use_event = 0;
1244
1245        file->close = 1;
1246
1247        ngx_close_cached_file(fev->cache, file, 0, ev->log);
1248
1249        /* free memory only when fev->cache and fev->file are already not needed */
1250
1251        ngx_free(ev->data);
1252        ngx_free(ev);
1253    }
```

One Level Up Top Level

src/http/ngx_http_script.h - nginx-1.7.10

Data types defined

- ngx_http_compile_complex_value_t
- ngx_http_complex_value_t
- ngx_http_script_code_pt
- ngx_http_script_compile_t
- ngx_http_script_complex_value_code_t
- ngx_http_script_copy_capture_code_t
- ngx_http_script_copy_code_t
- ngx_http_script_engine_t
- ngx_http_script_file_code_t
- ngx_http_script_file_op_e
- ngx_http_script_full_name_code_t
- ngx_http_script_if_code_t
- ngx_http_script_len_code_pt
- ngx_http_script_regex_code_t
- ngx_http_script_regex_end_code_t
- ngx_http_script_return_code_t
- ngx_http_script_value_code_t
- ngx_http_script_var_code_t
- ngx_http_script_var_handler_code_t

Macros defined

- _NGX_HTTP_SCRIPT_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_HTTP_SCRIPT_H_INCLUDED_
9  #define _NGX_HTTP_SCRIPT_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
```

```
typedef struct {
    u_char                     *ip;
    u_char                     *pos;
    ngx_http_variable_value_t  *sp;

    ngx_str_t                   buf;
    ngx_str_t                   line;

    /* the start of the rewritten arguments */
    u_char                     *args;

    unsigned                    flushed:1;
    unsigned                    skip:1;
    unsigned                    quote:1;
    unsigned                    is_args:1;
    unsigned                    log:1;

    ngx_int_t                   status;
    ngx_http_request_t         *request;
} ngx_http_script_engine_t;


typedef struct {
    ngx_conf_t                 *cf;
    ngx_str_t                  *source;

    ngx_array_t               **flushes;
    ngx_array_t               **lengths;
    ngx_array_t               **values;

    ngx_uint_t                  variables;
    ngx_uint_t                  ncaptures;
    ngx_uint_t                  captures_mask;
    ngx_uint_t                  size;

    void                       *main;

    unsigned                    compile_args:1;
    unsigned                    complete_lengths:1;
    unsigned                    complete_values:1;
    unsigned                    zero:1;
    unsigned                    conf_prefix:1;
    unsigned                    root_prefix:1;

    unsigned                    dup_capture:1;
    unsigned                    args:1;
} ngx_http_script_compile_t;


typedef struct {
    ngx_str_t                   value;
    ngx_uint_t                 *flushes;
    void                       *lengths;
    void                       *values;
} ngx_http_complex_value_t;


typedef struct {
    ngx_conf_t                 *cf;
    ngx_str_t                  *value;
    ngx_http_complex_value_t   *complex_value;

    unsigned                    zero:1;
    unsigned                    conf_prefix:1;
    unsigned                    root_prefix:1;
} ngx_http_compile_complex_value_t;


typedef void (*ngx_http_script_code_pt) (ngx_http_script_engine_t *e);
typedef size_t (*ngx_http_script_len_code_pt) (ngx_http_script_engine_t *e);


typedef struct {
    ngx_http_script_code_pt     code;
    uintptr_t                   len;
} ngx_http_script_copy_code_t;
```

```
93
94
95  typedef struct {
96      ngx_http_script_code_pt     code;
97      uintptr_t                   index;
98  } ngx_http_script_var_code_t;
99
100
101  typedef struct {
102      ngx_http_script_code_pt     code;
103      ngx_http_set_variable_pt    handler;
104      uintptr_t                   data;
105  } ngx_http_script_var_handler_code_t;
106
107
108  typedef struct {
109      ngx_http_script_code_pt     code;
110      uintptr_t                   n;
111  } ngx_http_script_copy_capture_code_t;
112
113
114  #if (NGX_PCRE)
115
116  typedef struct {
117      ngx_http_script_code_pt     code;
118      ngx_http_regex_t           *regex;
119      ngx_array_t                *lengths;
120      uintptr_t                   size;
121      uintptr_t                   status;
122      uintptr_t                   next;
123
124      uintptr_t                   test:1;
125      uintptr_t                   negative_test:1;
126      uintptr_t                   uri:1;
127      uintptr_t                   args:1;
128
129      /* add the r->args to the new arguments */
130      uintptr_t                   add_args:1;
131
132      uintptr_t                   redirect:1;
133      uintptr_t                   break_cycle:1;
134
135      ngx_str_t                   name;
136  } ngx_http_script_regex_code_t;
137
138
139  typedef struct {
140      ngx_http_script_code_pt     code;
141
142      uintptr_t                   uri:1;
143      uintptr_t                   args:1;
144
145      /* add the r->args to the new arguments */
146      uintptr_t                   add_args:1;
147
148      uintptr_t                   redirect:1;
149  } ngx_http_script_regex_end_code_t;
150
151  #endif
152
153
154  typedef struct {
155      ngx_http_script_code_pt     code;
156      uintptr_t                   conf_prefix;
157  } ngx_http_script_full_name_code_t;
158
159
160  typedef struct {
161      ngx_http_script_code_pt     code;
162      uintptr_t                   status;
163      ngx_http_complex_value_t    text;
164  } ngx_http_script_return_code_t;
165
166
167  typedef enum {
168      ngx_http_script_file_plain = 0,
```

```
    ngx_http_script_file_not_plain,
    ngx_http_script_file_dir,
    ngx_http_script_file_not_dir,
    ngx_http_script_file_exists,
    ngx_http_script_file_not_exists,
    ngx_http_script_file_exec,
    ngx_http_script_file_not_exec
} ngx_http_script_file_op_e;


typedef struct {
    ngx_http_script_code_pt     code;
    uintptr_t                   op;
} ngx_http_script_file_code_t;


typedef struct {
    ngx_http_script_code_pt     code;
    uintptr_t                   next;
    void                      **loc_conf;
} ngx_http_script_if_code_t;


typedef struct {
    ngx_http_script_code_pt     code;
    ngx_array_t                *lengths;
} ngx_http_script_complex_value_code_t;


typedef struct {
    ngx_http_script_code_pt     code;
    uintptr_t                   value;
    uintptr_t                   text_len;
    uintptr_t                   text_data;
} ngx_http_script_value_code_t;


void ngx_http_script_flush_complex_value(ngx_http_request_t *r,
    ngx_http_complex_value_t *val);
ngx_int_t ngx_http_complex_value(ngx_http_request_t *r,
    ngx_http_complex_value_t *val, ngx_str_t *value);
ngx_int_t ngx_http_compile_complex_value(ngx_http_compile_complex_value_t *ccv);
char *ngx_http_set_complex_value_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


ngx_int_t ngx_http_test_predicates(ngx_http_request_t *r,
    ngx_array_t *predicates);
char *ngx_http_set_predicate_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);

ngx_uint_t ngx_http_script_variables_count(ngx_str_t *value);
ngx_int_t ngx_http_script_compile(ngx_http_script_compile_t *sc);
u_char *ngx_http_script_run(ngx_http_request_t *r, ngx_str_t *value,
    void *code_lengths, size_t reserved, void *code_values);
void ngx_http_script_flush_no_cacheable_variables(ngx_http_request_t *r,
    ngx_array_t *indices);

void *ngx_http_script_start_code(ngx_pool_t *pool, ngx_array_t **codes,
    size_t size);
void *ngx_http_script_add_code(ngx_array_t *codes, size_t size, void *code);

size_t ngx_http_script_copy_len_code(ngx_http_script_engine_t *e);
void ngx_http_script_copy_code(ngx_http_script_engine_t *e);
size_t ngx_http_script_copy_var_len_code(ngx_http_script_engine_t *e);
void ngx_http_script_copy_var_code(ngx_http_script_engine_t *e);
size_t ngx_http_script_copy_capture_len_code(ngx_http_script_engine_t *e);
void ngx_http_script_copy_capture_code(ngx_http_script_engine_t *e);
size_t ngx_http_script_mark_args_code(ngx_http_script_engine_t *e);
void ngx_http_script_start_args_code(ngx_http_script_engine_t *e);
#if (NGX_PCRE)
void ngx_http_script_regex_start_code(ngx_http_script_engine_t *e);
void ngx_http_script_regex_end_code(ngx_http_script_engine_t *e);
#endif
void ngx_http_script_return_code(ngx_http_script_engine_t *e);
void ngx_http_script_break_code(ngx_http_script_engine_t *e);
```

```
245 void ngx_http_script_if_code(ngx_http_script_engine_t *e);
246 void ngx_http_script_equal_code(ngx_http_script_engine_t *e);
247 void ngx_http_script_not_equal_code(ngx_http_script_engine_t *e);
248 void ngx_http_script_file_code(ngx_http_script_engine_t *e);
249 void ngx_http_script_complex_value_code(ngx_http_script_engine_t *e);
250 void ngx_http_script_value_code(ngx_http_script_engine_t *e);
251 void ngx_http_script_set_var_code(ngx_http_script_engine_t *e);
252 void ngx_http_script_var_set_handler_code(ngx_http_script_engine_t *e);
253 void ngx_http_script_var_code(ngx_http_script_engine_t *e);
254 void ngx_http_script_nop_code(ngx_http_script_engine_t *e);
255
256
257 #endif /* _NGX_HTTP_SCRIPT_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_script.c - nginx-1.7.10

Global variables defined

- ngx_http_script_exit_code

Functions defined

- ngx_http_compile_complex_value
- ngx_http_complex_value
- ngx_http_script_add_args_code
- ngx_http_script_add_capture_code
- ngx_http_script_add_code
- ngx_http_script_add_copy_code
- ngx_http_script_add_full_name_code
- ngx_http_script_add_var_code
- ngx_http_script_break_code
- ngx_http_script_compile
- ngx_http_script_complex_value_code
- ngx_http_script_copy_capture_code
- ngx_http_script_copy_capture_len_code
- ngx_http_script_copy_code
- ngx_http_script_copy_len_code
- ngx_http_script_copy_var_code
- ngx_http_script_copy_var_len_code
- ngx_http_script_done
- ngx_http_script_equal_code
- ngx_http_script_file_code
- ngx_http_script_flush_complex_value
- ngx_http_script_flush_no_cacheable_variables
- ngx_http_script_full_name_code
- ngx_http_script_full_name_len_code
- ngx_http_script_if_code
- ngx_http_script_init_arrays
- ngx_http_script_mark_args_code

- ngx_http_script_nop_code
- ngx_http_script_not_equal_code
- ngx_http_script_regex_end_code
- ngx_http_script_regex_start_code
- ngx_http_script_return_code
- ngx_http_script_run
- ngx_http_script_set_var_code
- ngx_http_script_start_args_code
- ngx_http_script_start_code
- ngx_http_script_value_code
- ngx_http_script_var_code
- ngx_http_script_var_set_handler_code
- ngx_http_script_variables_count
- ngx_http_set_complex_value_slot
- ngx_http_set_predicate_slot
- ngx_http_test_predicates

Macros defined

- ngx_http_script_exit

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static ngx_int_t ngx_http_script_init_arrays(ngx_http_script_compile_t *sc);
14  static ngx_int_t ngx_http_script_done(ngx_http_script_compile_t *sc);
15  static ngx_int_t ngx_http_script_add_copy_code(ngx_http_script_compile_t *sc,
16      ngx_str_t *value, ngx_uint_t last);
17  static ngx_int_t ngx_http_script_add_var_code(ngx_http_script_compile_t *sc,
18      ngx_str_t *name);
19  static ngx_int_t ngx_http_script_add_args_code(ngx_http_script_compile_t *sc);
20  #if (NGX_PCRE)
21  static ngx_int_t ngx_http_script_add_capture_code(ngx_http_script_compile_t *sc,
22       ngx_uint_t n);
23  #endif
24  static ngx_int_t
25       ngx_http_script_add_full_name_code(ngx_http_script_compile_t *sc);
26  static size_t ngx_http_script_full_name_len_code(ngx_http_script_engine_t *e);
27  static void ngx_http_script_full_name_code(ngx_http_script_engine_t *e);
28
29
30  #define ngx_http_script_exit  (u_char *) &ngx_http_script_exit_code
```

```
31
32  static uintptr_t ngx_http_script_exit_code = (uintptr_t) NULL;
33
34
35  void
36  ngx_http_script_flush_complex_value(ngx_http_request_t *r,
37      ngx_http_complex_value_t *val)
38  {
39      ngx_uint_t *index;
40
41      index = val->flushes;
42
43      if (index) {
44          while (*index != (ngx_uint_t) -1) {
45
46              if (r->variables[*index].no_cacheable) {
47                  r->variables[*index].valid = 0;
48                  r->variables[*index].not_found = 0;
49              }
50
51              index++;
52          }
53      }
54  }
55
56
57  ngx_int_t
58  ngx_http_complex_value(ngx_http_request_t *r, ngx_http_complex_value_t *val,
59      ngx_str_t *value)
60  {
61      size_t                          len;
62      ngx_http_script_code_pt         code;
63      ngx_http_script_len_code_pt     lcode;
64      ngx_http_script_engine_t        e;
65
66      if (val->lengths == NULL) {
67          *value = val->value;
68          return NGX_OK;
69      }
70
71      ngx_http_script_flush_complex_value(r, val);
72
73      ngx_memzero(&e, sizeof(ngx_http_script_engine_t));
74
75      e.ip = val->lengths;
76      e.request = r;
77      e.flushed = 1;
78
79      len = 0;
80
81      while (*(uintptr_t *) e.ip) {
82          lcode = *(ngx_http_script_len_code_pt *) e.ip;
83          len += lcode(&e);
84      }
85
86      value->len = len;
87      value->data = ngx_pnalloc(r->pool, len);
88      if (value->data == NULL) {
89          return NGX_ERROR;
90      }
91
92      e.ip = val->values;
93      e.pos = value->data;
94      e.buf = *value;
95
96      while (*(uintptr_t *) e.ip) {
97          code = *(ngx_http_script_code_pt *) e.ip;
98          code((ngx_http_script_engine_t *) &e);
99      }
100
101      *value = e.buf;
102
103      return NGX_OK;
104  }
105
106
```

```
ngx_int_t
ngx_http_compile_complex_value(ngx_http_compile_complex_value_t *ccv)
{
    ngx_str_t                  *v;
    ngx_uint_t                  i, n, nv, nc;
    ngx_array_t                 flushes, lengths, values, *pf, *pl, *pv;
    ngx_http_script_compile_t   sc;

    v = ccv->value;

    nv = 0;
    nc = 0;

    for (i = 0; i < v->len; i++) {
        if (v->data[i] == '$') {
            if (v->data[i + 1] >= '1' && v->data[i + 1] <= '9') {
                nc++;

            } else {
                nv++;
            }
        }
    }

    if ((v->len == 0 || v->data[0] != '$')
        && (ccv->conf_prefix || ccv->root_prefix))
    {
        if (ngx_conf_full_name(ccv->cf->cycle, v, ccv->conf_prefix) != NGX_OK) {
            return NGX_ERROR;
        }

        ccv->conf_prefix = 0;
        ccv->root_prefix = 0;
    }

    ccv->complex_value->value = *v;
    ccv->complex_value->flushes = NULL;
    ccv->complex_value->lengths = NULL;
    ccv->complex_value->values = NULL;

    if (nv == 0 && nc == 0) {
        return NGX_OK;
    }

    n = nv + 1;

    if (ngx_array_init(&flushes, ccv->cf->pool, n, sizeof(ngx_uint_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    n = nv * (2 * sizeof(ngx_http_script_copy_code_t)
                + sizeof(ngx_http_script_var_code_t))
        + sizeof(uintptr_t);

    if (ngx_array_init(&lengths, ccv->cf->pool, n, 1) != NGX_OK) {
        return NGX_ERROR;
    }

    n = (nv * (2 * sizeof(ngx_http_script_copy_code_t)
                   + sizeof(ngx_http_script_var_code_t))
               + sizeof(uintptr_t)
               + v->len
               + sizeof(uintptr_t) - 1)
            & ~(sizeof(uintptr_t) - 1);

    if (ngx_array_init(&values, ccv->cf->pool, n, 1) != NGX_OK) {
        return NGX_ERROR;
    }

    pf = &flushes;
    pl = &lengths;
    pv = &values;

    ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
```

```
183
184     sc.cf = ccv->cf;
185     sc.source = v;
186     sc.flushes = &pf;
187     sc.lengths = &pl;
188     sc.values = &pv;
189     sc.complete_lengths = 1;
190     sc.complete_values = 1;
191     sc.zero = ccv->zero;
192     sc.conf_prefix = ccv->conf_prefix;
193     sc.root_prefix = ccv->root_prefix;
194
195     if (ngx_http_script_compile(&sc) != NGX_OK) {
196         return NGX_ERROR;
197     }
198
199     if (flushes.nelts) {
200         ccv->complex_value->flushes = flushes.elts;
201         ccv->complex_value->flushes[flushes.nelts] = (ngx_uint_t) -1;
202     }
203
204     ccv->complex_value->lengths = lengths.elts;
205     ccv->complex_value->values = values.elts;
206
207     return NGX_OK;
208 }
209
210
211 char *
212 ngx_http_set_complex_value_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
213 {
214     char  *p = conf;
215
216     ngx_str_t                          *value;
217     ngx_http_complex_value_t          **cv;
218     ngx_http_compile_complex_value_t    ccv;
219
220     cv = (ngx_http_complex_value_t **) (p + cmd->offset);
221
222     if (*cv != NULL) {
223         return "duplicate";
224     }
225
226     *cv = ngx_palloc(cf->pool, sizeof(ngx_http_complex_value_t));
227     if (*cv == NULL) {
228         return NGX_CONF_ERROR;
229     }
230
231     value = cf->args->elts;
232
233     ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
234
235     ccv.cf = cf;
236     ccv.value = &value[1];
237     ccv.complex_value = *cv;
238
239     if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
240         return NGX_CONF_ERROR;
241     }
242
243     return NGX_CONF_OK;
244 }
245
246
247 ngx_int_t
248 ngx_http_test_predicates(ngx_http_request_t *r, ngx_array_t *predicates)
249 {
250     ngx_str_t                  val;
251     ngx_uint_t                 i;
252     ngx_http_complex_value_t  *cv;
253
254     if (predicates == NULL) {
255         return NGX_OK;
256     }
257
258     cv = predicates->elts;
```

```
259
260     for (i = 0; i < predicates->nelts; i++) {
261         if (ngx_http_complex_value(r, &cv[i], &val) != NGX_OK) {
262             return NGX_ERROR;
263         }
264
265         if (val.len && (val.len != 1 || val.data[0] != '0')) {
266             return NGX_DECLINED;
267         }
268     }
269
270     return NGX_OK;
271 }
272
273
274 char *
275 ngx_http_set_predicate_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
276 {
277     char  *p = conf;
278
279     ngx_str_t                          *value;
280     ngx_uint_t                          i;
281     ngx_array_t                        **a;
282     ngx_http_complex_value_t           *cv;
283     ngx_http_compile_complex_value_t    ccv;
284
285     a = (ngx_array_t **) (p + cmd->offset);
286
287     if (*a == NGX_CONF_UNSET_PTR) {
288         *a = ngx_array_create(cf->pool, 1, sizeof(ngx_http_complex_value_t));
289         if (*a == NULL) {
290             return NGX_CONF_ERROR;
291         }
292     }
293
294     value = cf->args->elts;
295
296     for (i = 1; i < cf->args->nelts; i++) {
297         cv = ngx_array_push(*a);
298         if (cv == NULL) {
299             return NGX_CONF_ERROR;
300         }
301
302         ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
303
304         ccv.cf = cf;
305         ccv.value = &value[i];
306         ccv.complex_value = cv;
307
308         if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
309             return NGX_CONF_ERROR;
310         }
311     }
312
313     return NGX_CONF_OK;
314 }
315
316
317 ngx_uint_t
318 ngx_http_script_variables_count(ngx_str_t *value)
319 {
320     ngx_uint_t  i, n;
321
322     for (n = 0, i = 0; i < value->len; i++) {
323         if (value->data[i] == '$') {
324             n++;
325         }
326     }
327
328     return n;
329 }
330
331
332 ngx_int_t
333 ngx_http_script_compile(ngx_http_script_compile_t *sc)
334 {
```

```
    u_char       ch;
    ngx_str_t    name;
    ngx_uint_t   i, bracket;

    if (ngx_http_script_init_arrays(sc) != NGX_OK) {
        return NGX_ERROR;
    }

    for (i = 0; i < sc->source->len; /* void */ ) {

        name.len = 0;

        if (sc->source->data[i] == '$') {

            if (++i == sc->source->len) {
                goto invalid_variable;
            }

#if (NGX_PCRE)
            {
            ngx_uint_t  n;

            if (sc->source->data[i] >= '1' && sc->source->data[i] <= '9') {

                n = sc->source->data[i] - '0';

                if (sc->captures_mask & (1 << n)) {
                    sc->dup_capture = 1;
                }

                sc->captures_mask |= 1 << n;

                if (ngx_http_script_add_capture_code(sc, n) != NGX_OK) {
                    return NGX_ERROR;
                }

                i++;

                continue;
            }
            }
#endif

            if (sc->source->data[i] == '{') {
                bracket = 1;

                if (++i == sc->source->len) {
                    goto invalid_variable;
                }

                name.data = &sc->source->data[i];

            } else {
                bracket = 0;
                name.data = &sc->source->data[i];
            }

            for ( /* void */ ; i < sc->source->len; i++, name.len++) {
                ch = sc->source->data[i];

                if (ch == '}' && bracket) {
                    i++;
                    bracket = 0;
                    break;
                }

                if ((ch >= 'A' && ch <= 'Z')
                    || (ch >= 'a' && ch <= 'z')
                    || (ch >= '0' && ch <= '9')
                    || ch == '_')
                {
                    continue;
                }

                break;
            }
```

```
411
412             if (bracket) {
413                 ngx_conf_log_error(NGX_LOG_EMERG, sc->cf, 0,
414                                    "the closing bracket in \"%V\" "
415                                    "variable is missing", &name);
416                 return NGX_ERROR;
417             }
418
419             if (name.len == 0) {
420                 goto invalid_variable;
421             }
422
423             sc->variables++;
424
425             if (ngx_http_script_add_var_code(sc, &name) != NGX_OK) {
426                 return NGX_ERROR;
427             }
428
429             continue;
430         }
431
432         if (sc->source->data[i] == '?' && sc->compile_args) {
433             sc->args = 1;
434             sc->compile_args = 0;
435
436             if (ngx_http_script_add_args_code(sc) != NGX_OK) {
437                 return NGX_ERROR;
438             }
439
440             i++;
441
442             continue;
443         }
444
445         name.data = &sc->source->data[i];
446
447         while (i < sc->source->len) {
448
449             if (sc->source->data[i] == '$') {
450                 break;
451             }
452
453             if (sc->source->data[i] == '?') {
454
455                 sc->args = 1;
456
457                 if (sc->compile_args) {
458                     break;
459                 }
460             }
461
462             i++;
463             name.len++;
464         }
465
466         sc->size += name.len;
467
468         if (ngx_http_script_add_copy_code(sc, &name, (i == sc->source->len))
469             != NGX_OK)
470         {
471             return NGX_ERROR;
472         }
473     }
474
475     return ngx_http_script_done(sc);
476
477 invalid_variable:
478
479     ngx_conf_log_error(NGX_LOG_EMERG, sc->cf, 0, "invalid variable name");
480
481     return NGX_ERROR;
482 }
483
484
485 u_char *
486 ngx_http_script_run(ngx_http_request_t *r, ngx_str_t *value,
```

```
    void *code_lengths, size_t len, void *code_values)
{
    ngx_uint_t                   i;
    ngx_http_script_code_pt      code;
    ngx_http_script_len_code_pt  lcode;
    ngx_http_script_engine_t     e;
    ngx_http_core_main_conf_t   *cmcf;

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    for (i = 0; i < cmcf->variables.nelts; i++) {
        if (r->variables[i].no_cacheable) {
            r->variables[i].valid = 0;
            r->variables[i].not_found = 0;
        }
    }

    ngx_memzero(&e, sizeof(ngx_http_script_engine_t));

    e.ip = code_lengths;
    e.request = r;
    e.flushed = 1;

    while (*(uintptr_t *) e.ip) {
        lcode = *(ngx_http_script_len_code_pt *) e.ip;
        len += lcode(&e);
    }


    value->len = len;
    value->data = ngx_pnalloc(r->pool, len);
    if (value->data == NULL) {
        return NULL;
    }

    e.ip = code_values;
    e.pos = value->data;

    while (*(uintptr_t *) e.ip) {
        code = *(ngx_http_script_code_pt *) e.ip;
        code((ngx_http_script_engine_t *) &e);
    }

    return e.pos;
}


void
ngx_http_script_flush_no_cacheable_variables(ngx_http_request_t *r,
    ngx_array_t *indices)
{
    ngx_uint_t  n, *index;

    if (indices) {
        index = indices->elts;
        for (n = 0; n < indices->nelts; n++) {
            if (r->variables[index[n]].no_cacheable) {
                r->variables[index[n]].valid = 0;
                r->variables[index[n]].not_found = 0;
            }
        }
    }
}


static ngx_int_t
ngx_http_script_init_arrays(ngx_http_script_compile_t *sc)
{
    ngx_uint_t   n;

    if (sc->flushes && *sc->flushes == NULL) {
        n = sc->variables ? sc->variables : 1;
        *sc->flushes = ngx_array_create(sc->cf->pool, n, sizeof(ngx_uint_t));
        if (*sc->flushes == NULL) {
            return NGX_ERROR;
        }
```

```
    }

    if (*sc->lengths == NULL) {
        n = sc->variables * (2 * sizeof(ngx_http_script_copy_code_t)
                             + sizeof(ngx_http_script_var_code_t))
            + sizeof(uintptr_t);

        *sc->lengths = ngx_array_create(sc->cf->pool, n, 1);
        if (*sc->lengths == NULL) {
            return NGX_ERROR;
        }
    }

    if (*sc->values == NULL) {
        n = (sc->variables * (2 * sizeof(ngx_http_script_copy_code_t)
                              + sizeof(ngx_http_script_var_code_t))
                + sizeof(uintptr_t)
                + sc->source->len
                + sizeof(uintptr_t) - 1)
            & ~(sizeof(uintptr_t) - 1);

        *sc->values = ngx_array_create(sc->cf->pool, n, 1);
        if (*sc->values == NULL) {
            return NGX_ERROR;
        }
    }

    sc->variables = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_http_script_done(ngx_http_script_compile_t *sc)
{
    ngx_str_t    zero;
    uintptr_t   *code;

    if (sc->zero) {

        zero.len = 1;
        zero.data = (u_char *) "\0";

        if (ngx_http_script_add_copy_code(sc, &zero, 0) != NGX_OK) {
            return NGX_ERROR;
        }
    }

    if (sc->conf_prefix || sc->root_prefix) {
        if (ngx_http_script_add_full_name_code(sc) != NGX_OK) {
            return NGX_ERROR;
        }
    }

    if (sc->complete_lengths) {
        code = ngx_http_script_add_code(*sc->lengths, sizeof(uintptr_t), NULL);
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;
    }

    if (sc->complete_values) {
        code = ngx_http_script_add_code(*sc->values, sizeof(uintptr_t),
                                        &sc->main);
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;
    }

    return NGX_OK;
}
```

```
639
640
641 void *
642 ngx_http_script_start_code(ngx_pool_t *pool, ngx_array_t **codes, size_t size)
643 {
644     if (*codes == NULL) {
645         *codes = ngx_array_create(pool, 256, 1);
646         if (*codes == NULL) {
647             return NULL;
648         }
649     }
650
651     return ngx_array_push_n(*codes, size);
652 }
653
654
655 void *
656 ngx_http_script_add_code(ngx_array_t *codes, size_t size, void *code)
657 {
658     u_char  *elts, **p;
659     void    *new;
660
661     elts = codes->elts;
662
663     new = ngx_array_push_n(codes, size);
664     if (new == NULL) {
665         return NULL;
666     }
667
668     if (code) {
669         if (elts != codes->elts) {
670             p = code;
671             *p += (u_char *) codes->elts - elts;
672         }
673     }
674
675     return new;
676 }
677
678
679 static ngx_int_t
680 ngx_http_script_add_copy_code(ngx_http_script_compile_t *sc, ngx_str_t *value,
681     ngx_uint_t last)
682 {
683     u_char                      *p;
684     size_t                       size, len, zero;
685     ngx_http_script_copy_code_t  *code;
686
687     zero = (sc->zero && last);
688     len = value->len + zero;
689
690     code = ngx_http_script_add_code(*sc->lengths,
691                                     sizeof(ngx_http_script_copy_code_t), NULL);
692     if (code == NULL) {
693         return NGX_ERROR;
694     }
695
696     code->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
697     code->len = len;
698
699     size = (sizeof(ngx_http_script_copy_code_t) + len + sizeof(uintptr_t) - 1)
700            & ~(sizeof(uintptr_t) - 1);
701
702     code = ngx_http_script_add_code(*sc->values, size, &sc->main);
703     if (code == NULL) {
704         return NGX_ERROR;
705     }
706
707     code->code = ngx_http_script_copy_code;
708     code->len = len;
709
710     p = ngx_cpymem((u_char *) code + sizeof(ngx_http_script_copy_code_t),
711                    value->data, value->len);
712
713     if (zero) {
714         *p = '\0';
```

```
        sc->zero = 0;
    }

    return NGX_OK;
}


size_t
ngx_http_script_copy_len_code(ngx_http_script_engine_t *e)
{
    ngx_http_script_copy_code_t  *code;

    code = (ngx_http_script_copy_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_copy_code_t);

    return code->len;
}


void
ngx_http_script_copy_code(ngx_http_script_engine_t *e)
{
    u_char                       *p;
    ngx_http_script_copy_code_t  *code;

    code = (ngx_http_script_copy_code_t *) e->ip;

    p = e->pos;

    if (!e->skip) {
        e->pos = ngx_copy(p, e->ip + sizeof(ngx_http_script_copy_code_t),
                          code->len);
    }

    e->ip += sizeof(ngx_http_script_copy_code_t)
          + ((code->len + sizeof(uintptr_t) - 1) & ~(sizeof(uintptr_t) - 1));

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script copy: \"%*s\"", e->pos - p, p);
}


static ngx_int_t
ngx_http_script_add_var_code(ngx_http_script_compile_t *sc, ngx_str_t *name)
{
    ngx_int_t                    index, *p;
    ngx_http_script_var_code_t  *code;

    index = ngx_http_get_variable_index(sc->cf, name);

    if (index == NGX_ERROR) {
        return NGX_ERROR;
    }

    if (sc->flushes) {
        p = ngx_array_push(*sc->flushes);
        if (p == NULL) {
            return NGX_ERROR;
        }

        *p = index;
    }

    code = ngx_http_script_add_code(*sc->lengths,
                                    sizeof(ngx_http_script_var_code_t), NULL);
    if (code == NULL) {
        return NGX_ERROR;
    }

    code->code = (ngx_http_script_code_pt) ngx_http_script_copy_var_len_code;
    code->index = (uintptr_t) index;

    code = ngx_http_script_add_code(*sc->values,
                                    sizeof(ngx_http_script_var_code_t),
                                    &sc->main);
```

```
    if (code == NULL) {
        return NGX_ERROR;
    }

    code->code = ngx_http_script_copy_var_code;
    code->index = (uintptr_t) index;

    return NGX_OK;
}


size_t
ngx_http_script_copy_var_len_code(ngx_http_script_engine_t *e)
{
    ngx_http_variable_value_t   *value;
    ngx_http_script_var_code_t  *code;

    code = (ngx_http_script_var_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_var_code_t);

    if (e->flushed) {
        value = ngx_http_get_indexed_variable(e->request, code->index);

    } else {
        value = ngx_http_get_flushed_variable(e->request, code->index);
    }

    if (value && !value->not_found) {
        return value->len;
    }

    return 0;
}


void
ngx_http_script_copy_var_code(ngx_http_script_engine_t *e)
{
    u_char                      *p;
    ngx_http_variable_value_t   *value;
    ngx_http_script_var_code_t  *code;

    code = (ngx_http_script_var_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_var_code_t);

    if (!e->skip) {

        if (e->flushed) {
            value = ngx_http_get_indexed_variable(e->request, code->index);

        } else {
            value = ngx_http_get_flushed_variable(e->request, code->index);
        }

        if (value && !value->not_found) {
            p = e->pos;
            e->pos = ngx_copy(p, value->data, value->len);

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP,
                           e->request->connection->log, 0,
                           "http script var: \"%*s\"", e->pos - p, p);
        }
    }
}


static ngx_int_t
ngx_http_script_add_args_code(ngx_http_script_compile_t *sc)
{
    uintptr_t   *code;

    code = ngx_http_script_add_code(*sc->lengths, sizeof(uintptr_t), NULL);
    if (code == NULL) {
        return NGX_ERROR;
```

```
    }

    *code = (uintptr_t) ngx_http_script_mark_args_code;

    code = ngx_http_script_add_code(*sc->values, sizeof(uintptr_t), &sc->main);
    if (code == NULL) {
        return NGX_ERROR;
    }

    *code = (uintptr_t) ngx_http_script_start_args_code;

    return NGX_OK;
}


size_t
ngx_http_script_mark_args_code(ngx_http_script_engine_t *e)
{
    e->is_args = 1;
    e->ip += sizeof(uintptr_t);

    return 1;
}


void
ngx_http_script_start_args_code(ngx_http_script_engine_t *e)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script args");

    e->is_args = 1;
    e->args = e->pos;
    e->ip += sizeof(uintptr_t);
}


#if (NGX_PCRE)

void
ngx_http_script_regex_start_code(ngx_http_script_engine_t *e)
{
    size_t                      len;
    ngx_int_t                   rc;
    ngx_uint_t                  n;
    ngx_http_request_t         *r;
    ngx_http_script_engine_t    le;
    ngx_http_script_len_code_pt lcode;
    ngx_http_script_regex_code_t  *code;

    code = (ngx_http_script_regex_code_t *) e->ip;

    r = e->request;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http script regex: \"%V\"", &code->name);

    if (code->uri) {
        e->line = r->uri;
    } else {
        e->sp--;
        e->line.len = e->sp->len;
        e->line.data = e->sp->data;
    }

    rc = ngx_http_regex_exec(r, code->regex, &e->line);

    if (rc == NGX_DECLINED) {
        if (e->log || (r->connection->log->log_level & NGX_LOG_DEBUG_HTTP)) {
            ngx_log_error(NGX_LOG_NOTICE, r->connection->log, 0,
                          "\"%V\" does not match \"%V\"",
                          &code->name, &e->line);
        }

        r->ncaptures = 0;

```

```
        if (code->test) {
            if (code->negative_test) {
                e->sp->len = 1;
                e->sp->data = (u_char *) "1";

            } else {
                e->sp->len = 0;
                e->sp->data = (u_char *) "";
            }

            e->sp++;

            e->ip += sizeof(ngx_http_script_regex_code_t);
            return;
        }

        e->ip += code->next;
        return;
    }

    if (rc == NGX_ERROR) {
        e->ip = ngx_http_script_exit;
        e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
        return;
    }

    if (e->log || (r->connection->log->log_level & NGX_LOG_DEBUG_HTTP)) {
        ngx_log_error(NGX_LOG_NOTICE, r->connection->log, 0,
                      "\"%V\" matches \"%V\"", &code->name, &e->line);
    }

    if (code->test) {
        if (code->negative_test) {
            e->sp->len = 0;
            e->sp->data = (u_char *) "";

        } else {
            e->sp->len = 1;
            e->sp->data = (u_char *) "1";
        }

        e->sp++;

        e->ip += sizeof(ngx_http_script_regex_code_t);
        return;
    }

    if (code->status) {
        e->status = code->status;

        if (!code->redirect) {
            e->ip = ngx_http_script_exit;
            return;
        }
    }

    if (code->uri) {
         r->internal = 1;
         r->valid_unparsed_uri = 0;

         if (code->break_cycle) {
             r->valid_location = 0;
             r->uri_changed = 0;

         } else {
             r->uri_changed = 1;
         }
     }

     if (code->lengths == NULL) {
         e->buf.len = code->size;

         if (code->uri) {
             if (r->ncaptures && (r->quoted_uri || r->plus_in_uri)) {
                 e->buf.len += 2 * ngx_escape_uri(NULL, r->uri.data, r->uri.len,
                                                  NGX_ESCAPE_ARGS);
```

```
            }
        }

        for (n = 2; n < r->ncaptures; n += 2) {
            e->buf.len += r->captures[n + 1] - r->captures[n];
        }

    } else {
        ngx_memzero(&le, sizeof(ngx_http_script_engine_t));

        le.ip = code->lengths->elts;
        le.line = e->line;
        le.request = r;
        le.quote = code->redirect;

        len = 0;

        while (*(uintptr_t *) le.ip) {
            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            len += lcode(&le);
        }

        e->buf.len = len;
    }

    if (code->add_args && r->args.len) {
        e->buf.len += r->args.len + 1;
    }

    e->buf.data = ngx_pnalloc(r->pool, e->buf.len);
    if (e->buf.data == NULL) {
        e->ip = ngx_http_script_exit;
        e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
        return;
    }

    e->quote = code->redirect;

    e->pos = e->buf.data;

    e->ip += sizeof(ngx_http_script_regex_code_t);
}


void
ngx_http_script_regex_end_code(ngx_http_script_engine_t *e)
{
    u_char                            *dst, *src;
    ngx_http_request_t                *r;
    ngx_http_script_regex_end_code_t  *code;

    code = (ngx_http_script_regex_end_code_t *) e->ip;

    r = e->request;

    e->quote = 0;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http script regex end");

    if (code->redirect) {

        dst = e->buf.data;
        src = e->buf.data;

        ngx_unescape_uri(&dst, &src, e->pos - e->buf.data,
                         NGX_UNESCAPE_REDIRECT);

        if (src < e->pos) {
            dst = ngx_movemem(dst, src, e->pos - src);
        }

        e->pos = dst;

        if (code->add_args && r->args.len) {
            *e->pos++ = (u_char) (code->args ? '&' : '?');
```

```
            e->pos = ngx_copy(e->pos, r->args.data, r->args.len);
        }

        e->buf.len = e->pos - e->buf.data;

        if (e->log || (r->connection->log->log_level & NGX_LOG_DEBUG_HTTP)) {
            ngx_log_error(NGX_LOG_NOTICE, r->connection->log, 0,
                          "rewritten redirect: \"%V\"", &e->buf);
        }

        ngx_http_clear_location(r);

        r->headers_out.location = ngx_list_push(&r->headers_out.headers);
        if (r->headers_out.location == NULL) {
            e->ip = ngx_http_script_exit;
            e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
            return;
        }

        r->headers_out.location->hash = 1;
        ngx_str_set(&r->headers_out.location->key, "Location");
        r->headers_out.location->value = e->buf;

        e->ip += sizeof(ngx_http_script_regex_end_code_t);
        return;
    }

    if (e->args) {
        e->buf.len = e->args - e->buf.data;

        if (code->add_args && r->args.len) {
            *e->pos++ = '&';
            e->pos = ngx_copy(e->pos, r->args.data, r->args.len);
        }

        r->args.len = e->pos - e->args;
        r->args.data = e->args;

        e->args = NULL;

    } else {
        e->buf.len = e->pos - e->buf.data;

        if (!code->add_args) {
            r->args.len = 0;
        }
    }

    if (e->log || (r->connection->log->log_level & NGX_LOG_DEBUG_HTTP)) {
        ngx_log_error(NGX_LOG_NOTICE, r->connection->log, 0,
                      "rewritten data: \"%V\", args: \"%V\"",
                      &e->buf, &r->args);
    }

    if (code->uri) {
        r->uri = e->buf;

        if (r->uri.len == 0) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "the rewritten URI has a zero length");
            e->ip = ngx_http_script_exit;
            e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
            return;
        }

        ngx_http_set_exten(r);
    }

    e->ip += sizeof(ngx_http_script_regex_end_code_t);
}


static ngx_int_t
ngx_http_script_add_capture_code(ngx_http_script_compile_t *sc, ngx_uint_t n)
{
    ngx_http_script_copy_capture_code_t  *code;
```

```
1171
1172        code = ngx_http_script_add_code(*sc->lengths,
1173                                        sizeof(ngx_http_script_copy_capture_code_t),
1174                                        NULL);
1175        if (code == NULL) {
1176            return NGX_ERROR;
1177        }
1178
1179        code->code = (ngx_http_script_code_pt)
1180                          ngx_http_script_copy_capture_len_code;
1181        code->n = 2 * n;
1182
1183
1184        code = ngx_http_script_add_code(*sc->values,
1185                                        sizeof(ngx_http_script_copy_capture_code_t),
1186                                        &sc->main);
1187        if (code == NULL) {
1188            return NGX_ERROR;
1189        }
1190
1191        code->code = ngx_http_script_copy_capture_code;
1192        code->n = 2 * n;
1193
1194        if (sc->ncaptures < n) {
1195            sc->ncaptures = n;
1196        }
1197
1198        return NGX_OK;
1199    }
1200
1201
1202    size_t
1203    ngx_http_script_copy_capture_len_code(ngx_http_script_engine_t *e)
1204    {
1205        int                                 *cap;
1206        u_char                              *p;
1207        ngx_uint_t                           n;
1208        ngx_http_request_t                  *r;
1209        ngx_http_script_copy_capture_code_t  *code;
1210
1211        r = e->request;
1212
1213        code = (ngx_http_script_copy_capture_code_t *) e->ip;
1214
1215        e->ip += sizeof(ngx_http_script_copy_capture_code_t);
1216
1217        n = code->n;
1218
1219        if (n < r->ncaptures) {
1220
1221            cap = r->captures;
1222
1223            if ((e->is_args || e->quote)
1224                && (e->request->quoted_uri || e->request->plus_in_uri))
1225            {
1226                p = r->captures_data;
1227
1228                return cap[n + 1] - cap[n]
1229                       + 2 * ngx_escape_uri(NULL, &p[cap[n]], cap[n + 1] - cap[n],
1230                                            NGX_ESCAPE_ARGS);
1231            } else {
1232                return cap[n + 1] - cap[n];
1233            }
1234        }
1235
1236        return 0;
1237    }
1238
1239
1240    void
1241    ngx_http_script_copy_capture_code(ngx_http_script_engine_t *e)
1242    {
1243        int                                 *cap;
1244        u_char                              *p, *pos;
1245        ngx_uint_t                           n;
1246        ngx_http_request_t                  *r;
```

```
    ngx_http_script_copy_capture_code_t  *code;

    r = e->request;

    code = (ngx_http_script_copy_capture_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_copy_capture_code_t);

    n = code->n;

    pos = e->pos;

    if (n < r->ncaptures) {

        cap = r->captures;
        p = r->captures_data;

        if ((e->is_args || e->quote)
            && (e->request->quoted_uri || e->request->plus_in_uri))
        {
            e->pos = (u_char *) ngx_escape_uri(pos, &p[cap[n]],
                                               cap[n + 1] - cap[n],
                                               NGX_ESCAPE_ARGS);
        } else {
            e->pos = ngx_copy(pos, &p[cap[n]], cap[n + 1] - cap[n]);
        }
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script capture: \"%*s\"", e->pos - pos, pos);
}

#endif


static ngx_int_t
ngx_http_script_add_full_name_code(ngx_http_script_compile_t *sc)
{
    ngx_http_script_full_name_code_t  *code;

    code = ngx_http_script_add_code(*sc->lengths,
                                    sizeof(ngx_http_script_full_name_code_t),
                                    NULL);
    if (code == NULL) {
        return NGX_ERROR;
    }

    code->code = (ngx_http_script_code_pt) ngx_http_script_full_name_len_code;
    code->conf_prefix = sc->conf_prefix;

    code = ngx_http_script_add_code(*sc->values,
                                    sizeof(ngx_http_script_full_name_code_t),
                                    &sc->main);
    if (code == NULL) {
        return NGX_ERROR;
    }

    code->code = ngx_http_script_full_name_code;
    code->conf_prefix = sc->conf_prefix;

    return NGX_OK;
}


static size_t
ngx_http_script_full_name_len_code(ngx_http_script_engine_t *e)
{
    ngx_http_script_full_name_code_t  *code;

    code = (ngx_http_script_full_name_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_full_name_code_t);

    return code->conf_prefix ? ngx_cycle->conf_prefix.len:
                               ngx_cycle->prefix.len;
}
```

```
1323
1324
1325 static void
1326 ngx_http_script_full_name_code(ngx_http_script_engine_t *e)
1327 {
1328     ngx_http_script_full_name_code_t  *code;
1329
1330     ngx_str_t  value, *prefix;
1331
1332     code = (ngx_http_script_full_name_code_t *) e->ip;
1333
1334     value.data = e->buf.data;
1335     value.len = e->pos - e->buf.data;
1336
1337     prefix = code->conf_prefix ? (ngx_str_t *) &ngx_cycle->conf_prefix:
1338                                  (ngx_str_t *) &ngx_cycle->prefix;
1339
1340     if (ngx_get_full_name(e->request->pool, prefix, &value) != NGX_OK) {
1341         e->ip = ngx_http_script_exit;
1342         e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
1343         return;
1344     }
1345
1346     e->buf = value;
1347
1348     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
1349                    "http script fullname: \"%V\"", &value);
1350
1351     e->ip += sizeof(ngx_http_script_full_name_code_t);
1352 }
1353
1354
1355 void
1356 ngx_http_script_return_code(ngx_http_script_engine_t *e)
1357 {
1358     ngx_http_script_return_code_t  *code;
1359
1360     code = (ngx_http_script_return_code_t *) e->ip;
1361
1362     if (code->status < NGX_HTTP_BAD_REQUEST
1363         || code->text.value.len
1364         || code->text.lengths)
1365     {
1366         e->status = ngx_http_send_response(e->request, code->status, NULL,
1367                                            &code->text);
1368     } else {
1369         e->status = code->status;
1370     }
1371
1372     e->ip = ngx_http_script_exit;
1373 }
1374
1375
1376 void
1377 ngx_http_script_break_code(ngx_http_script_engine_t *e)
1378 {
1379     e->request->uri_changed = 0;
1380
1381     e->ip = ngx_http_script_exit;
1382 }
1383
1384
1385 void
1386 ngx_http_script_if_code(ngx_http_script_engine_t *e)
1387 {
1388     ngx_http_script_if_code_t  *code;
1389
1390     code = (ngx_http_script_if_code_t *) e->ip;
1391
1392     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
1393                    "http script if");
1394
1395     e->sp--;
1396
1397     if (e->sp->len && (e->sp->len != 1 || e->sp->data[0] != '0')) {
1398         if (code->loc_conf) {
```

```
            e->request->loc_conf = code->loc_conf;
            ngx_http_update_location_config(e->request);
        }

        e->ip += sizeof(ngx_http_script_if_code_t);
        return;
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script if: false");

    e->ip += code->next;
}


void
ngx_http_script_equal_code(ngx_http_script_engine_t *e)
{
    ngx_http_variable_value_t  *val, *res;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script equal");

    e->sp--;
    val = e->sp;
    res = e->sp - 1;

    e->ip += sizeof(uintptr_t);

    if (val->len == res->len
        && ngx_strncmp(val->data, res->data, res->len) == 0)
    {
        *res = ngx_http_variable_true_value;
        return;
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script equal: no");

    *res = ngx_http_variable_null_value;
}


void
ngx_http_script_not_equal_code(ngx_http_script_engine_t *e)
{
    ngx_http_variable_value_t  *val, *res;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script not equal");

    e->sp--;
    val = e->sp;
    res = e->sp - 1;

    e->ip += sizeof(uintptr_t);

    if (val->len == res->len
        && ngx_strncmp(val->data, res->data, res->len) == 0)
    {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                       "http script not equal: no");

        *res = ngx_http_variable_null_value;
        return;
    }

    *res = ngx_http_variable_true_value;
}


void
ngx_http_script_file_code(ngx_http_script_engine_t *e)
{
    ngx_str_t                   path;
    ngx_http_request_t         *r;
```

```
    ngx_open_file_info_t          of;
    ngx_http_core_loc_conf_t     *clcf;
    ngx_http_variable_value_t    *value;
    ngx_http_script_file_code_t  *code;

    value = e->sp - 1;

    code = (ngx_http_script_file_code_t *) e->ip;
    e->ip += sizeof(ngx_http_script_file_code_t);

    path.len = value->len - 1;
    path.data = value->data;

    r = e->request;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http script file op %p \"%V\"", code->op, &path);

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.read_ahead = clcf->read_ahead;
    of.directio = clcf->directio;
    of.valid = clcf->open_file_cache_valid;
    of.min_uses = clcf->open_file_cache_min_uses;
    of.test_only = 1;
    of.errors = clcf->open_file_cache_errors;
    of.events = clcf->open_file_cache_events;

    if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
        e->ip = ngx_http_script_exit;
        e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
        return;
    }

    if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
        != NGX_OK)
    {
        if (of.err != NGX_ENOENT
            && of.err != NGX_ENOTDIR
            && of.err != NGX_ENAMETOOLONG)
        {
            ngx_log_error(NGX_LOG_CRIT, r->connection->log, of.err,
                          "%s \"%s\" failed", of.failed, value->data);
        }

        switch (code->op) {

        case ngx_http_script_file_plain:
        case ngx_http_script_file_dir:
        case ngx_http_script_file_exists:
        case ngx_http_script_file_exec:
             goto false_value;

        case ngx_http_script_file_not_plain:
        case ngx_http_script_file_not_dir:
        case ngx_http_script_file_not_exists:
        case ngx_http_script_file_not_exec:
             goto true_value;
        }

        goto false_value;
    }

    switch (code->op) {
    case ngx_http_script_file_plain:
        if (of.is_file) {
             goto true_value;
        }
        goto false_value;

    case ngx_http_script_file_not_plain:
        if (of.is_file) {
            goto false_value;
        }
```

```
1551            goto true_value;
1552
1553        case ngx_http_script_file_dir:
1554            if (of.is_dir) {
1555                goto true_value;
1556            }
1557            goto false_value;
1558
1559        case ngx_http_script_file_not_dir:
1560            if (of.is_dir) {
1561                goto false_value;
1562            }
1563            goto true_value;
1564
1565        case ngx_http_script_file_exists:
1566            if (of.is_file || of.is_dir || of.is_link) {
1567                goto true_value;
1568            }
1569            goto false_value;
1570
1571        case ngx_http_script_file_not_exists:
1572            if (of.is_file || of.is_dir || of.is_link) {
1573                goto false_value;
1574            }
1575            goto true_value;
1576
1577        case ngx_http_script_file_exec:
1578            if (of.is_exec) {
1579                goto true_value;
1580            }
1581            goto false_value;
1582
1583        case ngx_http_script_file_not_exec:
1584            if (of.is_exec) {
1585                goto false_value;
1586            }
1587            goto true_value;
1588        }
1589
1590    false_value:
1591
1592        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1593                       "http script file op false");
1594
1595        *value = ngx_http_variable_null_value;
1596        return;
1597
1598    true_value:
1599
1600        *value = ngx_http_variable_true_value;
1601        return;
1602    }
1603
1604
1605    void
1606    ngx_http_script_complex_value_code(ngx_http_script_engine_t *e)
1607    {
1608        size_t                            len;
1609        ngx_http_script_engine_t          le;
1610        ngx_http_script_len_code_pt       lcode;
1611        ngx_http_script_complex_value_code_t  *code;
1612
1613        code = (ngx_http_script_complex_value_code_t *) e->ip;
1614
1615        e->ip += sizeof(ngx_http_script_complex_value_code_t);
1616
1617        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
1618                       "http script complex value");
1619
1620        ngx_memzero(&le, sizeof(ngx_http_script_engine_t));
1621
1622        le.ip = code->lengths->elts;
1623        le.line = e->line;
1624        le.request = e->request;
1625        le.quote = e->quote;
1626
```

```
    for (len = 0; *(uintptr_t *) le.ip; len += lcode(&le)) {
        lcode = *(ngx_http_script_len_code_pt *) le.ip;
    }

    e->buf.len = len;
    e->buf.data = ngx_pnalloc(e->request->pool, len);
    if (e->buf.data == NULL) {
        e->ip = ngx_http_script_exit;
        e->status = NGX_HTTP_INTERNAL_SERVER_ERROR;
        return;
    }

    e->pos = e->buf.data;

    e->sp->len = e->buf.len;
    e->sp->data = e->buf.data;
    e->sp++;
}


void
ngx_http_script_value_code(ngx_http_script_engine_t *e)
{
    ngx_http_script_value_code_t  *code;

    code = (ngx_http_script_value_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_value_code_t);

    e->sp->len = code->text_len;
    e->sp->data = (u_char *) code->text_data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script value: \"%v\"", e->sp);

    e->sp++;
}


void
ngx_http_script_set_var_code(ngx_http_script_engine_t *e)
{
    ngx_http_request_t          *r;
    ngx_http_script_var_code_t  *code;

    code = (ngx_http_script_var_code_t *) e->ip;

    e->ip += sizeof(ngx_http_script_var_code_t);

    r = e->request;

    e->sp--;

    r->variables[code->index].len = e->sp->len;
    r->variables[code->index].valid = 1;
    r->variables[code->index].no_cacheable = 0;
    r->variables[code->index].not_found = 0;
    r->variables[code->index].data = e->sp->data;

#if (NGX_DEBUG)
    {
    ngx_http_variable_t        *v;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    v = cmcf->variables.elts;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
                   "http script set $%V", &v[code->index].name);
    }
#endif
}


void
```

```
1703 ngx_http_script_var_set_handler_code(ngx_http_script_engine_t *e)
1704 {
1705     ngx_http_script_var_handler_code_t  *code;
1706
1707     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
1708                    "http script set var handler");
1709
1710     code = (ngx_http_script_var_handler_code_t *) e->ip;
1711
1712     e->ip += sizeof(ngx_http_script_var_handler_code_t);
1713
1714     e->sp--;
1715
1716     code->handler(e->request, e->sp, code->data);
1717 }
1718
1719
1720 void
1721 ngx_http_script_var_code(ngx_http_script_engine_t *e)
1722 {
1723     ngx_http_variable_value_t   *value;
1724     ngx_http_script_var_code_t  *code;
1725
1726     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
1727                    "http script var");
1728
1729     code = (ngx_http_script_var_code_t *) e->ip;
1730
1731     e->ip += sizeof(ngx_http_script_var_code_t);
1732
1733     value = ngx_http_get_flushed_variable(e->request, code->index);
1734
1735     if (value && !value->not_found) {
1736         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, e->request->connection->log, 0,
1737                        "http script var: \"%v\"", value);
1738
1739         *e->sp = *value;
1740         e->sp++;
1741
1742         return;
1743     }
1744
1745     *e->sp = ngx_http_variable_null_value;
1746     e->sp++;
1747 }
1748
1749
1750 void
1751 ngx_http_script_nop_code(ngx_http_script_engine_t *e)
1752 {
1753     e->ip += sizeof(uintptr_t);
1754 }
```

One Level Up Top Level

src/http/modules/ngx_http_userid_filter_module.c - nginx-1.7.10

Global variables defined

- expires
- ngx_http_next_header_filter
- ngx_http_userid_commands
- ngx_http_userid_domain_p
- ngx_http_userid_filter_module
- ngx_http_userid_filter_module_ctx
- ngx_http_userid_got
- ngx_http_userid_p3p_p
- ngx_http_userid_path_p
- ngx_http_userid_reset
- ngx_http_userid_reset_index
- ngx_http_userid_set
- ngx_http_userid_state
- sequencer_v1
- sequencer_v2
- start_value

Data types defined

- ngx_http_userid_conf_t
- ngx_http_userid_ctx_t

Functions defined

- ngx_http_userid_add_variables
- ngx_http_userid_create_conf
- ngx_http_userid_create_uid
- ngx_http_userid_domain
- ngx_http_userid_expires
- ngx_http_userid_filter
- ngx_http_userid_get_uid
- ngx_http_userid_got_variable
- ngx_http_userid_init

- ngx_http_userid_init_worker
- ngx_http_userid_mark
- ngx_http_userid_merge_conf
- ngx_http_userid_p3p
- ngx_http_userid_path
- ngx_http_userid_reset_variable
- ngx_http_userid_set_uid
- ngx_http_userid_set_variable
- ngx_http_userid_variable

Macros defined

- NGX_HTTP_USERID_LOG
- NGX_HTTP_USERID_MAX_EXPIRES
- NGX_HTTP_USERID_OFF
- NGX_HTTP_USERID_ON
- NGX_HTTP_USERID_V1

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define NGX_HTTP_USERID_OFF   0
14  #define NGX_HTTP_USERID_LOG   1
15  #define NGX_HTTP_USERID_V1    2
16  #define NGX_HTTP_USERID_ON    3
17
18  /* 31 Dec 2037 23:55:55 GMT */
19  #define NGX_HTTP_USERID_MAX_EXPIRES  2145916555
20
21
22  typedef struct {
23      ngx_uint_t  enable;
24
25      ngx_int_t   service;
26
27      ngx_str_t   name;
28      ngx_str_t   domain;
29      ngx_str_t   path;
30      ngx_str_t   p3p;
31
32      time_t      expires;
33
34      u_char      mark;
35  } ngx_http_userid_conf_t;
36
37
```

```
typedef struct {
    uint32_t    uid_got[4];
    uint32_t    uid_set[4];
    ngx_str_t   cookie;
    ngx_uint_t  reset;
} ngx_http_userid_ctx_t;


static ngx_http_userid_ctx_t *ngx_http_userid_get_uid(ngx_http_request_t *r,
    ngx_http_userid_conf_t *conf);
static ngx_int_t ngx_http_userid_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, ngx_str_t *name, uint32_t *uid);
static ngx_int_t ngx_http_userid_set_uid(ngx_http_request_t *r,
    ngx_http_userid_ctx_t *ctx, ngx_http_userid_conf_t *conf);
static ngx_int_t ngx_http_userid_create_uid(ngx_http_request_t *r,
    ngx_http_userid_ctx_t *ctx, ngx_http_userid_conf_t *conf);

static ngx_int_t ngx_http_userid_add_variables(ngx_conf_t *cf);
static ngx_int_t ngx_http_userid_init(ngx_conf_t *cf);
static void *ngx_http_userid_create_conf(ngx_conf_t *cf);
static char *ngx_http_userid_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);
static char *ngx_http_userid_domain(ngx_conf_t *cf, void *post, void *data);
static char *ngx_http_userid_path(ngx_conf_t *cf, void *post, void *data);
static char *ngx_http_userid_expires(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_userid_p3p(ngx_conf_t *cf, void *post, void *data);
static char *ngx_http_userid_mark(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static ngx_int_t ngx_http_userid_init_worker(ngx_cycle_t *cycle);



static uint32_t  start_value;
static uint32_t  sequencer_v1 = 1;
static uint32_t  sequencer_v2 = 0x03030302;


static u_char expires[] = "; expires=Thu, 31-Dec-37 23:55:55 GMT";


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;


static ngx_conf_enum_t  ngx_http_userid_state[] = {
    { ngx_string("off"), NGX_HTTP_USERID_OFF },
    { ngx_string("log"), NGX_HTTP_USERID_LOG },
    { ngx_string("v1"), NGX_HTTP_USERID_V1 },
    { ngx_string("on"), NGX_HTTP_USERID_ON },
    { ngx_null_string, 0 }
};


static ngx_conf_post_handler_pt  ngx_http_userid_domain_p =
    ngx_http_userid_domain;
static ngx_conf_post_handler_pt  ngx_http_userid_path_p = ngx_http_userid_path;
static ngx_conf_post_handler_pt  ngx_http_userid_p3p_p = ngx_http_userid_p3p;


static ngx_command_t  ngx_http_userid_commands[] = {

    { ngx_string("userid"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_userid_conf_t, enable),
      ngx_http_userid_state },

     { ngx_string("userid_service"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_userid_conf_t, service),
      NULL },

     { ngx_string("userid_name"),
```

```
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_userid_conf_t, name),
      NULL },

    { ngx_string("userid_domain"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_userid_conf_t, domain),
      &ngx_http_userid_domain_p },

    { ngx_string("userid_path"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_userid_conf_t, path),
      &ngx_http_userid_path_p },

    { ngx_string("userid_expires"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_userid_expires,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("userid_p3p"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_userid_conf_t, p3p),
      &ngx_http_userid_p3p_p },

    { ngx_string("userid_mark"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_userid_mark,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_userid_filter_module_ctx = {
    ngx_http_userid_add_variables,         /* preconfiguration */
    ngx_http_userid_init,                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_userid_create_conf,           /* create location configuration */
    ngx_http_userid_merge_conf             /* merge location configuration */
};


ngx_module_t  ngx_http_userid_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_userid_filter_module_ctx,    /* module context */
    ngx_http_userid_commands,              /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    ngx_http_userid_init_worker,           /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};
```

```
static ngx_str_t   ngx_http_userid_got = ngx_string("uid_got");
static ngx_str_t   ngx_http_userid_set = ngx_string("uid_set");
static ngx_str_t   ngx_http_userid_reset = ngx_string("uid_reset");
static ngx_uint_t  ngx_http_userid_reset_index;


static ngx_int_t
ngx_http_userid_filter(ngx_http_request_t *r)
{
    ngx_http_userid_ctx_t   *ctx;
    ngx_http_userid_conf_t  *conf;

    if (r != r->main) {
        return ngx_http_next_header_filter(r);
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_userid_filter_module);

    if (conf->enable < NGX_HTTP_USERID_V1) {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_http_userid_get_uid(r, conf);

    if (ctx == NULL) {
        return NGX_ERROR;
    }

    if (ngx_http_userid_set_uid(r, ctx, conf) == NGX_OK) {
        return ngx_http_next_header_filter(r);
    }

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_userid_got_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_http_userid_ctx_t   *ctx;
    ngx_http_userid_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r->main, ngx_http_userid_filter_module);

    if (conf->enable == NGX_HTTP_USERID_OFF) {
        v->not_found = 1;
        return NGX_OK;
    }

    ctx = ngx_http_userid_get_uid(r->main, conf);

    if (ctx == NULL) {
        return NGX_ERROR;
    }

    if (ctx->uid_got[3] != 0) {
        return ngx_http_userid_variable(r->main, v, &conf->name, ctx->uid_got);
    }

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_userid_set_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_http_userid_ctx_t   *ctx;
    ngx_http_userid_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r->main, ngx_http_userid_filter_module);

    if (conf->enable < NGX_HTTP_USERID_V1) {
```

```
        v->not_found = 1;
        return NGX_OK;
    }

    ctx = ngx_http_userid_get_uid(r->main, conf);

    if (ctx == NULL) {
        return NGX_ERROR;
    }

    if (ngx_http_userid_create_uid(r->main, ctx, conf) != NGX_OK) {
        return NGX_ERROR;
    }

    if (ctx->uid_set[3] == 0) {
        v->not_found = 1;
        return NGX_OK;
    }

    return ngx_http_userid_variable(r->main, v, &conf->name, ctx->uid_set);
}


static ngx_http_userid_ctx_t *
ngx_http_userid_get_uid(ngx_http_request_t *r, ngx_http_userid_conf_t *conf)
{
    ngx_int_t                n;
    ngx_str_t                src, dst;
    ngx_table_elt_t        **cookies;
    ngx_http_userid_ctx_t   *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_userid_filter_module);

    if (ctx) {
        return ctx;
    }

    if (ctx == NULL) {
        ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_userid_ctx_t));
        if (ctx == NULL) {
            return NULL;
        }

        ngx_http_set_ctx(r, ctx, ngx_http_userid_filter_module);
    }

    n = ngx_http_parse_multi_header_lines(&r->headers_in.cookies, &conf->name,
                                          &ctx->cookie);
    if (n == NGX_DECLINED) {
        return ctx;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "uid cookie: \"%V\"", &ctx->cookie);

    if (ctx->cookie.len < 22) {
        cookies = r->headers_in.cookies.elts;
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "client sent too short userid cookie \"%V\"",
                      &cookies[n]->value);
        return ctx;
    }

    src = ctx->cookie;

    /*
     * we have to limit the encoded string to 22 characters because
     *  1) cookie may be marked by "userid_mark",
     *  2) and there are already the millions cookies with a garbage
     *     instead of the correct base64 trail "=="
     */

    src.len = 22;

    dst.data = (u_char *) ctx->uid_got;

```

```
    if (ngx_decode_base64(&dst, &src) == NGX_ERROR) {
        cookies = r->headers_in.cookies.elts;
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "client sent invalid userid cookie \"%V\"",
                      &cookies[n]->value);
        return ctx;
    }

    ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "uid: %08XD%08XD%08XD%08XD",
                   ctx->uid_got[0], ctx->uid_got[1],
                   ctx->uid_got[2], ctx->uid_got[3]);

    return ctx;
}


static ngx_int_t
ngx_http_userid_set_uid(ngx_http_request_t *r, ngx_http_userid_ctx_t *ctx,
    ngx_http_userid_conf_t *conf)
{
    u_char           *cookie, *p;
    size_t            len;
    ngx_str_t         src, dst;
    ngx_table_elt_t  *set_cookie, *p3p;

    if (ngx_http_userid_create_uid(r, ctx, conf) != NGX_OK) {
        return NGX_ERROR;
    }

    if (ctx->uid_set[3] == 0) {
        return NGX_OK;
    }

    len = conf->name.len + 1 + ngx_base64_encoded_length(16) + conf->path.len;

    if (conf->expires) {
        len += sizeof(expires) - 1 + 2;
    }

    if (conf->domain.len) {
        len += conf->domain.len;
    }

    cookie = ngx_pnalloc(r->pool, len);
    if (cookie == NULL) {
        return NGX_ERROR;
    }

    p = ngx_copy(cookie, conf->name.data, conf->name.len);
    *p++ = '=';

    if (ctx->uid_got[3] == 0 || ctx->reset) {
        src.len = 16;
        src.data = (u_char *) ctx->uid_set;
        dst.data = p;

        ngx_encode_base64(&dst, &src);

        p += dst.len;

        if (conf->mark) {
            *(p - 2) = conf->mark;
        }

    } else {
        p = ngx_cpymem(p, ctx->cookie.data, 22);
        *p++ = conf->mark;
        *p++ = '=';
    }

    if (conf->expires == NGX_HTTP_USERID_MAX_EXPIRES) {
        p = ngx_cpymem(p, expires, sizeof(expires) - 1);

    } else if (conf->expires) {
        p = ngx_cpymem(p, expires, sizeof("; expires=") - 1);
```

```
        p = ngx_http_cookie_time(p, ngx_time() + conf->expires);
    }

    p = ngx_copy(p, conf->domain.data, conf->domain.len);

    p = ngx_copy(p, conf->path.data, conf->path.len);

    set_cookie = ngx_list_push(&r->headers_out.headers);
    if (set_cookie == NULL) {
        return NGX_ERROR;
    }

    set_cookie->hash = 1;
    ngx_str_set(&set_cookie->key, "Set-Cookie");
    set_cookie->value.len = p - cookie;
    set_cookie->value.data = cookie;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "uid cookie: \"%V\"", &set_cookie->value);

    if (conf->p3p.len == 0) {
        return NGX_OK;
    }

    p3p = ngx_list_push(&r->headers_out.headers);
    if (p3p == NULL) {
        return NGX_ERROR;
    }

    p3p->hash = 1;
    ngx_str_set(&p3p->key, "P3P");
    p3p->value = conf->p3p;

    return NGX_OK;
}


static ngx_int_t
ngx_http_userid_create_uid(ngx_http_request_t *r, ngx_http_userid_ctx_t *ctx,
    ngx_http_userid_conf_t *conf)
{
    ngx_connection_t           *c;
    struct sockaddr_in         *sin;
    ngx_http_variable_value_t  *vv;
#if (NGX_HAVE_INET6)
    u_char                     *p;
    struct sockaddr_in6        *sin6;
#endif

    if (ctx->uid_set[3] != 0) {
        return NGX_OK;
    }

    if (ctx->uid_got[3] != 0) {

        vv = ngx_http_get_indexed_variable(r, ngx_http_userid_reset_index);

        if (vv->len == 0 || (vv->len == 1 && vv->data[0] == '0')) {

            if (conf->mark == '\0'
                || (ctx->cookie.len > 23
                    && ctx->cookie.data[22] == conf->mark
                    && ctx->cookie.data[23] == '='))
            {
                return NGX_OK;
            }

            ctx->uid_set[0] = ctx->uid_got[0];
            ctx->uid_set[1] = ctx->uid_got[1];
            ctx->uid_set[2] = ctx->uid_got[2];
            ctx->uid_set[3] = ctx->uid_got[3];

            return NGX_OK;

        } else {
            ctx->reset = 1;
```

```
494
495             if (vv->len == 3 && ngx_strncmp(vv->data, "log", 3) == 0) {
496                 ngx_log_error(NGX_LOG_NOTICE, r->connection->log, 0,
497                         "userid cookie \"%V=%08XD%08XD%08XD%08XD\" was reset",
498                         &conf->name, ctx->uid_got[0], ctx->uid_got[1],
499                         ctx->uid_got[2], ctx->uid_got[3]);
500             }
501         }
502     }
503
504     /*
505      * TODO: in the threaded mode the sequencers should be in TLS and their
506      * ranges should be divided between threads
507      */
508
509     if (conf->enable == NGX_HTTP_USERID_V1) {
510         if (conf->service == NGX_CONF_UNSET) {
511             ctx->uid_set[0] = 0;
512         } else {
513             ctx->uid_set[0] = conf->service;
514         }
515         ctx->uid_set[1] = (uint32_t) ngx_time();
516         ctx->uid_set[2] = start_value;
517         ctx->uid_set[3] = sequencer_v1;
518         sequencer_v1 += 0x100;
519
520     } else {
521         if (conf->service == NGX_CONF_UNSET) {
522
523             c = r->connection;
524
525             if (ngx_connection_local_sockaddr(c, NULL, 0) != NGX_OK) {
526                 return NGX_ERROR;
527             }
528
529             switch (c->local_sockaddr->sa_family) {
530
531 #if (NGX_HAVE_INET6)
532             case AF_INET6:
533                 sin6 = (struct sockaddr_in6 *) c->local_sockaddr;
534
535                 p = (u_char *) &ctx->uid_set[0];
536
537                 *p++ = sin6->sin6_addr.s6_addr[12];
538                 *p++ = sin6->sin6_addr.s6_addr[13];
539                 *p++ = sin6->sin6_addr.s6_addr[14];
540                 *p = sin6->sin6_addr.s6_addr[15];
541
542                 break;
543 #endif
544             default: /* AF_INET */
545                 sin = (struct sockaddr_in *) c->local_sockaddr;
546                 ctx->uid_set[0] = sin->sin_addr.s_addr;
547                 break;
548             }
549
550         } else {
551             ctx->uid_set[0] = htonl(conf->service);
552         }
553
554         ctx->uid_set[1] = htonl((uint32_t) ngx_time());
555         ctx->uid_set[2] = htonl(start_value);
556         ctx->uid_set[3] = htonl(sequencer_v2);
557         sequencer_v2 += 0x100;
558         if (sequencer_v2 < 0x03030302) {
559             sequencer_v2 = 0x03030302;
560         }
561     }
562
563     return NGX_OK;
564 }
565
566
567 static ngx_int_t
568 ngx_http_userid_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
569     ngx_str_t *name, uint32_t *uid)
```

```
{
    v->len = name->len + sizeof("=00001111222233334444555566667777") - 1;
    v->data = ngx_pnalloc(r->pool, v->len);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    ngx_sprintf(v->data, "%V=%08XD%08XD%08XD%08XD",
                name, uid[0], uid[1], uid[2], uid[3]);

    return NGX_OK;
}


static ngx_int_t
ngx_http_userid_reset_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    *v = ngx_http_variable_null_value;

    return NGX_OK;
}


static ngx_int_t
ngx_http_userid_add_variables(ngx_conf_t *cf)
{
    ngx_int_t             n;
    ngx_http_variable_t  *var;

    var = ngx_http_add_variable(cf, &ngx_http_userid_got, 0);
    if (var == NULL) {
        return NGX_ERROR;
    }

    var->get_handler = ngx_http_userid_got_variable;

    var = ngx_http_add_variable(cf, &ngx_http_userid_set, 0);
    if (var == NULL) {
        return NGX_ERROR;
    }

    var->get_handler = ngx_http_userid_set_variable;

    var = ngx_http_add_variable(cf, &ngx_http_userid_reset,
                                NGX_HTTP_VAR_CHANGEABLE);
    if (var == NULL) {
        return NGX_ERROR;
    }

    var->get_handler = ngx_http_userid_reset_variable;

    n = ngx_http_get_variable_index(cf, &ngx_http_userid_reset);
    if (n == NGX_ERROR) {
        return NGX_ERROR;
    }

    ngx_http_userid_reset_index = n;

    return NGX_OK;
}


static void *
ngx_http_userid_create_conf(ngx_conf_t *cf)
{
    ngx_http_userid_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_userid_conf_t));
    if (conf == NULL) {
        return NULL;
    }
```

```
646
647     /*
648      * set by ngx_pcalloc():
649      *
650      *     conf->name = { 0, NULL };
651      *     conf->domain = { 0, NULL };
652      *     conf->path = { 0, NULL };
653      *     conf->p3p = { 0, NULL };
654      */
655
656     conf->enable = NGX_CONF_UNSET_UINT;
657     conf->service = NGX_CONF_UNSET;
658     conf->expires = NGX_CONF_UNSET;
659     conf->mark = (u_char) '\xFF';
660
661     return conf;
662 }
663
664
665 static char *
666 ngx_http_userid_merge_conf(ngx_conf_t *cf, void *parent, void *child)
667 {
668     ngx_http_userid_conf_t *prev = parent;
669     ngx_http_userid_conf_t *conf = child;
670
671     ngx_conf_merge_uint_value(conf->enable, prev->enable,
672                               NGX_HTTP_USERID_OFF);
673
674     ngx_conf_merge_str_value(conf->name, prev->name, "uid");
675     ngx_conf_merge_str_value(conf->domain, prev->domain, "");
676     ngx_conf_merge_str_value(conf->path, prev->path, "; path=/");
677     ngx_conf_merge_str_value(conf->p3p, prev->p3p, "");
678
679     ngx_conf_merge_value(conf->service, prev->service, NGX_CONF_UNSET);
680     ngx_conf_merge_sec_value(conf->expires, prev->expires, 0);
681
682     if (conf->mark == (u_char) '\xFF') {
683         if (prev->mark == (u_char) '\xFF') {
684             conf->mark = '\0';
685         } else {
686             conf->mark = prev->mark;
687         }
688     }
689
690     return NGX_CONF_OK;
691 }
692
693
694 static ngx_int_t
695 ngx_http_userid_init(ngx_conf_t *cf)
696 {
697     ngx_http_next_header_filter = ngx_http_top_header_filter;
698     ngx_http_top_header_filter = ngx_http_userid_filter;
699
700     return NGX_OK;
701 }
702
703
704 static char *
705 ngx_http_userid_domain(ngx_conf_t *cf, void *post, void *data)
706 {
707     ngx_str_t  *domain = data;
708
709     u_char  *p, *new;
710
711     if (ngx_strcmp(domain->data, "none") == 0) {
712         ngx_str_set(domain, "");
713         return NGX_CONF_OK;
714     }
715
716     new = ngx_pnalloc(cf->pool, sizeof("; domain=") - 1 + domain->len);
717     if (new == NULL) {
718         return NGX_CONF_ERROR;
719     }
720
721     p = ngx_cpymem(new, "; domain=", sizeof("; domain=") - 1);
```

```
    ngx_memcpy(p, domain->data, domain->len);

    domain->len += sizeof("; domain=") - 1;
    domain->data = new;

    return NGX_CONF_OK;
}


static char *
ngx_http_userid_path(ngx_conf_t *cf, void *post, void *data)
{
    ngx_str_t  *path = data;

    u_char  *p, *new;

    new = ngx_pnalloc(cf->pool, sizeof("; path=") - 1 + path->len);
    if (new == NULL) {
        return NGX_CONF_ERROR;
    }

    p = ngx_cpymem(new, "; path=", sizeof("; path=") - 1);
    ngx_memcpy(p, path->data, path->len);

    path->len += sizeof("; path=") - 1;
    path->data = new;

    return NGX_CONF_OK;
}


static char *
ngx_http_userid_expires(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_userid_conf_t *ucf = conf;

    ngx_str_t  *value;

    if (ucf->expires != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    if (ngx_strcmp(value[1].data, "max") == 0) {
        ucf->expires = NGX_HTTP_USERID_MAX_EXPIRES;
        return NGX_CONF_OK;
    }

    if (ngx_strcmp(value[1].data, "off") == 0) {
        ucf->expires = 0;
        return NGX_CONF_OK;
    }

    ucf->expires = ngx_parse_time(&value[1], 1);
    if (ucf->expires == (time_t) NGX_ERROR) {
        return "invalid value";
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_userid_p3p(ngx_conf_t *cf, void *post, void *data)
{
    ngx_str_t  *p3p = data;

    if (ngx_strcmp(p3p->data, "none") == 0) {
        ngx_str_set(p3p, "");
    }

    return NGX_CONF_OK;
}
```

```
798 static char *
799 ngx_http_userid_mark(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
800 {
801     ngx_http_userid_conf_t *ucf = conf;
802
803     ngx_str_t  *value;
804
805     if (ucf->mark != (u_char) '\xFF') {
806         return "is duplicate";
807     }
808
809     value = cf->args->elts;
810
811     if (ngx_strcmp(value[1].data, "off") == 0) {
812         ucf->mark = '\0';
813         return NGX_CONF_OK;
814     }
815
816     if (value[1].len != 1
817         || !((value[1].data[0] >= '0' && value[1].data[0] <= '9')
818               || (value[1].data[0] >= 'A' && value[1].data[0] <= 'Z')
819               || (value[1].data[0] >= 'a' && value[1].data[0] <= 'z')
820               || value[1].data[0] == '='))
821     {
822         return "value must be \"off\" or a single letter, digit or \"=\"";
823     }
824
825     ucf->mark = value[1].data[0];
826
827     return NGX_CONF_OK;
828 }
829
830
831 static ngx_int_t
832 ngx_http_userid_init_worker(ngx_cycle_t *cycle)
833 {
834     struct timeval  tp;
835
836     ngx_gettimeofday(&tp);
837
838     /* use the most significant usec part that fits to 16 bits */
839     start_value = ((tp.tv_usec / 20) << 16) | ngx_pid;
840
841     return NGX_OK;
842 }
```

One Level Up Top Level

src/http/ngx_http_spdy_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_header_filter
- ngx_http_spdy_filter_module
- ngx_http_spdy_filter_module_ctx

Functions defined

- ngx_http_spdy_data_frame_handler
- ngx_http_spdy_filter_cleanup
- ngx_http_spdy_filter_get_data_frame
- ngx_http_spdy_filter_get_shadow
- ngx_http_spdy_filter_init
- ngx_http_spdy_filter_send
- ngx_http_spdy_flow_control
- ngx_http_spdy_handle_frame
- ngx_http_spdy_handle_stream
- ngx_http_spdy_header_filter
- ngx_http_spdy_send_chain
- ngx_http_spdy_syn_frame_handler
- ngx_http_spdy_waiting_queue

Macros defined

- ngx_http_spdy_nv_nsize
- ngx_http_spdy_nv_vsize
- ngx_http_spdy_nv_write_name
- ngx_http_spdy_nv_write_nlen
- ngx_http_spdy_nv_write_num
- ngx_http_spdy_nv_write_val
- ngx_http_spdy_nv_write_vlen

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  * Copyright (C) Valentin V. Bartenev
5  */
```

```
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <nginx.h>
12  #include <ngx_http_spdy_module.h>
13
14  #include <zlib.h>
15
16
17  #define ngx_http_spdy_nv_nsize(h)  (NGX_SPDY_NV_NLEN_SIZE + sizeof(h) - 1)
18  #define ngx_http_spdy_nv_vsize(h)  (NGX_SPDY_NV_VLEN_SIZE + sizeof(h) - 1)
19
20  #define ngx_http_spdy_nv_write_num   ngx_spdy_frame_write_uint32
21  #define ngx_http_spdy_nv_write_nlen  ngx_spdy_frame_write_uint32
22  #define ngx_http_spdy_nv_write_vlen  ngx_spdy_frame_write_uint32
23
24  #define ngx_http_spdy_nv_write_name(p, h)                                    \
25      ngx_cpymem(ngx_http_spdy_nv_write_nlen(p, sizeof(h) - 1), h, sizeof(h) - 1)
26
27  #define ngx_http_spdy_nv_write_val(p, h)                                     \
28      ngx_cpymem(ngx_http_spdy_nv_write_vlen(p, sizeof(h) - 1), h, sizeof(h) - 1)
29
30
31  static ngx_chain_t *ngx_http_spdy_send_chain(ngx_connection_t *fc,
32      ngx_chain_t *in, off_t limit);
33
34  static ngx_inline ngx_int_t ngx_http_spdy_filter_send(
35      ngx_connection_t *fc, ngx_http_spdy_stream_t *stream);
36  static ngx_inline ngx_int_t ngx_http_spdy_flow_control(
37      ngx_http_spdy_connection_t *sc, ngx_http_spdy_stream_t *stream);
38  static void ngx_http_spdy_waiting_queue(ngx_http_spdy_connection_t *sc,
39      ngx_http_spdy_stream_t *stream);
40
41  static ngx_chain_t *ngx_http_spdy_filter_get_shadow(
42      ngx_http_spdy_stream_t *stream, ngx_buf_t *buf, off_t offset, off_t size);
43  static ngx_http_spdy_out_frame_t *ngx_http_spdy_filter_get_data_frame(
44      ngx_http_spdy_stream_t *stream, size_t len, ngx_chain_t *first,
45      ngx_chain_t *last);
46
47  static ngx_int_t ngx_http_spdy_syn_frame_handler(
48      ngx_http_spdy_connection_t *sc, ngx_http_spdy_out_frame_t *frame);
49  static ngx_int_t ngx_http_spdy_data_frame_handler(
50      ngx_http_spdy_connection_t *sc, ngx_http_spdy_out_frame_t *frame);
51  static ngx_inline void ngx_http_spdy_handle_frame(
52      ngx_http_spdy_stream_t *stream, ngx_http_spdy_out_frame_t *frame);
53  static ngx_inline void ngx_http_spdy_handle_stream(
54      ngx_http_spdy_connection_t *sc, ngx_http_spdy_stream_t *stream);
55
56  static void ngx_http_spdy_filter_cleanup(void *data);
57
58  static ngx_int_t ngx_http_spdy_filter_init(ngx_conf_t *cf);
59
60
61  static ngx_http_module_t  ngx_http_spdy_filter_module_ctx = {
62      NULL,                                  /* preconfiguration */
63      ngx_http_spdy_filter_init,             /* postconfiguration */
64
65      NULL,                                  /* create main configuration */
66      NULL,                                  /* init main configuration */
67
68      NULL,                                  /* create server configuration */
69      NULL,                                  /* merge server configuration */
70
71      NULL,                                  /* create location configuration */
72      NULL                                   /* merge location configuration */
73  };
74
75
76  ngx_module_t  ngx_http_spdy_filter_module = {
77      NGX_MODULE_V1,
78      &ngx_http_spdy_filter_module_ctx,      /* module context */
79      NULL,                                  /* module directives */
80      NGX_HTTP_MODULE,                       /* module type */
81      NULL,                                  /* init master */
```

```
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;


static ngx_int_t
ngx_http_spdy_header_filter(ngx_http_request_t *r)
{
    int                           rc;
    size_t                        len;
     u_char                       *p, *buf, *last;
     ngx_buf_t                    *b;
     ngx_str_t                     host;
     ngx_uint_t                    i, j, count, port;
     ngx_chain_t                  *cl;
     ngx_list_part_t              *part, *pt;
     ngx_table_elt_t              *header, *h;
     ngx_connection_t             *c;
     ngx_http_cleanup_t           *cln;
     ngx_http_core_loc_conf_t     *clcf;
     ngx_http_core_srv_conf_t     *cscf;
     ngx_http_spdy_stream_t       *stream;
     ngx_http_spdy_out_frame_t    *frame;
     ngx_http_spdy_connection_t   *sc;
     struct sockaddr_in           *sin;
 #if (NGX_HAVE_INET6)
     struct sockaddr_in6          *sin6;
 #endif
     u_char                        addr[NGX_SOCKADDR_STRLEN];

     if (!r->spdy_stream) {
         return ngx_http_next_header_filter(r);
     }

     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                    "spdy header filter");

     if (r->header_sent) {
         return NGX_OK;
     }

     r->header_sent = 1;

     if (r != r->main) {
         return NGX_OK;
     }

     c = r->connection;

     if (r->method == NGX_HTTP_HEAD) {
         r->header_only = 1;
     }

     switch (r->headers_out.status) {

     case NGX_HTTP_OK:
     case NGX_HTTP_PARTIAL_CONTENT:
         break;

     case NGX_HTTP_NOT_MODIFIED:
         r->header_only = 1;
         break;

     case NGX_HTTP_NO_CONTENT:
         r->header_only = 1;

         ngx_str_null(&r->headers_out.content_type);

```

```
        r->headers_out.content_length = NULL;
        r->headers_out.content_length_n = -1;

        /* fall through */

    default:
        r->headers_out.last_modified_time = -1;
        r->headers_out.last_modified = NULL;
    }

    len = NGX_SPDY_NV_NUM_SIZE
          + ngx_http_spdy_nv_nsize(":version")
          + ngx_http_spdy_nv_vsize("HTTP/1.1")
          + ngx_http_spdy_nv_nsize(":status")
          + (r->headers_out.status_line.len
             ? NGX_SPDY_NV_VLEN_SIZE + r->headers_out.status_line.len
             : ngx_http_spdy_nv_vsize("418"));

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (r->headers_out.server == NULL) {
        len += ngx_http_spdy_nv_nsize("server");
        len += clcf->server_tokens ? ngx_http_spdy_nv_vsize(NGINX_VER)
                                   : ngx_http_spdy_nv_vsize("nginx");
    }

    if (r->headers_out.date == NULL) {
        len += ngx_http_spdy_nv_nsize("date")
               + ngx_http_spdy_nv_vsize("Wed, 31 Dec 1986 10:00:00 GMT");
    }

    if (r->headers_out.content_type.len) {
        len += ngx_http_spdy_nv_nsize("content-type")
               + NGX_SPDY_NV_VLEN_SIZE + r->headers_out.content_type.len;

        if (r->headers_out.content_type_len == r->headers_out.content_type.len
            && r->headers_out.charset.len)
        {
            len += sizeof("; charset=") - 1 + r->headers_out.charset.len;
        }
    }

    if (r->headers_out.content_length == NULL
        && r->headers_out.content_length_n >= 0)
    {
        len += ngx_http_spdy_nv_nsize("content-length")
               + NGX_SPDY_NV_VLEN_SIZE + NGX_OFF_T_LEN;
    }

    if (r->headers_out.last_modified == NULL
        && r->headers_out.last_modified_time != -1)
    {
        len += ngx_http_spdy_nv_nsize("last-modified")
               + ngx_http_spdy_nv_vsize("Wed, 31 Dec 1986 10:00:00 GMT");
    }

    if (r->headers_out.location
        && r->headers_out.location->value.len
        && r->headers_out.location->value.data[0] == '/')
    {
        r->headers_out.location->hash = 0;

        if (clcf->server_name_in_redirect) {
            cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
            host = cscf->server_name;

        } else if (r->headers_in.server.len) {
            host = r->headers_in.server;

        } else {
            host.len = NGX_SOCKADDR_STRLEN;
            host.data = addr;

            if (ngx_connection_local_sockaddr(c, &host, 0) != NGX_OK) {
                return NGX_ERROR;
            }
```

```
        }

        switch (c->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            sin6 = (struct sockaddr_in6 *) c->local_sockaddr;
            port = ntohs(sin6->sin6_port);
            break;
#endif
#if (NGX_HAVE_UNIX_DOMAIN)
        case AF_UNIX:
            port = 0;
            break;
#endif
        default: /* AF_INET */
            sin = (struct sockaddr_in *) c->local_sockaddr;
            port = ntohs(sin->sin_port);
            break;
        }

        len += ngx_http_spdy_nv_nsize("location")
               + ngx_http_spdy_nv_vsize("https://")
               + host.len
               + r->headers_out.location->value.len;

        if (clcf->port_in_redirect) {

#if (NGX_HTTP_SSL)
            if (c->ssl)
                port = (port == 443) ? 0 : port;
            else
#endif
                port = (port == 80) ? 0 : port;

        } else {
            port = 0;
        }

        if (port) {
            len += sizeof(":65535") - 1;
        }

    } else {
        ngx_str_null(&host);
        port = 0;
    }

    part = &r->headers_out.headers.part;
    header = part->elts;

    for (i = 0; /* void */; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            header = part->elts;
            i = 0;
        }

        if (header[i].hash == 0) {
            continue;
        }

        len += NGX_SPDY_NV_NLEN_SIZE + header[i].key.len
               + NGX_SPDY_NV_VLEN_SIZE  + header[i].value.len;
    }

    buf = ngx_alloc(len, r->pool->log);
    if (buf == NULL) {
        return NGX_ERROR;
    }

```

```
    last = buf + NGX_SPDY_NV_NUM_SIZE;

    last = ngx_http_spdy_nv_write_name(last, ":version");
    last = ngx_http_spdy_nv_write_val(last, "HTTP/1.1");

    last = ngx_http_spdy_nv_write_name(last, ":status");

    if (r->headers_out.status_line.len) {
        last = ngx_http_spdy_nv_write_vlen(last,
                                           r->headers_out.status_line.len);
        last = ngx_cpymem(last, r->headers_out.status_line.data,
                          r->headers_out.status_line.len);
    } else {
        last = ngx_http_spdy_nv_write_vlen(last, 3);
        last = ngx_sprintf(last, "%03ui", r->headers_out.status);
    }

    count = 2;

    if (r->headers_out.server == NULL) {
        last = ngx_http_spdy_nv_write_name(last, "server");
        last = clcf->server_tokens
               ? ngx_http_spdy_nv_write_val(last, NGINX_VER)
               : ngx_http_spdy_nv_write_val(last, "nginx");

        count++;
    }

    if (r->headers_out.date == NULL) {
        last = ngx_http_spdy_nv_write_name(last, "date");

        last = ngx_http_spdy_nv_write_vlen(last, ngx_cached_http_time.len);

        last = ngx_cpymem(last, ngx_cached_http_time.data,
                          ngx_cached_http_time.len);

        count++;
    }

    if (r->headers_out.content_type.len) {

        last = ngx_http_spdy_nv_write_name(last, "content-type");

        p = last + NGX_SPDY_NV_VLEN_SIZE;

        last = ngx_cpymem(p, r->headers_out.content_type.data,
                          r->headers_out.content_type.len);

        if (r->headers_out.content_type_len == r->headers_out.content_type.len
            && r->headers_out.charset.len)
        {
            last = ngx_cpymem(last, "; charset=", sizeof("; charset=") - 1);

            last = ngx_cpymem(last, r->headers_out.charset.data,
                              r->headers_out.charset.len);

            /* update r->headers_out.content_type for possible logging */

            r->headers_out.content_type.len = last - p;
            r->headers_out.content_type.data = p;
        }

        (void) ngx_http_spdy_nv_write_vlen(p - NGX_SPDY_NV_VLEN_SIZE,
                                           r->headers_out.content_type.len);

        count++;
    }

    if (r->headers_out.content_length == NULL
        && r->headers_out.content_length_n >= 0)
    {
        last = ngx_http_spdy_nv_write_name(last, "content-length");

        p = last + NGX_SPDY_NV_VLEN_SIZE;

        last = ngx_sprintf(p, "%O", r->headers_out.content_length_n);
```

```
386
387         (void) ngx_http_spdy_nv_write_vlen(p - NGX_SPDY_NV_VLEN_SIZE,
388                                            last - p);
389
390         count++;
391     }
392
393     if (r->headers_out.last_modified == NULL
394         && r->headers_out.last_modified_time != -1)
395     {
396         last = ngx_http_spdy_nv_write_name(last, "last-modified");
397
398         p = last + NGX_SPDY_NV_VLEN_SIZE;
399
400         last = ngx_http_time(p, r->headers_out.last_modified_time);
401
402         (void) ngx_http_spdy_nv_write_vlen(p - NGX_SPDY_NV_VLEN_SIZE,
403                                            last - p);
404
405         count++;
406     }
407
408     if (host.data) {
409
410         last = ngx_http_spdy_nv_write_name(last, "location");
411
412         p = last + NGX_SPDY_NV_VLEN_SIZE;
413
414         last = ngx_cpymem(p, "http", sizeof("http") - 1);
415
416 #if (NGX_HTTP_SSL)
417         if (c->ssl) {
418             *last++ ='s';
419         }
420 #endif
421
422         *last++ = ':'; *last++ = '/'; *last++ = '/';
423
424         last = ngx_cpymem(last, host.data, host.len);
425
426         if (port) {
427             last = ngx_sprintf(last, ":%ui", port);
428         }
429
430         last = ngx_cpymem(last, r->headers_out.location->value.data,
431                           r->headers_out.location->value.len);
432
433         /* update r->headers_out.location->value for possible logging */
434
435         r->headers_out.location->value.len = last - p;
436         r->headers_out.location->value.data = p;
437         ngx_str_set(&r->headers_out.location->key, "location");
438
439         (void) ngx_http_spdy_nv_write_vlen(p - NGX_SPDY_NV_VLEN_SIZE,
440                                            r->headers_out.location->value.len);
441
442         count++;
443     }
444
445     part = &r->headers_out.headers.part;
446     header = part->elts;
447
448     for (i = 0; /* void */; i++) {
449
450         if (i >= part->nelts) {
451             if (part->next == NULL) {
452                 break;
453             }
454
455             part = part->next;
456             header = part->elts;
457             i = 0;
458         }
459
460         if (header[i].hash == 0 || header[i].hash == 2) {
461             continue;
```

```
        }

        last = ngx_http_spdy_nv_write_nlen(last, header[i].key.len);

        ngx_strlow(last, header[i].key.data, header[i].key.len);
        last += header[i].key.len;

        p = last + NGX_SPDY_NV_VLEN_SIZE;

        last = ngx_cpymem(p, header[i].value.data, header[i].value.len);

        pt = part;
        h = header;

        for (j = i + 1; /* void */; j++) {

            if (j >= pt->nelts) {
                if (pt->next == NULL) {
                    break;
                }

                pt = pt->next;
                h = pt->elts;
                j = 0;
            }

            if (h[j].hash == 0 || h[j].hash == 2
                || h[j].key.len != header[i].key.len
                || ngx_strncasecmp(header[i].key.data, h[j].key.data,
                                   header[i].key.len))
            {
                continue;
            }

            if (h[j].value.len) {
                if (last != p) {
                    *last++ = '\0';
                }

                last = ngx_cpymem(last, h[j].value.data, h[j].value.len);
            }

            h[j].hash = 2;
        }

        (void) ngx_http_spdy_nv_write_vlen(p - NGX_SPDY_NV_VLEN_SIZE,
                                           last - p);

        count++;
    }

    (void) ngx_http_spdy_nv_write_num(buf, count);

    stream = r->spdy_stream;
    sc = stream->connection;

    len = last - buf;

    b = ngx_create_temp_buf(r->pool, NGX_SPDY_FRAME_HEADER_SIZE
                                     + NGX_SPDY_SYN_REPLY_SIZE
                                     + deflateBound(&sc->zstream_out, len));
    if (b == NULL) {
        ngx_free(buf);
        return NGX_ERROR;
    }

    b->last += NGX_SPDY_FRAME_HEADER_SIZE + NGX_SPDY_SYN_REPLY_SIZE;

    sc->zstream_out.next_in = buf;
    sc->zstream_out.avail_in = len;
    sc->zstream_out.next_out = b->last;
    sc->zstream_out.avail_out = b->end - b->last;

    rc = deflate(&sc->zstream_out, Z_SYNC_FLUSH);

    ngx_free(buf);
```

```
538
539     if (rc != Z_OK) {
540         ngx_log_error(NGX_LOG_ALERT, c->log, 0, "deflate() failed: %d", rc);
541         return NGX_ERROR;
542     }
543
544     ngx_log_debug5(NGX_LOG_DEBUG_HTTP, c->log, 0,
545                    "spdy deflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
546                    sc->zstream_out.next_in, sc->zstream_out.next_out,
547                    sc->zstream_out.avail_in, sc->zstream_out.avail_out,
548                    rc);
549
550     b->last = sc->zstream_out.next_out;
551
552     p = b->pos;
553     p = ngx_spdy_frame_write_head(p, NGX_SPDY_SYN_REPLY);
554
555     len = b->last - b->pos;
556
557     r->header_size = len;
558
559     len -= NGX_SPDY_FRAME_HEADER_SIZE;
560
561     if (r->header_only) {
562         b->last_buf = 1;
563         p = ngx_spdy_frame_write_flags_and_len(p, NGX_SPDY_FLAG_FIN, len);
564
565     } else {
566         p = ngx_spdy_frame_write_flags_and_len(p, 0, len);
567     }
568
569     (void) ngx_spdy_frame_write_sid(p, stream->id);
570
571     cl = ngx_alloc_chain_link(r->pool);
572     if (cl == NULL) {
573         return NGX_ERROR;
574     }
575
576     cl->buf = b;
577     cl->next = NULL;
578
579     frame = ngx_palloc(r->pool, sizeof(ngx_http_spdy_out_frame_t));
580     if (frame == NULL) {
581         return NGX_ERROR;
582     }
583
584     frame->first = cl;
585     frame->last = cl;
586     frame->handler = ngx_http_spdy_syn_frame_handler;
587     frame->stream = stream;
588     frame->length = len;
589     frame->priority = stream->priority;
590     frame->blocked = 1;
591     frame->fin = r->header_only;
592
593     ngx_log_debug3(NGX_LOG_DEBUG_HTTP, stream->request->connection->log, 0,
594                    "spdy:%ui create SYN_REPLY frame %p: len:%uz",
595                    stream->id, frame, frame->length);
596
597     ngx_http_spdy_queue_blocked_frame(sc, frame);
598
599     cln = ngx_http_cleanup_add(r, 0);
600     if (cln == NULL) {
601         return NGX_ERROR;
602     }
603
604     cln->handler = ngx_http_spdy_filter_cleanup;
605     cln->data = stream;
606
607     stream->queued = 1;
608
609     c->send_chain = ngx_http_spdy_send_chain;
610     c->need_last_buf = 1;
611
612     return ngx_http_spdy_filter_send(c, stream);
613 }
```

```
614
615
616 static ngx_chain_t *
617 ngx_http_spdy_send_chain(ngx_connection_t *fc, ngx_chain_t *in, off_t limit)
618 {
619     off_t                       size, offset;
620     size_t                      rest, frame_size;
621     ngx_chain_t                *cl, *out, **ln;
622     ngx_http_request_t         *r;
623     ngx_http_spdy_stream_t     *stream;
624     ngx_http_spdy_loc_conf_t   *slcf;
625     ngx_http_spdy_out_frame_t  *frame;
626     ngx_http_spdy_connection_t  *sc;
627
628     r = fc->data;
629     stream = r->spdy_stream;
630
631 #if (NGX_SUPPRESS_WARN)
632     size = 0;
633 #endif
634
635     while (in) {
636         size = ngx_buf_size(in->buf);
637
638         if (size || in->buf->last_buf) {
639             break;
640         }
641
642         in = in->next;
643     }
644
645     if (in == NULL) {
646
647         if (stream->queued) {
648             fc->write->delayed = 1;
649         } else {
650             fc->buffered &= ~NGX_SPDY_BUFFERED;
651         }
652
653         return NULL;
654     }
655
656     sc = stream->connection;
657
658     if (size && ngx_http_spdy_flow_control(sc, stream) == NGX_DECLINED) {
659         fc->write->delayed = 1;
660         return in;
661     }
662
663     if (limit == 0 || limit > (off_t) sc->send_window) {
664         limit = sc->send_window;
665     }
666
667     if (limit > stream->send_window) {
668         limit = (stream->send_window > 0) ? stream->send_window : 0;
669     }
670
671     if (in->buf->tag == (ngx_buf_tag_t) &ngx_http_spdy_filter_get_shadow) {
672         cl = ngx_alloc_chain_link(r->pool);
673         if (cl == NULL) {
674             return NGX_CHAIN_ERROR;
675         }
676
677         cl->buf = in->buf;
678         in->buf = cl->buf->shadow;
679
680         offset = ngx_buf_in_memory(in->buf)
681                  ? (cl->buf->pos - in->buf->pos)
682                  : (cl->buf->file_pos - in->buf->file_pos);
683
684         cl->next = stream->free_bufs;
685         stream->free_bufs = cl;
686
687     } else {
688         offset = 0;
689     }
```

```
690
691 #if (NGX_SUPPRESS_WARN)
692     cl = NULL;
693 #endif
694
695     slcf = ngx_http_get_module_loc_conf(r, ngx_http_spdy_module);
696
697     frame_size = (limit <= (off_t) slcf->chunk_size) ? (size_t) limit
698                                                      : slcf->chunk_size;
699
700     for ( ;; ) {
701         ln = &out;
702         rest = frame_size;
703
704         while ((off_t) rest >= size) {
705
706             if (offset) {
707                 cl = ngx_http_spdy_filter_get_shadow(stream, in->buf,
708                                                      offset, size);
709                 if (cl == NULL) {
710                     return NGX_CHAIN_ERROR;
711                 }
712
713                 offset = 0;
714
715             } else {
716                 cl = ngx_alloc_chain_link(r->pool);
717                 if (cl == NULL) {
718                     return NGX_CHAIN_ERROR;
719                 }
720
721                 cl->buf = in->buf;
722             }
723
724             *ln = cl;
725             ln = &cl->next;
726
727             rest -= (size_t) size;
728             in = in->next;
729
730             if (in == NULL) {
731                 frame_size -= rest;
732                 rest = 0;
733                 break;
734             }
735
736             size = ngx_buf_size(in->buf);
737         }
738
739         if (rest) {
740             cl = ngx_http_spdy_filter_get_shadow(stream, in->buf,
741                                                  offset, rest);
742             if (cl == NULL) {
743                 return NGX_CHAIN_ERROR;
744             }
745
746             cl->buf->flush = 0;
747             cl->buf->last_buf = 0;
748
749             *ln = cl;
750
751             offset += rest;
752             size -= rest;
753         }
754
755         frame = ngx_http_spdy_filter_get_data_frame(stream, frame_size,
756                                                     out, cl);
757         if (frame == NULL) {
758             return NGX_CHAIN_ERROR;
759         }
760
761         ngx_http_spdy_queue_frame(sc, frame);
762
763         sc->send_window -= frame_size;
764
765         stream->send_window -= frame_size;
```

```
        stream->queued++;

        if (in == NULL) {
            break;
        }

        limit -= frame_size;

        if (limit == 0) {
            break;
        }

        if (limit < (off_t) slcf->chunk_size) {
            frame_size = (size_t) limit;
        }
    }

    if (offset) {
        cl = ngx_http_spdy_filter_get_shadow(stream, in->buf, offset, size);
        if (cl == NULL) {
            return NGX_CHAIN_ERROR;
        }

        in->buf = cl->buf;
        ngx_free_chain(r->pool, cl);
    }

    if (ngx_http_spdy_filter_send(fc, stream) == NGX_ERROR) {
        return NGX_CHAIN_ERROR;
    }

    if (in && ngx_http_spdy_flow_control(sc, stream) == NGX_DECLINED) {
        fc->write->delayed = 1;
    }

    return in;
}


static ngx_chain_t *
ngx_http_spdy_filter_get_shadow(ngx_http_spdy_stream_t *stream, ngx_buf_t *buf,
    off_t offset, off_t size)
{
    ngx_buf_t    *chunk;
    ngx_chain_t  *cl;

    cl = ngx_chain_get_free_buf(stream->request->pool, &stream->free_bufs);
    if (cl == NULL) {
        return NULL;
    }

    chunk = cl->buf;

    ngx_memcpy(chunk, buf, sizeof(ngx_buf_t));

    chunk->tag = (ngx_buf_tag_t) &ngx_http_spdy_filter_get_shadow;
    chunk->shadow = buf;

    if (ngx_buf_in_memory(chunk)) {
        chunk->pos += offset;
        chunk->last = chunk->pos + size;
    }

    if (chunk->in_file) {
        chunk->file_pos += offset;
        chunk->file_last = chunk->file_pos + size;
    }

    return cl;
}


static ngx_http_spdy_out_frame_t *
ngx_http_spdy_filter_get_data_frame(ngx_http_spdy_stream_t *stream,
    size_t len, ngx_chain_t *first, ngx_chain_t *last)
{
```

```
    u_char                    *p;
    ngx_buf_t                 *buf;
    ngx_uint_t                 flags;
    ngx_chain_t               *cl;
    ngx_http_spdy_out_frame_t  *frame;


    frame = stream->free_frames;

    if (frame) {
        stream->free_frames = frame->next;

    } else {
        frame = ngx_palloc(stream->request->pool,
                           sizeof(ngx_http_spdy_out_frame_t));
        if (frame == NULL) {
            return NULL;
        }
    }

    flags = last->buf->last_buf ? NGX_SPDY_FLAG_FIN : 0;

    ngx_log_debug4(NGX_LOG_DEBUG_HTTP, stream->request->connection->log, 0,
                   "spdy:%ui create DATA frame %p: len:%uz flags:%ui",
                   stream->id, frame, len, flags);

    cl = ngx_chain_get_free_buf(stream->request->pool,
                                &stream->free_data_headers);
    if (cl == NULL) {
        return NULL;
    }

    buf = cl->buf;

    if (buf->start) {
        p = buf->start;
        buf->pos = p;

        p += NGX_SPDY_SID_SIZE;

        (void) ngx_spdy_frame_write_flags_and_len(p, flags, len);

    } else {
        p = ngx_palloc(stream->request->pool, NGX_SPDY_FRAME_HEADER_SIZE);
        if (p == NULL) {
            return NULL;
        }

        buf->pos = p;
        buf->start = p;

        p = ngx_spdy_frame_write_sid(p, stream->id);
        p = ngx_spdy_frame_write_flags_and_len(p, flags, len);

        buf->last = p;
        buf->end = p;

        buf->tag = (ngx_buf_tag_t) &ngx_http_spdy_filter_get_data_frame;
        buf->memory = 1;
    }

    cl->next = first;
    first = cl;

    last->buf->flush = 1;

    frame->first = first;
    frame->last = last;
    frame->handler = ngx_http_spdy_data_frame_handler;
    frame->stream = stream;
    frame->length = len;
    frame->priority = stream->priority;
    frame->blocked = 0;
    frame->fin = last->buf->last_buf;

    return frame;
```

```
}


static ngx_inline ngx_int_t
ngx_http_spdy_filter_send(ngx_connection_t *fc, ngx_http_spdy_stream_t *stream)
{
    stream->blocked = 1;

    if (ngx_http_spdy_send_output_queue(stream->connection) == NGX_ERROR) {
        fc->error = 1;
        return NGX_ERROR;
    }

    stream->blocked = 0;

    if (stream->queued) {
        fc->buffered |= NGX_SPDY_BUFFERED;
        fc->write->delayed = 1;
        return NGX_AGAIN;
    }

    fc->buffered &= ~NGX_SPDY_BUFFERED;

    return NGX_OK;
}


static ngx_inline ngx_int_t
ngx_http_spdy_flow_control(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_stream_t *stream)
{
    if (stream->send_window <= 0) {
        stream->exhausted = 1;
        return NGX_DECLINED;
    }

    if (sc->send_window == 0) {
        ngx_http_spdy_waiting_queue(sc, stream);
        return NGX_DECLINED;
    }

    return NGX_OK;
}


static void
ngx_http_spdy_waiting_queue(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_stream_t *stream)
{
    ngx_queue_t             *q;
    ngx_http_spdy_stream_t  *s;

    if (stream->handled) {
        return;
    }

    stream->handled = 1;

    for (q = ngx_queue_last(&sc->waiting);
         q != ngx_queue_sentinel(&sc->waiting);
         q = ngx_queue_prev(q))
    {
        s = ngx_queue_data(q, ngx_http_spdy_stream_t, queue);

        /*
         * NB: higher values represent lower priorities.
         */
        if (stream->priority >= s->priority) {
            break;
        }
    }

    ngx_queue_insert_after(q, &stream->queue);
}


```

```
994  static ngx_int_t
995  ngx_http_spdy_syn_frame_handler(ngx_http_spdy_connection_t *sc,
996      ngx_http_spdy_out_frame_t *frame)
997  {
998      ngx_buf_t               *buf;
999      ngx_http_spdy_stream_t  *stream;
1000
1001     buf = frame->first->buf;
1002
1003     if (buf->pos != buf->last) {
1004         return NGX_AGAIN;
1005     }
1006
1007     stream = frame->stream;
1008
1009     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1010                    "spdy:%ui SYN_REPLY frame %p was sent", stream->id, frame);
1011
1012     ngx_free_chain(stream->request->pool, frame->first);
1013
1014     ngx_http_spdy_handle_frame(stream, frame);
1015
1016     ngx_http_spdy_handle_stream(sc, stream);
1017
1018     return NGX_OK;
1019 }
1020
1021
1022 static ngx_int_t
1023 ngx_http_spdy_data_frame_handler(ngx_http_spdy_connection_t *sc,
1024     ngx_http_spdy_out_frame_t *frame)
1025 {
1026     ngx_buf_t               *buf;
1027     ngx_chain_t             *cl, *ln;
1028     ngx_http_spdy_stream_t  *stream;
1029
1030     stream = frame->stream;
1031
1032     cl = frame->first;
1033
1034     if (cl->buf->tag == (ngx_buf_tag_t) &ngx_http_spdy_filter_get_data_frame) {
1035
1036         if (cl->buf->pos != cl->buf->last) {
1037             ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
1038                            "spdy:%ui DATA frame %p was sent partially",
1039                            stream->id, frame);
1040
1041             return NGX_AGAIN;
1042         }
1043
1044         ln = cl->next;
1045
1046         cl->next = stream->free_data_headers;
1047         stream->free_data_headers = cl;
1048
1049         if (cl == frame->last) {
1050             goto done;
1051         }
1052
1053         cl = ln;
1054     }
1055
1056     for ( ;; ) {
1057         if (cl->buf->tag == (ngx_buf_tag_t) &ngx_http_spdy_filter_get_shadow) {
1058             buf = cl->buf->shadow;
1059
1060             if (ngx_buf_in_memory(buf)) {
1061                 buf->pos = cl->buf->pos;
1062             }
1063
1064             if (buf->in_file) {
1065                 buf->file_pos = cl->buf->file_pos;
1066             }
1067         }
1068
1069         if (ngx_buf_size(cl->buf) != 0) {
```

```
            if (cl != frame->first) {
                frame->first = cl;
                ngx_http_spdy_handle_stream(sc, stream);
            }

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                           "spdy:%ui DATA frame %p was sent partially",
                           stream->id, frame);

            return NGX_AGAIN;
        }

        ln = cl->next;

        if (cl->buf->tag == (ngx_buf_tag_t) &ngx_http_spdy_filter_get_shadow) {
            cl->next = stream->free_bufs;
            stream->free_bufs = cl;

        } else {
            ngx_free_chain(stream->request->pool, cl);
        }

        if (cl == frame->last) {
            goto done;
        }

        cl = ln;
    }

done:

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, sc->connection->log, 0,
                   "spdy:%ui DATA frame %p was sent", stream->id, frame);

    stream->request->header_size += NGX_SPDY_FRAME_HEADER_SIZE;

    ngx_http_spdy_handle_frame(stream, frame);

    ngx_http_spdy_handle_stream(sc, stream);

    return NGX_OK;
}


static ngx_inline void
ngx_http_spdy_handle_frame(ngx_http_spdy_stream_t *stream,
    ngx_http_spdy_out_frame_t *frame)
{
    ngx_http_request_t  *r;

    r = stream->request;

    r->connection->sent += NGX_SPDY_FRAME_HEADER_SIZE + frame->length;

    if (frame->fin) {
        stream->out_closed = 1;
    }

    frame->next = stream->free_frames;
    stream->free_frames = frame;

    stream->queued--;
}


static ngx_inline void
ngx_http_spdy_handle_stream(ngx_http_spdy_connection_t *sc,
    ngx_http_spdy_stream_t *stream)
{
    ngx_event_t  *wev;

    if (stream->handled || stream->blocked || stream->exhausted) {
        return;
    }

```

```
1146      wev = stream->request->connection->write;
1147
1148      /*
1149       * This timer can only be set if the stream was delayed because of rate
1150       * limit.  In that case the event should be triggered by the timer.
1151       */
1152
1153      if (!wev->timer_set) {
1154          wev->delayed = 0;
1155
1156          stream->handled = 1;
1157          ngx_queue_insert_tail(&sc->posted, &stream->queue);
1158      }
1159  }
1160
1161
1162  static void
1163  ngx_http_spdy_filter_cleanup(void *data)
1164  {
1165      ngx_http_spdy_stream_t *stream = data;
1166
1167      size_t                      delta;
1168      ngx_http_spdy_out_frame_t   *frame, **fn;
1169      ngx_http_spdy_connection_t  *sc;
1170
1171      if (stream->handled) {
1172          stream->handled = 0;
1173          ngx_queue_remove(&stream->queue);
1174      }
1175
1176      if (stream->queued == 0) {
1177          return;
1178      }
1179
1180      delta = 0;
1181      sc = stream->connection;
1182      fn = &sc->last_out;
1183
1184      for ( ;; ) {
1185          frame = *fn;
1186
1187          if (frame == NULL) {
1188              break;
1189          }
1190
1191          if (frame->stream == stream && !frame->blocked) {
1192              *fn = frame->next;
1193
1194              delta += frame->length;
1195
1196              if (--stream->queued == 0) {
1197                  break;
1198              }
1199
1200              continue;
1201          }
1202
1203          fn = &frame->next;
1204      }
1205
1206      if (sc->send_window == 0 && delta && !ngx_queue_empty(&sc->waiting)) {
1207          ngx_queue_add(&sc->posted, &sc->waiting);
1208          ngx_queue_init(&sc->waiting);
1209      }
1210
1211      sc->send_window += delta;
1212  }
1213
1214
1215  static ngx_int_t
1216  ngx_http_spdy_filter_init(ngx_conf_t *cf)
1217  {
1218      ngx_http_next_header_filter = ngx_http_top_header_filter;
1219      ngx_http_top_header_filter = ngx_http_spdy_header_filter;
1220
1221      return NGX_OK;
```

```
1222 }
```

One Level Up Top Level

src/http/ngx_http_parse_time.c - nginx-1.7.10

Global variables defined

- mday

Functions defined

- ngx_http_parse_time

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static ngx_uint_t  mday[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
14
15  time_t
16  ngx_http_parse_time(u_char *value, size_t len)
17  {
18      u_char      *p, *end;
19      ngx_int_t    month;
20      ngx_uint_t   day, year, hour, min, sec;
21      uint64_t     time;
22      enum {
23          no = 0,
24          rfc822,   /* Tue, 10 Nov 2002 23:50:13   */
25          rfc850,   /* Tuesday, 10-Dec-02 23:50:13 */
26          isoc      /* Tue Dec 10 23:50:13 2002    */
27      } fmt;
28
29      fmt = 0;
30      end = value + len;
31
32  #if (NGX_SUPPRESS_WARN)
33      day = 32;
34      year = 2038;
35  #endif
36
37      for (p = value; p < end; p++) {
38          if (*p == ',') {
39              break;
40          }
41
42          if (*p == ' ') {
43              fmt = isoc;
44              break;
45          }
46      }
47
48      for (p++; p < end; p++)
49          if (*p != ' ') {
50              break;
51          }
52
53      if (end - p < 18) {
54          return NGX_ERROR;
55          }
56
57      if (fmt != isoc) {
```

```
        if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9') {
            return NGX_ERROR;
        }

        day = (*p - '0') * 10 + *(p + 1) - '0';
        p += 2;

        if (*p == ' ') {
            if (end - p < 18) {
                return NGX_ERROR;
            }
            fmt = rfc822;

        } else if (*p == '-') {
            fmt = rfc850;

        } else {
            return NGX_ERROR;
        }

        p++;
    }

    switch (*p) {

    case 'J':
        month = *(p + 1) == 'a' ? 0 : *(p + 2) == 'n' ? 5 : 6;
        break;

    case 'F':
        month = 1;
        break;

    case 'M':
        month = *(p + 2) == 'r' ? 2 : 4;
        break;

    case 'A':
        month = *(p + 1) == 'p' ? 3 : 7;
        break;

    case 'S':
         month = 8;
         break;

     case 'O':
         month = 9;
         break;

     case 'N':
         month = 10;
         break;

     case 'D':
         month = 11;
         break;

     default:
         return NGX_ERROR;
     }

     p += 3;

     if ((fmt == rfc822 && *p != ' ') || (fmt == rfc850 && *p != '-')) {
         return NGX_ERROR;
     }

     p++;

     if (fmt == rfc822) {
         if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9'
             || *(p + 2) < '0' || *(p + 2) > '9'
             || *(p + 3) < '0' || *(p + 3) > '9')
         {
             return NGX_ERROR;
         }
```

```
134
135         year = (*p - '0') * 1000 + (*(p + 1) - '0') * 100
136                + (*(p + 2) - '0') * 10 + *(p + 3) - '0';
137         p += 4;
138
139     } else if (fmt == rfc850) {
140         if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9') {
141             return NGX_ERROR;
142         }
143
144         year = (*p - '0') * 10 + *(p + 1) - '0';
145         year += (year < 70) ? 2000 : 1900;
146         p += 2;
147     }
148
149     if (fmt == isoc) {
150         if (*p == ' ') {
151             p++;
152         }
153
154         if (*p < '0' || *p > '9') {
155             return NGX_ERROR;
156         }
157
158         day = *p++ - '0';
159
160         if (*p != ' ') {
161             if (*p < '0' || *p > '9') {
162                 return NGX_ERROR;
163             }
164
165             day = day * 10 + *p++ - '0';
166         }
167
168         if (end - p < 14) {
169             return NGX_ERROR;
170         }
171     }
172
173     if (*p++ != ' ') {
174         return NGX_ERROR;
175     }
176
177     if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9') {
178         return NGX_ERROR;
179     }
180
181     hour = (*p - '0') * 10 + *(p + 1) - '0';
182     p += 2;
183
184     if (*p++ != ':') {
185         return NGX_ERROR;
186     }
187
188     if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9') {
189         return NGX_ERROR;
190     }
191
192     min = (*p - '0') * 10 + *(p + 1) - '0';
193     p += 2;
194
195     if (*p++ != ':') {
196         return NGX_ERROR;
197     }
198
199     if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9') {
200         return NGX_ERROR;
201     }
202
203     sec = (*p - '0') * 10 + *(p + 1) - '0';
204
205     if (fmt == isoc) {
206         p += 2;
207
208         if (*p++ != ' ') {
209             return NGX_ERROR;
```

```
        }

        if (*p < '0' || *p > '9' || *(p + 1) < '0' || *(p + 1) > '9'
            || *(p + 2) < '0' || *(p + 2) > '9'
            || *(p + 3) < '0' || *(p + 3) > '9')
        {
            return NGX_ERROR;
        }

        year = (*p - '0') * 1000 + (*(p + 1) - '0') * 100
               + (*(p + 2) - '0') * 10 + *(p + 3) - '0';
    }

    if (hour > 23 || min > 59 || sec > 59) {
         return NGX_ERROR;
    }

    if (day == 29 && month == 1) {
        if ((year & 3) || ((year % 100 == 0) && (year % 400) != 0)) {
            return NGX_ERROR;
        }

    } else if (day > mday[month]) {
        return NGX_ERROR;
    }

    /*
     * shift new year to March 1 and start months from 1 (not 0),
     * it is needed for Gauss' formula
     */

    if (--month <= 0) {
        month += 12;
        year -= 1;
    }

    /* Gauss' formula for Gregorian days since March 1, 1 BC */

    time = (uint64_t) (
            /* days in years including leap years since March 1, 1 BC */

            365 * year + year / 4 - year / 100 + year / 400

            /* days before the month */

            + 367 * month / 12 - 30

            /* days before the day */

            + day - 1

            /*
             * 719527 days were between March 1, 1 BC and March 1, 1970,
             * 31 and 28 days were in January and February 1970
             */

            - 719527 + 31 + 28) * 86400 + hour * 3600 + min * 60 + sec;

#if (NGX_TIME_T_SIZE <= 4)

    if (time > 0x7fffffff) {
        return NGX_ERROR;
    }

#endif

    return (time_t) time;
}
```

One Level Up Top Level

src/core/ngx_regex.c - nginx-1.7.10

Global variables defined

- ngx_pcre_pool
- ngx_pcre_studies
- ngx_regex_commands
- ngx_regex_module
- ngx_regex_module_ctx
- ngx_regex_pcre_jit_post

Data types defined

- ngx_regex_conf_t

Functions defined

- ngx_pcre_free_studies
- ngx_regex_compile
- ngx_regex_create_conf
- ngx_regex_exec_array
- ngx_regex_free
- ngx_regex_init
- ngx_regex_init_conf
- ngx_regex_malloc
- ngx_regex_malloc_done
- ngx_regex_malloc_init
- ngx_regex_module_init
- ngx_regex_pcre_jit

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  typedef struct {
13      ngx_flag_t  pcre_jit;
14  } ngx_regex_conf_t;
```

```
15
16
17  static void * ngx_libc_cdecl ngx_regex_malloc(size_t size);
18  static void ngx_libc_cdecl ngx_regex_free(void *p);
19  #if (NGX_HAVE_PCRE_JIT)
20  static void ngx_pcre_free_studies(void *data);
21  #endif
22
23  static ngx_int_t ngx_regex_module_init(ngx_cycle_t *cycle);
24
25  static void *ngx_regex_create_conf(ngx_cycle_t *cycle);
26  static char *ngx_regex_init_conf(ngx_cycle_t *cycle, void *conf);
27
28  static char *ngx_regex_pcre_jit(ngx_conf_t *cf, void *post, void *data);
29  static ngx_conf_post_t  ngx_regex_pcre_jit_post = { ngx_regex_pcre_jit };
30
31
32  static ngx_command_t  ngx_regex_commands[] = {
33
34      { ngx_string("pcre_jit"),
35        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
36        ngx_conf_set_flag_slot,
37        0,
38        offsetof(ngx_regex_conf_t, pcre_jit),
39        &ngx_regex_pcre_jit_post },
40
41        ngx_null_command
42  };
43
44
45  static ngx_core_module_t  ngx_regex_module_ctx = {
46      ngx_string("regex"),
47      ngx_regex_create_conf,
48      ngx_regex_init_conf
49  };
50
51
52  ngx_module_t  ngx_regex_module = {
53      NGX_MODULE_V1,
54      &ngx_regex_module_ctx,                 /* module context */
55      ngx_regex_commands,                    /* module directives */
56      NGX_CORE_MODULE,                       /* module type */
57      NULL,                                  /* init master */
58      ngx_regex_module_init,                 /* init module */
59      NULL,                                  /* init process */
60      NULL,                                  /* init thread */
61      NULL,                                  /* exit thread */
62      NULL,                                  /* exit process */
63      NULL,                                  /* exit master */
64      NGX_MODULE_V1_PADDING
65  };
66
67
68  static ngx_pool_t  *ngx_pcre_pool;
69  static ngx_list_t  *ngx_pcre_studies;
70
71
72  void
73  ngx_regex_init(void)
74  {
75      pcre_malloc = ngx_regex_malloc;
76      pcre_free = ngx_regex_free;
77  }
78
79
80  static ngx_inline void
81  ngx_regex_malloc_init(ngx_pool_t *pool)
82  {
83  #if (NGX_THREADS)
84      ngx_core_tls_t  *tls;
85
86      if (ngx_threaded) {
87          tls = ngx_thread_get_tls(ngx_core_tls_key);
88          tls->pool = pool;
89          return;
90      }
```

```
91
92  #endif
93
94      ngx_pcre_pool = pool;
95  }
96
97
98  static ngx_inline void
99  ngx_regex_malloc_done(void)
100  {
101  #if (NGX_THREADS)
102      ngx_core_tls_t  *tls;
103
104      if (ngx_threaded) {
105          tls = ngx_thread_get_tls(ngx_core_tls_key);
106          tls->pool = NULL;
107          return;
108      }
109
110  #endif
111
112      ngx_pcre_pool = NULL;
113  }
114
115
116  ngx_int_t
117  ngx_regex_compile(ngx_regex_compile_t *rc)
118  {
119      int               n, erroff;
120      char             *p;
121      pcre             *re;
122      const char       *errstr;
123      ngx_regex_elt_t  *elt;
124
125      ngx_regex_malloc_init(rc->pool);
126
127      re = pcre_compile((const char *) rc->pattern.data, (int) rc->options,
128                        &errstr, &erroff, NULL);
129
130      /* ensure that there is no current pool */
131      ngx_regex_malloc_done();
132
133      if (re == NULL) {
134          if ((size_t) erroff == rc->pattern.len) {
135              rc->err.len = ngx_snprintf(rc->err.data, rc->err.len,
136                                         "pcre_compile() failed: %s in \"%V\"",
137                                         errstr, &rc->pattern)
138                            - rc->err.data;
139
140          } else {
141              rc->err.len = ngx_snprintf(rc->err.data, rc->err.len,
142                                         "pcre_compile() failed: %s in \"%V\" at \"%s\"",
143                                         errstr, &rc->pattern, rc->pattern.data + erroff)
144                            - rc->err.data;
145          }
146
147          return NGX_ERROR;
148      }
149
150      rc->regex = ngx_pcalloc(rc->pool, sizeof(ngx_regex_t));
151      if (rc->regex == NULL) {
152          goto nomem;
153      }
154
155      rc->regex->code = re;
156
157      /* do not study at runtime */
158
159      if (ngx_pcre_studies != NULL) {
160          elt = ngx_list_push(ngx_pcre_studies);
161          if (elt == NULL) {
162              goto nomem;
163          }
164
165          elt->regex = rc->regex;
166          elt->name = rc->pattern.data;
```

```
    }

    n = pcre_fullinfo(re, NULL, PCRE_INFO_CAPTURECOUNT, &rc->captures);
    if (n < 0) {
        p = "pcre_fullinfo(\"%V\", PCRE_INFO_CAPTURECOUNT) failed: %d";
        goto failed;
    }

    if (rc->captures == 0) {
        return NGX_OK;
    }

    n = pcre_fullinfo(re, NULL, PCRE_INFO_NAMECOUNT, &rc->named_captures);
    if (n < 0) {
        p = "pcre_fullinfo(\"%V\", PCRE_INFO_NAMECOUNT) failed: %d";
        goto failed;
    }

    if (rc->named_captures == 0) {
        return NGX_OK;
    }

    n = pcre_fullinfo(re, NULL, PCRE_INFO_NAMEENTRYSIZE, &rc->name_size);
    if (n < 0) {
        p = "pcre_fullinfo(\"%V\", PCRE_INFO_NAMEENTRYSIZE) failed: %d";
        goto failed;
    }

    n = pcre_fullinfo(re, NULL, PCRE_INFO_NAMETABLE, &rc->names);
    if (n < 0) {
        p = "pcre_fullinfo(\"%V\", PCRE_INFO_NAMETABLE) failed: %d";
        goto failed;
    }

    return NGX_OK;

failed:

    rc->err.len = ngx_snprintf(rc->err.data, rc->err.len, p, &rc->pattern, n)
                  - rc->err.data;
    return NGX_ERROR;

nomem:

    rc->err.len = ngx_snprintf(rc->err.data, rc->err.len,
                               "regex \"%V\" compilation failed: no memory",
                               &rc->pattern)
                  - rc->err.data;
    return NGX_ERROR;
}


ngx_int_t
ngx_regex_exec_array(ngx_array_t *a, ngx_str_t *s, ngx_log_t *log)
{
    ngx_int_t         n;
    ngx_uint_t        i;
    ngx_regex_elt_t  *re;

    re = a->elts;

    for (i = 0; i < a->nelts; i++) {

        n = ngx_regex_exec(re[i].regex, s, NULL, 0);

        if (n == NGX_REGEX_NO_MATCHED) {
            continue;
        }

        if (n < 0) {
            ngx_log_error(NGX_LOG_ALERT, log, 0,
                          ngx_regex_exec_n " failed: %i on \"%V\" using \"%s\"",
                          n, s, re[i].name);
            return NGX_ERROR;
        }

```

```
        /* match */

        return NGX_OK;
    }

    return NGX_DECLINED;
}


static void * ngx_libc_cdecl
ngx_regex_malloc(size_t size)
{
    ngx_pool_t      *pool;
#if (NGX_THREADS)
    ngx_core_tls_t  *tls;

    if (ngx_threaded) {
        tls = ngx_thread_get_tls(ngx_core_tls_key);
        pool = tls->pool;

    } else {
        pool = ngx_pcre_pool;
    }

#else

    pool = ngx_pcre_pool;

#endif

    if (pool) {
        return ngx_palloc(pool, size);
    }

    return NULL;
}


static void ngx_libc_cdecl
ngx_regex_free(void *p)
{
    return;
}


#if (NGX_HAVE_PCRE_JIT)

static void
ngx_pcre_free_studies(void *data)
{
    ngx_list_t *studies = data;

    ngx_uint_t        i;
    ngx_list_part_t  *part;
    ngx_regex_elt_t  *elts;

    part = &studies->part;
    elts = part->elts;

    for (i = 0 ; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            elts = part->elts;
            i = 0;
        }

        if (elts[i].regex->extra != NULL) {
            pcre_free_study(elts[i].regex->extra);
        }
    }
}
```

```
#endif


static ngx_int_t
ngx_regex_module_init(ngx_cycle_t *cycle)
{
    int               opt;
    const char       *errstr;
    ngx_uint_t        i;
    ngx_list_part_t  *part;
    ngx_regex_elt_t  *elts;

    opt = 0;

#if (NGX_HAVE_PCRE_JIT)
    {
    ngx_regex_conf_t    *rcf;
    ngx_pool_cleanup_t  *cln;

    rcf = (ngx_regex_conf_t *) ngx_get_conf(cycle->conf_ctx, ngx_regex_module);

    if (rcf->pcre_jit) {
        opt = PCRE_STUDY_JIT_COMPILE;

        /*
         * The PCRE JIT compiler uses mmap for its executable codes, so we
         * have to explicitly call the pcre_free_study() function to free
         * this memory.
         */

        cln = ngx_pool_cleanup_add(cycle->pool, 0);
        if (cln == NULL) {
            return NGX_ERROR;
        }

        cln->handler = ngx_pcre_free_studies;
        cln->data = ngx_pcre_studies;
    }
    }
#endif

    ngx_regex_malloc_init(cycle->pool);

    part = &ngx_pcre_studies->part;
    elts = part->elts;

    for (i = 0 ; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            elts = part->elts;
            i = 0;
        }

        elts[i].regex->extra = pcre_study(elts[i].regex->code, opt, &errstr);

        if (errstr != NULL) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                          "pcre_study() failed: %s in \"%s\"",
                          errstr, elts[i].name);
        }

#if (NGX_HAVE_PCRE_JIT)
        if (opt & PCRE_STUDY_JIT_COMPILE) {
            int jit, n;

            jit = 0;
            n = pcre_fullinfo(elts[i].regex->code, elts[i].regex->extra,
                              PCRE_INFO_JIT, &jit);

            if (n != 0 || jit != 1) {
```

```
                ngx_log_error(NGX_LOG_INFO, cycle->log, 0,
                              "JIT compiler does not support pattern: \"%s\"",
                              elts[i].name);
            }
        }
#endif
    }

    ngx_regex_malloc_done();

    ngx_pcre_studies = NULL;

    return NGX_OK;
}


static void *
ngx_regex_create_conf(ngx_cycle_t *cycle)
{
    ngx_regex_conf_t  *rcf;

    rcf = ngx_pcalloc(cycle->pool, sizeof(ngx_regex_conf_t));
    if (rcf == NULL) {
        return NULL;
    }

    rcf->pcre_jit = NGX_CONF_UNSET;

    ngx_pcre_studies = ngx_list_create(cycle->pool, 8, sizeof(ngx_regex_elt_t));
    if (ngx_pcre_studies == NULL) {
        return NULL;
    }

    return rcf;
}


static char *
ngx_regex_init_conf(ngx_cycle_t *cycle, void *conf)
{
    ngx_regex_conf_t *rcf = conf;

    ngx_conf_init_value(rcf->pcre_jit, 0);

    return NGX_CONF_OK;
}


static char *
ngx_regex_pcre_jit(ngx_conf_t *cf, void *post, void *data)
{
    ngx_flag_t  *fp = data;

    if (*fp == 0) {
        return NGX_CONF_OK;
    }

#if (NGX_HAVE_PCRE_JIT)
    {
    int  jit, r;

    jit = 0;
    r = pcre_config(PCRE_CONFIG_JIT, &jit);

    if (r != 0 || jit != 1) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "PCRE library does not support JIT");
        *fp = 0;
    }
    }
#else
    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                       "nginx was built without PCRE JIT support");
    *fp = 0;
#endif

```

```
471     return NGX_CONF_OK;
472 }
```

One Level Up Top Level

src/core/ngx_regex.h - nginx-1.7.10

Data types defined

- ngx_regex_compile_t
- ngx_regex_elt_t
- ngx_regex_t

Macros defined

- NGX_REGEX_CASELESS
- NGX_REGEX_NO_MATCHED
- _NGX_REGEX_H_INCLUDED_
- ngx_regex_exec
- ngx_regex_exec_n

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_REGEX_H_INCLUDED_
9  #define _NGX_REGEX_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15  #include <pcre.h>
16
17
18  #define NGX_REGEX_NO_MATCHED  PCRE_ERROR_NOMATCH   /* -1 */
19
20  #define NGX_REGEX_CASELESS    PCRE_CASELESS
21
22
23  typedef struct {
24      pcre        *code;
25      pcre_extra  *extra;
26  } ngx_regex_t;
27
28
29  typedef struct {
30      ngx_str_t     pattern;
31      ngx_pool_t   *pool;
32      ngx_int_t     options;
33
34      ngx_regex_t  *regex;
35      int           captures;
36      int           named_captures;
37      int           name_size;
38      u_char       *names;
39      ngx_str_t     err;
40  } ngx_regex_compile_t;
41
42
43  typedef struct {
```

```
    ngx_regex_t  *regex;
    u_char       *name;
} ngx_regex_elt_t;


void ngx_regex_init(void);
ngx_int_t ngx_regex_compile(ngx_regex_compile_t *rc);

#define ngx_regex_exec(re, s, captures, size)                                \
    pcre_exec(re->code, re->extra, (const char *) (s)->data, (s)->len, 0, 0, \
              captures, size)
#define ngx_regex_exec_n      "pcre_exec()"

ngx_int_t ngx_regex_exec_array(ngx_array_t *a, ngx_str_t *s, ngx_log_t *log);


#endif /* _NGX_REGEX_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_proxy_protocol.c - nginx-1.7.10

Functions defined

- ngx_proxy_protocol_parse

Source code

```
1
2  /*
3  * Copyright (C) Roman Arutyunyan
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  u_char *
13  ngx_proxy_protocol_parse(ngx_connection_t *c, u_char *buf, u_char *last)
14  {
15      size_t  len;
16      u_char  ch, *p, *addr;
17
18      p = buf;
19      len = last - buf;
20
21      if (len < 8 || ngx_strncmp(p, "PROXY ", 6) != 0) {
22          goto invalid;
23      }
24
25      p += 6;
26      len -= 6;
27
28      if (len >= 7 && ngx_strncmp(p, "UNKNOWN", 7) == 0) {
29          ngx_log_debug0(NGX_LOG_DEBUG_CORE, c->log, 0,
30                         "PROXY protocol unknown protocol");
31          p += 7;
32          goto skip;
33      }
34
35      if (len < 5 || ngx_strncmp(p, "TCP", 3) != 0
36          || (p[3] != '4' && p[3] != '6') || p[4] != ' ')
37      {
38          goto invalid;
39      }
40
41      p += 5;
42      addr = p;
43
44      for ( ;; ) {
45          if (p == last) {
46              goto invalid;
47          }
48
49          ch = *p++;
50
51          if (ch == ' ') {
52              break;
53          }
54
55          if (ch != ':' && ch != '.'
56              && (ch < 'a' || ch > 'f')
57              && (ch < 'A' || ch > 'F')
58              && (ch < '0' || ch > '9'))
59          {
60              goto invalid;
61          }
62      }
```

```
63
64      len = p - addr - 1;
65      c->proxy_protocol_addr.data = ngx_pnalloc(c->pool, len);
66
67      if (c->proxy_protocol_addr.data == NULL) {
68          return NULL;
69      }
70
71      ngx_memcpy(c->proxy_protocol_addr.data, addr, len);
72      c->proxy_protocol_addr.len = len;
73
74      ngx_log_debug1(NGX_LOG_DEBUG_CORE, c->log, 0,
75                     "PROXY protocol address: \"%V\"", &c->proxy_protocol_addr);
76
77  skip:
78
79      for ( /* void */ ; p < last - 1; p++) {
80          if (p[0] == CR && p[1] == LF) {
81              return p + 2;
82          }
83      }
84
85  invalid:
86
87      ngx_log_error(NGX_LOG_ERR, c->log, 0,
88                    "broken header: \"%*s\"", (size_t) (last - buf), buf);
89
90      return NULL;
91  }
```

One Level Up Top Level

src/core/ngx_proxy_protocol.h - nginx-1.7.10

Macros defined

- NGX_PROXY_PROTOCOL_MAX_HEADER
- _NGX_PROXY_PROTOCOL_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Roman Arutyunyan
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_PROXY_PROTOCOL_H_INCLUDED_
9  #define _NGX_PROXY_PROTOCOL_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #define NGX_PROXY_PROTOCOL_MAX_HEADER  107
17
18
19  u_char *ngx_proxy_protocol_parse(ngx_connection_t *c, u_char *buf,
20      u_char *last);
21
22
23  #endif /* _NGX_PROXY_PROTOCOL_H_INCLUDED_ */
```

src/http/ngx_http_special_response.c - nginx-1.7.10

Global variables defined

- ngx_http_error_301_page
- ngx_http_error_302_page
- ngx_http_error_303_page
- ngx_http_error_307_page
- ngx_http_error_400_page
- ngx_http_error_401_page
- ngx_http_error_402_page
- ngx_http_error_403_page
- ngx_http_error_404_page
- ngx_http_error_405_page
- ngx_http_error_406_page
- ngx_http_error_408_page
- ngx_http_error_409_page
- ngx_http_error_410_page
- ngx_http_error_411_page
- ngx_http_error_412_page
- ngx_http_error_413_page
- ngx_http_error_414_page
- ngx_http_error_415_page
- ngx_http_error_416_page
- ngx_http_error_494_page
- ngx_http_error_495_page
- ngx_http_error_496_page
- ngx_http_error_497_page
- ngx_http_error_500_page
- ngx_http_error_501_page
- ngx_http_error_502_page
- ngx_http_error_503_page
- ngx_http_error_504_page
- ngx_http_error_507_page

- ngx_http_error_full_tail
- ngx_http_error_pages
- ngx_http_error_tail
- ngx_http_get_name
- ngx_http_msie_padding
- ngx_http_msie_refresh_head
- ngx_http_msie_refresh_tail

Functions defined

- ngx_http_clean_header
- ngx_http_filter_finalize_request
- ngx_http_send_error_page
- ngx_http_send_refresh
- ngx_http_send_special_response
- ngx_http_special_response_handler

Macros defined

- NGX_HTTP_LAST_2XX
- NGX_HTTP_LAST_3XX
- NGX_HTTP_LAST_4XX
- NGX_HTTP_LAST_5XX
- NGX_HTTP_OFF_3XX
- NGX_HTTP_OFF_4XX
- NGX_HTTP_OFF_5XX

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <nginx.h>
12
13
14  static ngx_int_t ngx_http_send_error_page(ngx_http_request_t *r,
15      ngx_http_err_page_t *err_page);
16  static ngx_int_t ngx_http_send_special_response(ngx_http_request_t *r,
17      ngx_http_core_loc_conf_t *clcf, ngx_uint_t err);
18  static ngx_int_t ngx_http_send_refresh(ngx_http_request_t *r);
19
20
```

```
static u_char ngx_http_error_full_tail[] =
"<hr><center>" NGINX_VER "</center>" CRLF
"</body>" CRLF
"</html>" CRLF
;


static u_char ngx_http_error_tail[] =
"<hr><center>nginx</center>" CRLF
"</body>" CRLF
"</html>" CRLF
;


static u_char ngx_http_msie_padding[] =
"<!-- a padding to disable MSIE and Chrome friendly error page -->" CRLF
"<!-- a padding to disable MSIE and Chrome friendly error page -->" CRLF
"<!-- a padding to disable MSIE and Chrome friendly error page -->" CRLF
"<!-- a padding to disable MSIE and Chrome friendly error page -->" CRLF
"<!-- a padding to disable MSIE and Chrome friendly error page -->" CRLF
"<!-- a padding to disable MSIE and Chrome friendly error page -->" CRLF
;


static u_char ngx_http_msie_refresh_head[] =
"<html><head><meta http-equiv=\"Refresh\" content=\"0; URL=";


static u_char ngx_http_msie_refresh_tail[] =
"\"></head><body></body></html>" CRLF;


static char ngx_http_error_301_page[] =
"<html>" CRLF
"<head><title>301 Moved Permanently</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>301 Moved Permanently</h1></center>" CRLF
;


static char ngx_http_error_302_page[] =
"<html>" CRLF
"<head><title>302 Found</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>302 Found</h1></center>" CRLF
;


static char ngx_http_error_303_page[] =
"<html>" CRLF
"<head><title>303 See Other</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>303 See Other</h1></center>" CRLF
;


static char ngx_http_error_307_page[] =
"<html>" CRLF
"<head><title>307 Temporary Redirect</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>307 Temporary Redirect</h1></center>" CRLF
;


static char ngx_http_error_400_page[] =
"<html>" CRLF
"<head><title>400 Bad Request</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>400 Bad Request</h1></center>" CRLF
;


static char ngx_http_error_401_page[] =
"<html>" CRLF
"<head><title>401 Authorization Required</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
```

```
"<center><h1>401 Authorization Required</h1></center>" CRLF
;


static char ngx_http_error_402_page[] =
"<html>" CRLF
"<head><title>402 Payment Required</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>402 Payment Required</h1></center>" CRLF
;


static char ngx_http_error_403_page[] =
"<html>" CRLF
"<head><title>403 Forbidden</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>403 Forbidden</h1></center>" CRLF
;


static char ngx_http_error_404_page[] =
"<html>" CRLF
"<head><title>404 Not Found</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>404 Not Found</h1></center>" CRLF
;


static char ngx_http_error_405_page[] =
"<html>" CRLF
"<head><title>405 Not Allowed</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>405 Not Allowed</h1></center>" CRLF
;


static char ngx_http_error_406_page[] =
"<html>" CRLF
"<head><title>406 Not Acceptable</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>406 Not Acceptable</h1></center>" CRLF
;


static char ngx_http_error_408_page[] =
"<html>" CRLF
"<head><title>408 Request Time-out</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>408 Request Time-out</h1></center>" CRLF
;


static char ngx_http_error_409_page[] =
"<html>" CRLF
"<head><title>409 Conflict</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>409 Conflict</h1></center>" CRLF
;


static char ngx_http_error_410_page[] =
"<html>" CRLF
"<head><title>410 Gone</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>410 Gone</h1></center>" CRLF
;


static char ngx_http_error_411_page[] =
"<html>" CRLF
"<head><title>411 Length Required</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>411 Length Required</h1></center>" CRLF
;


```

```
static char ngx_http_error_412_page[] =
"<html>" CRLF
"<head><title>412 Precondition Failed</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>412 Precondition Failed</h1></center>" CRLF
;


static char ngx_http_error_413_page[] =
"<html>" CRLF
"<head><title>413 Request Entity Too Large</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>413 Request Entity Too Large</h1></center>" CRLF
;


static char ngx_http_error_414_page[] =
"<html>" CRLF
"<head><title>414 Request-URI Too Large</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>414 Request-URI Too Large</h1></center>" CRLF
;


static char ngx_http_error_415_page[] =
"<html>" CRLF
"<head><title>415 Unsupported Media Type</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>415 Unsupported Media Type</h1></center>" CRLF
;


static char ngx_http_error_416_page[] =
"<html>" CRLF
"<head><title>416 Requested Range Not Satisfiable</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>416 Requested Range Not Satisfiable</h1></center>" CRLF
;


static char ngx_http_error_494_page[] =
"<html>" CRLF
"<head><title>400 Request Header Or Cookie Too Large</title></head>"
CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>400 Bad Request</h1></center>" CRLF
"<center>Request Header Or Cookie Too Large</center>" CRLF
;


static char ngx_http_error_495_page[] =
"<html>" CRLF
"<head><title>400 The SSL certificate error</title></head>"
CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>400 Bad Request</h1></center>" CRLF
"<center>The SSL certificate error</center>" CRLF
;


static char ngx_http_error_496_page[] =
"<html>" CRLF
"<head><title>400 No required SSL certificate was sent</title></head>"
CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>400 Bad Request</h1></center>" CRLF
"<center>No required SSL certificate was sent</center>" CRLF
;


static char ngx_http_error_497_page[] =
"<html>" CRLF
"<head><title>400 The plain HTTP request was sent to HTTPS port</title></head>"
CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>400 Bad Request</h1></center>" CRLF
```

```
"<center>The plain HTTP request was sent to HTTPS port</center>" CRLF
;


static char ngx_http_error_500_page[] =
"<html>" CRLF
"<head><title>500 Internal Server Error</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>500 Internal Server Error</h1></center>" CRLF
;


static char ngx_http_error_501_page[] =
"<html>" CRLF
"<head><title>501 Not Implemented</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>501 Not Implemented</h1></center>" CRLF
;


static char ngx_http_error_502_page[] =
"<html>" CRLF
"<head><title>502 Bad Gateway</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>502 Bad Gateway</h1></center>" CRLF
;


static char ngx_http_error_503_page[] =
"<html>" CRLF
"<head><title>503 Service Temporarily Unavailable</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>503 Service Temporarily Unavailable</h1></center>" CRLF
;


static char ngx_http_error_504_page[] =
"<html>" CRLF
"<head><title>504 Gateway Time-out</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>504 Gateway Time-out</h1></center>" CRLF
;


static char ngx_http_error_507_page[] =
"<html>" CRLF
"<head><title>507 Insufficient Storage</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>507 Insufficient Storage</h1></center>" CRLF
;


static ngx_str_t ngx_http_error_pages[] = {

    ngx_null_string,                     /* 201, 204 */

#define NGX_HTTP_LAST_2XX  202
#define NGX_HTTP_OFF_3XX   (NGX_HTTP_LAST_2XX - 201)

    /* ngx_null_string, */               /* 300 */
    ngx_string(ngx_http_error_301_page),
    ngx_string(ngx_http_error_302_page),
    ngx_string(ngx_http_error_303_page),
    ngx_null_string,                     /* 304 */
    ngx_null_string,                     /* 305 */
    ngx_null_string,                     /* 306 */
    ngx_string(ngx_http_error_307_page),

#define NGX_HTTP_LAST_3XX  308
#define NGX_HTTP_OFF_4XX   (NGX_HTTP_LAST_3XX - 301 + NGX_HTTP_OFF_3XX)

    ngx_string(ngx_http_error_400_page),
    ngx_string(ngx_http_error_401_page),
    ngx_string(ngx_http_error_402_page),
    ngx_string(ngx_http_error_403_page),
    ngx_string(ngx_http_error_404_page),
```

```
    ngx_string(ngx_http_error_405_page),
    ngx_string(ngx_http_error_406_page),
    ngx_null_string,                     /* 407 */
    ngx_string(ngx_http_error_408_page),
    ngx_string(ngx_http_error_409_page),
    ngx_string(ngx_http_error_410_page),
    ngx_string(ngx_http_error_411_page),
    ngx_string(ngx_http_error_412_page),
    ngx_string(ngx_http_error_413_page),
    ngx_string(ngx_http_error_414_page),
    ngx_string(ngx_http_error_415_page),
    ngx_string(ngx_http_error_416_page),

#define NGX_HTTP_LAST_4XX  417
#define NGX_HTTP_OFF_5XX   (NGX_HTTP_LAST_4XX - 400 + NGX_HTTP_OFF_4XX)

    ngx_string(ngx_http_error_494_page), /* 494, request header too large */
    ngx_string(ngx_http_error_495_page), /* 495, https certificate error */
    ngx_string(ngx_http_error_496_page), /* 496, https no certificate */
    ngx_string(ngx_http_error_497_page), /* 497, http to https */
    ngx_string(ngx_http_error_404_page), /* 498, canceled */
    ngx_null_string,                     /* 499, client has closed connection */

    ngx_string(ngx_http_error_500_page),
    ngx_string(ngx_http_error_501_page),
    ngx_string(ngx_http_error_502_page),
    ngx_string(ngx_http_error_503_page),
    ngx_string(ngx_http_error_504_page),
    ngx_null_string,                     /* 505 */
    ngx_null_string,                     /* 506 */
    ngx_string(ngx_http_error_507_page)

#define NGX_HTTP_LAST_5XX  508

};


static ngx_str_t  ngx_http_get_name = { 3, (u_char *) "GET " };


ngx_int_t
ngx_http_special_response_handler(ngx_http_request_t *r, ngx_int_t error)
{
    ngx_uint_t                 i, err;
    ngx_http_err_page_t       *err_page;
    ngx_http_core_loc_conf_t  *clcf;

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http special response: %i, \"%V?%V\"",
                   error, &r->uri, &r->args);

    r->err_status = error;

    if (r->keepalive) {
        switch (error) {
            case NGX_HTTP_BAD_REQUEST:
            case NGX_HTTP_REQUEST_ENTITY_TOO_LARGE:
            case NGX_HTTP_REQUEST_URI_TOO_LARGE:
            case NGX_HTTP_TO_HTTPS:
            case NGX_HTTPS_CERT_ERROR:
            case NGX_HTTPS_NO_CERT:
            case NGX_HTTP_INTERNAL_SERVER_ERROR:
            case NGX_HTTP_NOT_IMPLEMENTED:
                r->keepalive = 0;
        }
    }

    if (r->lingering_close) {
        switch (error) {
            case NGX_HTTP_BAD_REQUEST:
            case NGX_HTTP_TO_HTTPS:
            case NGX_HTTPS_CERT_ERROR:
            case NGX_HTTPS_NO_CERT:
                r->lingering_close = 0;
        }
    }
```

```
401
402     r->headers_out.content_type.len = 0;
403
404     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
405
406     if (!r->error_page && clcf->error_pages && r->uri_changes != 0) {
407
408         if (clcf->recursive_error_pages == 0) {
409             r->error_page = 1;
410         }
411
412         err_page = clcf->error_pages->elts;
413
414         for (i = 0; i < clcf->error_pages->nelts; i++) {
415             if (err_page[i].status == error) {
416                 return ngx_http_send_error_page(r, &err_page[i]);
417             }
418         }
419     }
420
421     r->expect_tested = 1;
422
423     if (ngx_http_discard_request_body(r) != NGX_OK) {
424         r->keepalive = 0;
425     }
426
427     if (clcf->msie_refresh
428         && r->headers_in.msie
429         && (error == NGX_HTTP_MOVED_PERMANENTLY
430             || error == NGX_HTTP_MOVED_TEMPORARILY))
431     {
432         return ngx_http_send_refresh(r);
433     }
434
435     if (error == NGX_HTTP_CREATED) {
436         /* 201 */
437         err = 0;
438
439     } else if (error == NGX_HTTP_NO_CONTENT) {
440         /* 204 */
441         err = 0;
442
443     } else if (error >= NGX_HTTP_MOVED_PERMANENTLY
444                && error < NGX_HTTP_LAST_3XX)
445     {
446         /* 3XX */
447         err = error - NGX_HTTP_MOVED_PERMANENTLY + NGX_HTTP_OFF_3XX;
448
449     } else if (error >= NGX_HTTP_BAD_REQUEST
450                && error < NGX_HTTP_LAST_4XX)
451     {
452         /* 4XX */
453         err = error - NGX_HTTP_BAD_REQUEST + NGX_HTTP_OFF_4XX;
454
455     } else if (error >= NGX_HTTP_NGINX_CODES
456                && error < NGX_HTTP_LAST_5XX)
457     {
458         /* 49X, 5XX */
459         err = error - NGX_HTTP_NGINX_CODES + NGX_HTTP_OFF_5XX;
460         switch (error) {
461             case NGX_HTTP_TO_HTTPS:
462             case NGX_HTTPS_CERT_ERROR:
463             case NGX_HTTPS_NO_CERT:
464             case NGX_HTTP_REQUEST_HEADER_TOO_LARGE:
465                 r->err_status = NGX_HTTP_BAD_REQUEST;
466                 break;
467         }
468
469     } else {
470         /* unknown code, zero body */
471         err = 0;
472     }
473
474     return ngx_http_send_special_response(r, clcf, err);
475 }
476
```

```
ngx_int_t
ngx_http_filter_finalize_request(ngx_http_request_t *r, ngx_module_t *m,
    ngx_int_t error)
{
    void       *ctx;
    ngx_int_t   rc;

    ngx_http_clean_header(r);

    ctx = NULL;

    if (m) {
        ctx = r->ctx[m->ctx_index];
    }

    /* clear the modules contexts */
    ngx_memzero(r->ctx, sizeof(void *) * ngx_http_max_module);

    if (m) {
        r->ctx[m->ctx_index] = ctx;
    }

    r->filter_finalize = 1;

    rc = ngx_http_special_response_handler(r, error);

    /* NGX_ERROR resets any pending data */

    switch (rc) {

    case NGX_OK:
    case NGX_DONE:
        return NGX_ERROR;

    default:
        return rc;
    }
}


void
ngx_http_clean_header(ngx_http_request_t *r)
{
    ngx_memzero(&r->headers_out.status,
                sizeof(ngx_http_headers_out_t)
                    - offsetof(ngx_http_headers_out_t, status));

    r->headers_out.headers.part.nelts = 0;
    r->headers_out.headers.part.next = NULL;
    r->headers_out.headers.last = &r->headers_out.headers.part;

    r->headers_out.content_length_n = -1;
    r->headers_out.last_modified_time = -1;
}


static ngx_int_t
ngx_http_send_error_page(ngx_http_request_t *r, ngx_http_err_page_t *err_page)
{
    ngx_int_t                  overwrite;
    ngx_str_t                  uri, args;
    ngx_table_elt_t           *location;
    ngx_http_core_loc_conf_t  *clcf;

    overwrite = err_page->overwrite;

    if (overwrite && overwrite != NGX_HTTP_OK) {
        r->expect_tested = 1;
    }

    if (overwrite >= 0) {
        r->err_status = overwrite;
    }

    if (ngx_http_complex_value(r, &err_page->value, &uri) != NGX_OK) {
```

```
553         return NGX_ERROR;
554     }
555 
556     if (uri.data[0] == '/') {
557 
558         if (err_page->value.lengths) {
559             ngx_http_split_args(r, &uri, &args);
560 
561         } else {
562             args = err_page->args;
563         }
564 
565         if (r->method != NGX_HTTP_HEAD) {
566             r->method = NGX_HTTP_GET;
567             r->method_name = ngx_http_get_name;
568         }
569 
570         return ngx_http_internal_redirect(r, &uri, &args);
571     }
572 
573     if (uri.data[0] == '@') {
574         return ngx_http_named_location(r, &uri);
575     }
576 
577     location = ngx_list_push(&r->headers_out.headers);
578 
579     if (location == NULL) {
580         return NGX_ERROR;
581     }
582 
583     if (overwrite != NGX_HTTP_MOVED_PERMANENTLY
584         && overwrite != NGX_HTTP_MOVED_TEMPORARILY
585         && overwrite != NGX_HTTP_SEE_OTHER
586         && overwrite != NGX_HTTP_TEMPORARY_REDIRECT)
587     {
588         r->err_status = NGX_HTTP_MOVED_TEMPORARILY;
589     }
590 
591     location->hash = 1;
592     ngx_str_set(&location->key, "Location");
593     location->value = uri;
594 
595     ngx_http_clear_location(r);
596 
597     r->headers_out.location = location;
598 
599     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
600 
601     if (clcf->msie_refresh && r->headers_in.msie) {
602         return ngx_http_send_refresh(r);
603     }
604 
605     return ngx_http_send_special_response(r, clcf, r->err_status
606                                                    - NGX_HTTP_MOVED_PERMANENTLY
607                                                    + NGX_HTTP_OFF_3XX);
608 }
609 
610 
611 static ngx_int_t
612 ngx_http_send_special_response(ngx_http_request_t *r,
613     ngx_http_core_loc_conf_t *clcf, ngx_uint_t err)
614 {
615     u_char       *tail;
616     size_t        len;
617     ngx_int_t     rc;
618     ngx_buf_t    *b;
619     ngx_uint_t    msie_padding;
620     ngx_chain_t   out[3];
621 
622     if (clcf->server_tokens) {
623         len = sizeof(ngx_http_error_full_tail) - 1;
624         tail = ngx_http_error_full_tail;
625 
626     } else {
627         len = sizeof(ngx_http_error_tail) - 1;
628         tail = ngx_http_error_tail;
```

```
    }

    msie_padding = 0;

    if (ngx_http_error_pages[err].len) {
        r->headers_out.content_length_n = ngx_http_error_pages[err].len + len;
        if (clcf->msie_padding
            && (r->headers_in.msie || r->headers_in.chrome)
            && r->http_version >= NGX_HTTP_VERSION_10
            && err >= NGX_HTTP_OFF_4XX)
        {
            r->headers_out.content_length_n +=
                                        sizeof(ngx_http_msie_padding) - 1;
            msie_padding = 1;
        }

        r->headers_out.content_type_len = sizeof("text/html") - 1;
        ngx_str_set(&r->headers_out.content_type, "text/html");
        r->headers_out.content_type_lowcase = NULL;

    } else {
        r->headers_out.content_length_n = 0;
    }

    if (r->headers_out.content_length) {
        r->headers_out.content_length->hash = 0;
        r->headers_out.content_length = NULL;
    }

    ngx_http_clear_accept_ranges(r);
    ngx_http_clear_last_modified(r);
    ngx_http_clear_etag(r);

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || r->header_only) {
        return rc;
    }

    if (ngx_http_error_pages[err].len == 0) {
        return ngx_http_send_special(r, NGX_HTTP_LAST);
    }

    b = ngx_calloc_buf(r->pool);
    if (b == NULL) {
        return NGX_ERROR;
    }

    b->memory = 1;
    b->pos = ngx_http_error_pages[err].data;
    b->last = ngx_http_error_pages[err].data + ngx_http_error_pages[err].len;

    out[0].buf = b;
    out[0].next = &out[1];

    b = ngx_calloc_buf(r->pool);
    if (b == NULL) {
        return NGX_ERROR;
    }

    b->memory = 1;

    b->pos = tail;
    b->last = tail + len;

    out[1].buf = b;
    out[1].next = NULL;

    if (msie_padding) {
        b = ngx_calloc_buf(r->pool);
        if (b == NULL) {
            return NGX_ERROR;
        }

        b->memory = 1;
        b->pos = ngx_http_msie_padding;
```

```
        b->last = ngx_http_msie_padding + sizeof(ngx_http_msie_padding) - 1;

        out[1].next = &out[2];
        out[2].buf = b;
        out[2].next = NULL;
    }

    if (r == r->main) {
        b->last_buf = 1;
    }

    b->last_in_chain = 1;

    return ngx_http_output_filter(r, &out[0]);
}


static ngx_int_t
ngx_http_send_refresh(ngx_http_request_t *r)
{
    u_char       *p, *location;
    size_t        len, size;
    uintptr_t     escape;
    ngx_int_t     rc;
    ngx_buf_t    *b;
    ngx_chain_t   out;

    len = r->headers_out.location->value.len;
    location = r->headers_out.location->value.data;

    escape = 2 * ngx_escape_uri(NULL, location, len, NGX_ESCAPE_REFRESH);

    size = sizeof(ngx_http_msie_refresh_head) - 1
           + escape + len
           + sizeof(ngx_http_msie_refresh_tail) - 1;

    r->err_status = NGX_HTTP_OK;

    r->headers_out.content_type_len = sizeof("text/html") - 1;
    ngx_str_set(&r->headers_out.content_type, "text/html");
    r->headers_out.content_type_lowcase = NULL;

    r->headers_out.location->hash = 0;
    r->headers_out.location = NULL;

    r->headers_out.content_length_n = size;

    if (r->headers_out.content_length) {
        r->headers_out.content_length->hash = 0;
        r->headers_out.content_length = NULL;
    }

    ngx_http_clear_accept_ranges(r);
    ngx_http_clear_last_modified(r);
    ngx_http_clear_etag(r);

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || r->header_only) {
        return rc;
    }

    b = ngx_create_temp_buf(r->pool, size);
    if (b == NULL) {
        return NGX_ERROR;
    }

    p = ngx_cpymem(b->pos, ngx_http_msie_refresh_head,
                   sizeof(ngx_http_msie_refresh_head) - 1);

    if (escape == 0) {
        p = ngx_cpymem(p, location, len);

    } else {
        p = (u_char *) ngx_escape_uri(p, location, len, NGX_ESCAPE_REFRESH);
    }
```

```
781
782     b->last = ngx_cpymem(p, ngx_http_msie_refresh_tail,
783                          sizeof(ngx_http_msie_refresh_tail) - 1);
784
785     b->last_buf = 1;
786     b->last_in_chain = 1;
787
788     out.buf = b;
789     out.next = NULL;
790
791     return ngx_http_output_filter(r, &out);
792 }
```

One Level Up Top Level

src/http/ngx_http_header_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_header_filter_module
- ngx_http_header_filter_module_ctx
- ngx_http_headers_out
- ngx_http_server_full_string
- ngx_http_server_string
- ngx_http_status_lines

Functions defined

- ngx_http_header_filter
- ngx_http_header_filter_init

Macros defined

- NGX_HTTP_LAST_2XX
- NGX_HTTP_LAST_3XX
- NGX_HTTP_LAST_4XX
- NGX_HTTP_LAST_5XX
- NGX_HTTP_OFF_3XX
- NGX_HTTP_OFF_4XX
- NGX_HTTP_OFF_5XX

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <nginx.h>
12
13
14  static ngx_int_t ngx_http_header_filter_init(ngx_conf_t *cf);
15  static ngx_int_t ngx_http_header_filter(ngx_http_request_t *r);
16
17
18  static ngx_http_module_t  ngx_http_header_filter_module_ctx = {
19      NULL,                                  /* preconfiguration */
20      ngx_http_header_filter_init,           /* postconfiguration */
21
22      NULL,                                  /* create main configuration */
23      NULL,                                  /* init main configuration */
24
```

```
    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL,                                  /* merge location configuration */
};


ngx_module_t  ngx_http_header_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_header_filter_module_ctx,    /* module context */
    NULL,                                  /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static char ngx_http_server_string[] = "Server: nginx" CRLF;
static char ngx_http_server_full_string[] = "Server: " NGINX_VER CRLF;


static ngx_str_t ngx_http_status_lines[] = {

    ngx_string("200 OK"),
    ngx_string("201 Created"),
    ngx_string("202 Accepted"),
    ngx_null_string,  /* "203 Non-Authoritative Information" */
    ngx_string("204 No Content"),
    ngx_null_string,  /* "205 Reset Content" */
    ngx_string("206 Partial Content"),

    /* ngx_null_string, */  /* "207 Multi-Status" */

#define NGX_HTTP_LAST_2XX  207
#define NGX_HTTP_OFF_3XX   (NGX_HTTP_LAST_2XX - 200)

    /* ngx_null_string, */  /* "300 Multiple Choices" */

    ngx_string("301 Moved Permanently"),
    ngx_string("302 Moved Temporarily"),
    ngx_string("303 See Other"),
    ngx_string("304 Not Modified"),
    ngx_null_string,  /* "305 Use Proxy" */
    ngx_null_string,  /* "306 unused" */
    ngx_string("307 Temporary Redirect"),

#define NGX_HTTP_LAST_3XX  308
#define NGX_HTTP_OFF_4XX   (NGX_HTTP_LAST_3XX - 301 + NGX_HTTP_OFF_3XX)

    ngx_string("400 Bad Request"),
    ngx_string("401 Unauthorized"),
    ngx_string("402 Payment Required"),
    ngx_string("403 Forbidden"),
    ngx_string("404 Not Found"),
    ngx_string("405 Not Allowed"),
    ngx_string("406 Not Acceptable"),
    ngx_null_string,  /* "407 Proxy Authentication Required" */
    ngx_string("408 Request Time-out"),
    ngx_string("409 Conflict"),
    ngx_string("410 Gone"),
    ngx_string("411 Length Required"),
    ngx_string("412 Precondition Failed"),
    ngx_string("413 Request Entity Too Large"),
    ngx_string("414 Request-URI Too Large"),
    ngx_string("415 Unsupported Media Type"),
    ngx_string("416 Requested Range Not Satisfiable"),

    /* ngx_null_string, */  /* "417 Expectation Failed" */
     /* ngx_null_string, */  /* "418 unused" */
```

```
    /* ngx_null_string, */  /* "419 unused" */
    /* ngx_null_string, */  /* "420 unused" */
    /* ngx_null_string, */  /* "421 unused" */
    /* ngx_null_string, */  /* "422 Unprocessable Entity" */
    /* ngx_null_string, */  /* "423 Locked" */
    /* ngx_null_string, */  /* "424 Failed Dependency" */

#define NGX_HTTP_LAST_4XX  417
#define NGX_HTTP_OFF_5XX   (NGX_HTTP_LAST_4XX - 400 + NGX_HTTP_OFF_4XX)

    ngx_string("500 Internal Server Error"),
    ngx_string("501 Not Implemented"),
    ngx_string("502 Bad Gateway"),
    ngx_string("503 Service Temporarily Unavailable"),
    ngx_string("504 Gateway Time-out"),

    ngx_null_string,        /* "505 HTTP Version Not Supported" */
    ngx_null_string,        /* "506 Variant Also Negotiates" */
    ngx_string("507 Insufficient Storage"),
    /* ngx_null_string, */  /* "508 unused" */
    /* ngx_null_string, */  /* "509 unused" */
    /* ngx_null_string, */  /* "510 Not Extended" */

#define NGX_HTTP_LAST_5XX  508

};


ngx_http_header_out_t  ngx_http_headers_out[] = {
    { ngx_string("Server"), offsetof(ngx_http_headers_out_t, server) },
    { ngx_string("Date"), offsetof(ngx_http_headers_out_t, date) },
    { ngx_string("Content-Length"),
                offsetof(ngx_http_headers_out_t, content_length) },
    { ngx_string("Content-Encoding"),
                offsetof(ngx_http_headers_out_t, content_encoding) },
    { ngx_string("Location"), offsetof(ngx_http_headers_out_t, location) },
    { ngx_string("Last-Modified"),
                offsetof(ngx_http_headers_out_t, last_modified) },
    { ngx_string("Accept-Ranges"),
                offsetof(ngx_http_headers_out_t, accept_ranges) },
    { ngx_string("Expires"), offsetof(ngx_http_headers_out_t, expires) },
    { ngx_string("Cache-Control"),
                offsetof(ngx_http_headers_out_t, cache_control) },
    { ngx_string("ETag"), offsetof(ngx_http_headers_out_t, etag) },

    { ngx_null_string, 0 }
};


static ngx_int_t
ngx_http_header_filter(ngx_http_request_t *r)
{
    u_char                    *p;
    size_t                     len;
    ngx_str_t                  host, *status_line;
    ngx_buf_t                 *b;
    ngx_uint_t                 status, i, port;
    ngx_chain_t                out;
    ngx_list_part_t           *part;
    ngx_table_elt_t           *header;
    ngx_connection_t          *c;
    ngx_http_core_loc_conf_t  *clcf;
    ngx_http_core_srv_conf_t  *cscf;
    struct sockaddr_in        *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6       *sin6;
#endif
    u_char                     addr[NGX_SOCKADDR_STRLEN];

    if (r->header_sent) {
        return NGX_OK;
    }

    r->header_sent = 1;

    if (r != r->main) {
```

```
        return NGX_OK;
    }

    if (r->http_version < NGX_HTTP_VERSION_10) {
        return NGX_OK;
    }

    if (r->method == NGX_HTTP_HEAD) {
        r->header_only = 1;
    }

    if (r->headers_out.last_modified_time != -1) {
        if (r->headers_out.status != NGX_HTTP_OK
            && r->headers_out.status != NGX_HTTP_PARTIAL_CONTENT
            && r->headers_out.status != NGX_HTTP_NOT_MODIFIED)
        {
            r->headers_out.last_modified_time = -1;
            r->headers_out.last_modified = NULL;
        }
    }

    len = sizeof("HTTP/1.x ") - 1 + sizeof(CRLF) - 1
          /* the end of the header */
          + sizeof(CRLF) - 1;

    /* status line */

    if (r->headers_out.status_line.len) {
        len += r->headers_out.status_line.len;
        status_line = &r->headers_out.status_line;
#if (NGX_SUPPRESS_WARN)
        status = 0;
#endif

    } else {

        status = r->headers_out.status;

        if (status >= NGX_HTTP_OK
            && status < NGX_HTTP_LAST_2XX)
        {
            /* 2XX */

            if (status == NGX_HTTP_NO_CONTENT) {
                r->header_only = 1;
                ngx_str_null(&r->headers_out.content_type);
                r->headers_out.last_modified_time = -1;
                r->headers_out.last_modified = NULL;
                r->headers_out.content_length = NULL;
                r->headers_out.content_length_n = -1;
            }

            status -= NGX_HTTP_OK;
            status_line = &ngx_http_status_lines[status];
            len += ngx_http_status_lines[status].len;

        } else if (status >= NGX_HTTP_MOVED_PERMANENTLY
                   && status < NGX_HTTP_LAST_3XX)
        {
            /* 3XX */

            if (status == NGX_HTTP_NOT_MODIFIED) {
                r->header_only = 1;
            }

            status = status - NGX_HTTP_MOVED_PERMANENTLY + NGX_HTTP_OFF_3XX;
            status_line = &ngx_http_status_lines[status];
            len += ngx_http_status_lines[status].len;

        } else if (status >= NGX_HTTP_BAD_REQUEST
                   && status < NGX_HTTP_LAST_4XX)
        {
            /* 4XX */
            status = status - NGX_HTTP_BAD_REQUEST
                            + NGX_HTTP_OFF_4XX;

```

```
253             status_line = &ngx_http_status_lines[status];
254             len += ngx_http_status_lines[status].len;
255 
256         } else if (status >= NGX_HTTP_INTERNAL_SERVER_ERROR
257                    && status < NGX_HTTP_LAST_5XX)
258         {
259             /* 5XX */
260             status = status - NGX_HTTP_INTERNAL_SERVER_ERROR
261                             + NGX_HTTP_OFF_5XX;
262 
263             status_line = &ngx_http_status_lines[status];
264             len += ngx_http_status_lines[status].len;
265 
266         } else {
267             len += NGX_INT_T_LEN + 1 /* SP */;
268             status_line = NULL;
269         }
270 
271         if (status_line && status_line->len == 0) {
272             status = r->headers_out.status;
273             len += NGX_INT_T_LEN + 1 /* SP */;
274             status_line = NULL;
275         }
276     }
277 
278     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
279 
280     if (r->headers_out.server == NULL) {
281         len += clcf->server_tokens ? sizeof(ngx_http_server_full_string) - 1:
282                                      sizeof(ngx_http_server_string) - 1;
283     }
284 
285     if (r->headers_out.date == NULL) {
286         len += sizeof("Date: Mon, 28 Sep 1970 06:00:00 GMT" CRLF) - 1;
287     }
288 
289     if (r->headers_out.content_type.len) {
290         len += sizeof("Content-Type: ") - 1
291                + r->headers_out.content_type.len + 2;
292 
293         if (r->headers_out.content_type_len == r->headers_out.content_type.len
294             && r->headers_out.charset.len)
295         {
296             len += sizeof("; charset=") - 1 + r->headers_out.charset.len;
297         }
298     }
299 
300     if (r->headers_out.content_length == NULL
301         && r->headers_out.content_length_n >= 0)
302     {
303         len += sizeof("Content-Length: ") - 1 + NGX_OFF_T_LEN + 2;
304     }
305 
306     if (r->headers_out.last_modified == NULL
307         && r->headers_out.last_modified_time != -1)
308     {
309         len += sizeof("Last-Modified: Mon, 28 Sep 1970 06:00:00 GMT" CRLF) - 1;
310     }
311 
312     c = r->connection;
313 
314     if (r->headers_out.location
315         && r->headers_out.location->value.len
316         && r->headers_out.location->value.data[0] == '/')
317     {
318         r->headers_out.location->hash = 0;
319 
320         if (clcf->server_name_in_redirect) {
321             cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
322             host = cscf->server_name;
323 
324         } else if (r->headers_in.server.len) {
325             host = r->headers_in.server;
326 
327         } else {
328             host.len = NGX_SOCKADDR_STRLEN;
```

```
            host.data = addr;

            if (ngx_connection_local_sockaddr(c, &host, 0) != NGX_OK) {
                return NGX_ERROR;
            }
        }

        switch (c->local_sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            sin6 = (struct sockaddr_in6 *) c->local_sockaddr;
            port = ntohs(sin6->sin6_port);
            break;
#endif
#if (NGX_HAVE_UNIX_DOMAIN)
        case AF_UNIX:
            port = 0;
            break;
#endif
        default: /* AF_INET */
            sin = (struct sockaddr_in *) c->local_sockaddr;
            port = ntohs(sin->sin_port);
            break;
        }

        len += sizeof("Location: https://") - 1
               + host.len
               + r->headers_out.location->value.len + 2;

        if (clcf->port_in_redirect) {

#if (NGX_HTTP_SSL)
            if (c->ssl)
                port = (port == 443) ? 0 : port;
            else
#endif
                port = (port == 80) ? 0 : port;

        } else {
            port = 0;
        }

        if (port) {
            len += sizeof(":65535") - 1;
        }

    } else {
        ngx_str_null(&host);
        port = 0;
    }

    if (r->chunked) {
        len += sizeof("Transfer-Encoding: chunked" CRLF) - 1;
    }

    if (r->headers_out.status == NGX_HTTP_SWITCHING_PROTOCOLS) {
        len += sizeof("Connection: upgrade" CRLF) - 1;

    } else if (r->keepalive) {
        len += sizeof("Connection: keep-alive" CRLF) - 1;

        /*
         * MSIE and Opera ignore the "Keep-Alive: timeout=<N>" header.
         * MSIE keeps the connection alive for about 60-65 seconds.
         * Opera keeps the connection alive very long.
         * Mozilla keeps the connection alive for N plus about 1-10 seconds.
         * Konqueror keeps the connection alive for about N seconds.
         */

        if (clcf->keepalive_header) {
            len += sizeof("Keep-Alive: timeout=") - 1 + NGX_TIME_T_LEN + 2;
        }

    } else {
        len += sizeof("Connection: close" CRLF) - 1;
```

```
    }

#if (NGX_HTTP_GZIP)
    if (r->gzip_vary) {
        if (clcf->gzip_vary) {
            len += sizeof("Vary: Accept-Encoding" CRLF) - 1;

        } else {
            r->gzip_vary = 0;
        }
    }
#endif

    part = &r->headers_out.headers.part;
    header = part->elts;

    for (i = 0; /* void */; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            header = part->elts;
            i = 0;
        }

        if (header[i].hash == 0) {
            continue;
        }

        len += header[i].key.len + sizeof(": ") - 1 + header[i].value.len
               + sizeof(CRLF) - 1;
    }

    b = ngx_create_temp_buf(r->pool, len);
    if (b == NULL) {
        return NGX_ERROR;
    }

    /* "HTTP/1.x " */
    b->last = ngx_cpymem(b->last, "HTTP/1.1 ", sizeof("HTTP/1.x ") - 1);

    /* status line */
    if (status_line) {
        b->last = ngx_copy(b->last, status_line->data, status_line->len);

    } else {
        b->last = ngx_sprintf(b->last, "%03ui ", status);
    }
    *b->last++ = CR; *b->last++ = LF;

    if (r->headers_out.server == NULL) {
        if (clcf->server_tokens) {
            p = (u_char *) ngx_http_server_full_string;
            len = sizeof(ngx_http_server_full_string) - 1;

        } else {
            p = (u_char *) ngx_http_server_string;
            len = sizeof(ngx_http_server_string) - 1;
        }

        b->last = ngx_cpymem(b->last, p, len);
    }

    if (r->headers_out.date == NULL) {
        b->last = ngx_cpymem(b->last, "Date: ", sizeof("Date: ") - 1);
        b->last = ngx_cpymem(b->last, ngx_cached_http_time.data,
                             ngx_cached_http_time.len);

        *b->last++ = CR; *b->last++ = LF;
    }

    if (r->headers_out.content_type.len) {
        b->last = ngx_cpymem(b->last, "Content-Type: ",
```

```
                            sizeof("Content-Type: ") - 1);
        p = b->last;
        b->last = ngx_copy(b->last, r->headers_out.content_type.data,
                           r->headers_out.content_type.len);

        if (r->headers_out.content_type_len == r->headers_out.content_type.len
            && r->headers_out.charset.len)
        {
            b->last = ngx_cpymem(b->last, "; charset=",
                                 sizeof("; charset=") - 1);
            b->last = ngx_copy(b->last, r->headers_out.charset.data,
                               r->headers_out.charset.len);

            /* update r->headers_out.content_type for possible logging */

            r->headers_out.content_type.len = b->last - p;
            r->headers_out.content_type.data = p;
        }

        *b->last++ = CR; *b->last++ = LF;
    }

    if (r->headers_out.content_length == NULL
        && r->headers_out.content_length_n >= 0)
    {
        b->last = ngx_sprintf(b->last, "Content-Length: %O" CRLF,
                              r->headers_out.content_length_n);
    }

    if (r->headers_out.last_modified == NULL
        && r->headers_out.last_modified_time != -1)
    {
        b->last = ngx_cpymem(b->last, "Last-Modified: ",
                             sizeof("Last-Modified: ") - 1);
        b->last = ngx_http_time(b->last, r->headers_out.last_modified_time);

        *b->last++ = CR; *b->last++ = LF;
    }

    if (host.data) {

        p = b->last + sizeof("Location: ") - 1;

        b->last = ngx_cpymem(b->last, "Location: http",
                             sizeof("Location: http") - 1);

#if (NGX_HTTP_SSL)
        if (c->ssl) {
            *b->last++ ='s';
        }
#endif

        *b->last++ = ':'; *b->last++ = '/'; *b->last++ = '/';
        b->last = ngx_copy(b->last, host.data, host.len);

        if (port) {
            b->last = ngx_sprintf(b->last, ":%ui", port);
        }

        b->last = ngx_copy(b->last, r->headers_out.location->value.data,
                           r->headers_out.location->value.len);

        /* update r->headers_out.location->value for possible logging */

        r->headers_out.location->value.len = b->last - p;
        r->headers_out.location->value.data = p;
        ngx_str_set(&r->headers_out.location->key, "Location");

        *b->last++ = CR; *b->last++ = LF;
    }

    if (r->chunked) {
        b->last = ngx_cpymem(b->last, "Transfer-Encoding: chunked" CRLF,
                             sizeof("Transfer-Encoding: chunked" CRLF) - 1);
    }

```

```
    if (r->headers_out.status == NGX_HTTP_SWITCHING_PROTOCOLS) {
        b->last = ngx_cpymem(b->last, "Connection: upgrade" CRLF,
                             sizeof("Connection: upgrade" CRLF) - 1);

    } else if (r->keepalive) {
        b->last = ngx_cpymem(b->last, "Connection: keep-alive" CRLF,
                             sizeof("Connection: keep-alive" CRLF) - 1);

        if (clcf->keepalive_header) {
            b->last = ngx_sprintf(b->last, "Keep-Alive: timeout=%T" CRLF,
                                  clcf->keepalive_header);
        }

    } else {
        b->last = ngx_cpymem(b->last, "Connection: close" CRLF,
                             sizeof("Connection: close" CRLF) - 1);
    }

#if (NGX_HTTP_GZIP)
    if (r->gzip_vary) {
        b->last = ngx_cpymem(b->last, "Vary: Accept-Encoding" CRLF,
                             sizeof("Vary: Accept-Encoding" CRLF) - 1);
    }
#endif

    part = &r->headers_out.headers.part;
    header = part->elts;

    for (i = 0; /* void */; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            header = part->elts;
            i = 0;
        }

        if (header[i].hash == 0) {
            continue;
        }

        b->last = ngx_copy(b->last, header[i].key.data, header[i].key.len);
        *b->last++ = ':'; *b->last++ = ' ';

        b->last = ngx_copy(b->last, header[i].value.data, header[i].value.len);
        *b->last++ = CR; *b->last++ = LF;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "%*s", (size_t) (b->last - b->pos), b->pos);

    /* the end of HTTP header */
    *b->last++ = CR; *b->last++ = LF;

    r->header_size = b->last - b->pos;

    if (r->header_only) {
        b->last_buf = 1;
    }

    out.buf = b;
    out.next = NULL;

    return ngx_http_write_filter(r, &out);
}


static ngx_int_t
ngx_http_header_filter_init(ngx_conf_t *cf)
{
    ngx_http_top_header_filter = ngx_http_header_filter;

    return NGX_OK;
```

```
633 }
```

One Level Up Top Level

src/http/ngx_http_write_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_write_filter_module
- ngx_http_write_filter_module_ctx

Functions defined

- ngx_http_write_filter
- ngx_http_write_filter_init

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10 #include <ngx_http.h>
11
12
13 static ngx_int_t ngx_http_write_filter_init(ngx_conf_t *cf);
14
15
16 static ngx_http_module_t  ngx_http_write_filter_module_ctx = {
17     NULL,                                  /* preconfiguration */
18     ngx_http_write_filter_init,            /* postconfiguration */
19
20     NULL,                                  /* create main configuration */
21     NULL,                                  /* init main configuration */
22
23     NULL,                                  /* create server configuration */
24     NULL,                                  /* merge server configuration */
25
26     NULL,                                  /* create location configuration */
27     NULL,                                  /* merge location configuration */
28 };
29
30
31 ngx_module_t  ngx_http_write_filter_module = {
32     NGX_MODULE_V1,
33     &ngx_http_write_filter_module_ctx,     /* module context */
34     NULL,                                  /* module directives */
35     NGX_HTTP_MODULE,                       /* module type */
36     NULL,                                  /* init master */
37     NULL,                                  /* init module */
38     NULL,                                  /* init process */
39     NULL,                                  /* init thread */
40     NULL,                                  /* exit thread */
41     NULL,                                  /* exit process */
42     NULL,                                  /* exit master */
43     NGX_MODULE_V1_PADDING
44 };
45
46
47 ngx_int_t
48 ngx_http_write_filter(ngx_http_request_t *r, ngx_chain_t *in)
49 {
50     off_t                      size, sent, nsent, limit;
51     ngx_uint_t                 last, flush, sync;
52     ngx_msec_t                 delay;
```

```
    ngx_chain_t               *cl, *ln, **ll, *chain;
    ngx_connection_t          *c;
    ngx_http_core_loc_conf_t  *clcf;

    c = r->connection;

    if (c->error) {
        return NGX_ERROR;
    }

    size = 0;
    flush = 0;
    sync = 0;
    last = 0;
    ll = &r->out;

    /* find the size, the flush point and the last link of the saved chain */

    for (cl = r->out; cl; cl = cl->next) {
        ll = &cl->next;

        ngx_log_debug7(NGX_LOG_DEBUG_EVENT, c->log, 0,
                       "write old buf t:%d f:%d %p, pos %p, size: %z "
                       "file: %O, size: %z",
                       cl->buf->temporary, cl->buf->in_file,
                       cl->buf->start, cl->buf->pos,
                       cl->buf->last - cl->buf->pos,
                       cl->buf->file_pos,
                       cl->buf->file_last - cl->buf->file_pos);

#if 1
        if (ngx_buf_size(cl->buf) == 0 && !ngx_buf_special(cl->buf)) {
            ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                          "zero size buf in writer "
                          "t:%d r:%d f:%d %p %p-%p %p %O-%O",
                          cl->buf->temporary,
                          cl->buf->recycled,
                          cl->buf->in_file,
                          cl->buf->start,
                          cl->buf->pos,
                          cl->buf->last,
                          cl->buf->file,
                          cl->buf->file_pos,
                          cl->buf->file_last);

            ngx_debug_point();
            return NGX_ERROR;
        }
#endif

        size += ngx_buf_size(cl->buf);

        if (cl->buf->flush || cl->buf->recycled) {
            flush = 1;
        }

        if (cl->buf->sync) {
            sync = 1;
        }

        if (cl->buf->last_buf) {
            last = 1;
        }
    }

    /* add the new chain to the existent one */

    for (ln = in; ln; ln = ln->next) {
        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        cl->buf = ln->buf;
        *ll = cl;
        ll = &cl->next;
```

```
129
130         ngx_log_debug7(NGX_LOG_DEBUG_EVENT, c->log, 0,
131                        "write new buf t:%d f:%d %p, pos %p, size: %z "
132                        "file: %O, size: %z",
133                        cl->buf->temporary, cl->buf->in_file,
134                        cl->buf->start, cl->buf->pos,
135                        cl->buf->last - cl->buf->pos,
136                        cl->buf->file_pos,
137                        cl->buf->file_last - cl->buf->file_pos);
138
139 #if 1
140         if (ngx_buf_size(cl->buf) == 0 && !ngx_buf_special(cl->buf)) {
141             ngx_log_error(NGX_LOG_ALERT, c->log, 0,
142                           "zero size buf in writer "
143                           "t:%d r:%d f:%d %p %p-%p %p %O-%O",
144                           cl->buf->temporary,
145                           cl->buf->recycled,
146                           cl->buf->in_file,
147                           cl->buf->start,
148                           cl->buf->pos,
149                           cl->buf->last,
150                           cl->buf->file,
151                           cl->buf->file_pos,
152                           cl->buf->file_last);
153
154             ngx_debug_point();
155             return NGX_ERROR;
156         }
157 #endif
158
159         size += ngx_buf_size(cl->buf);
160
161         if (cl->buf->flush || cl->buf->recycled) {
162             flush = 1;
163         }
164
165         if (cl->buf->sync) {
166             sync = 1;
167         }
168
169         if (cl->buf->last_buf) {
170             last = 1;
171         }
172     }
173
174     *ll = NULL;
175
176     ngx_log_debug3(NGX_LOG_DEBUG_HTTP, c->log, 0,
177                    "http write filter: l:%d f:%d s:%O", last, flush, size);
178
179     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
180
181     /*
182      * avoid the output if there are no last buf, no flush point,
183      * there are the incoming bufs and the size of all bufs
184      * is smaller than "postpone_output" directive
185      */
186
187     if (!last && !flush && in && size < (off_t) clcf->postpone_output) {
188         return NGX_OK;
189     }
190
191     if (c->write->delayed) {
192         c->buffered |= NGX_HTTP_WRITE_BUFFERED;
193         return NGX_AGAIN;
194     }
195
196     if (size == 0
197         && !(c->buffered & NGX_LOWLEVEL_BUFFERED)
198         && !(last && c->need_last_buf))
199     {
200         if (last || flush || sync) {
201             for (cl = r->out; cl; /* void */) {
202                 ln = cl;
203                 cl = cl->next;
204                 ngx_free_chain(r->pool, ln);
```

```
            }

            r->out = NULL;
            c->buffered &= ~NGX_HTTP_WRITE_BUFFERED;

            return NGX_OK;
        }

        ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                      "the http output chain is empty");

        ngx_debug_point();

        return NGX_ERROR;
    }

    if (r->limit_rate) {
        if (r->limit_rate_after == 0) {
            r->limit_rate_after = clcf->limit_rate_after;
        }

        limit = (off_t) r->limit_rate * (ngx_time() - r->start_sec + 1)
                - (c->sent - r->limit_rate_after);

        if (limit <= 0) {
            c->write->delayed = 1;
            delay = (ngx_msec_t) (- limit * 1000 / r->limit_rate + 1);
            ngx_add_timer(c->write, delay);

            c->buffered |= NGX_HTTP_WRITE_BUFFERED;

            return NGX_AGAIN;
        }

        if (clcf->sendfile_max_chunk
            && (off_t) clcf->sendfile_max_chunk < limit)
        {
            limit = clcf->sendfile_max_chunk;
        }

    } else {
        limit = clcf->sendfile_max_chunk;
    }

    sent = c->sent;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http write filter limit %O", limit);

    chain = c->send_chain(c, r->out, limit);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http write filter %p", chain);

    if (chain == NGX_CHAIN_ERROR) {
        c->error = 1;
        return NGX_ERROR;
    }

    if (r->limit_rate) {

        nsent = c->sent;

        if (r->limit_rate_after) {

            sent -= r->limit_rate_after;
            if (sent < 0) {
                sent = 0;
            }

            nsent -= r->limit_rate_after;
            if (nsent < 0) {
                nsent = 0;
            }
        }

```

```
281         delay = (ngx_msec_t) ((nsent - sent) * 1000 / r->limit_rate);
282
283         if (delay > 0) {
284             limit = 0;
285             c->write->delayed = 1;
286             ngx_add_timer(c->write, delay);
287         }
288     }
289
290     if (limit
291         && c->write->ready
292         && c->sent - sent >= limit - (off_t) (2 * ngx_pagesize))
293     {
294         c->write->delayed = 1;
295         ngx_add_timer(c->write, 1);
296     }
297
298     for (cl = r->out; cl && cl != chain; /* void */) {
299         ln = cl;
300         cl = cl->next;
301         ngx_free_chain(r->pool, ln);
302     }
303
304     r->out = chain;
305
306     if (chain) {
307         c->buffered |= NGX_HTTP_WRITE_BUFFERED;
308         return NGX_AGAIN;
309     }
310
311     c->buffered &= ~NGX_HTTP_WRITE_BUFFERED;
312
313     if ((c->buffered & NGX_LOWLEVEL_BUFFERED) && r->postponed == NULL) {
314         return NGX_AGAIN;
315     }
316
317     return NGX_OK;
318 }
319
320
321 static ngx_int_t
322 ngx_http_write_filter_init(ngx_conf_t *cf)
323 {
324     ngx_http_top_body_filter = ngx_http_write_filter;
325
326     return NGX_OK;
327 }
```

One Level Up Top Level

src/os/unix/ngx_file_aio_read.c - nginx-1.7.10

Functions defined

- ngx_file_aio_event_handler
- ngx_file_aio_read
- ngx_file_aio_result

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10 #include <ngx_event.h>
11
12
13 /*
14 * FreeBSD file AIO features and quirks:
15 *
16 *    if an asked data are already in VM cache, then aio_error() returns 0,
17 *    and the data are already copied in buffer;
18 *
19 *    aio_read() preread in VM cache as minimum 16K (probably BKVASIZE);
20 *    the first AIO preload may be up to 128K;
21 *
22 *    aio_read/aio_error() may return EINPROGRESS for just written data;
23 *
24 *    kqueue EVFILT_AIO filter is level triggered only: an event repeats
25 *    until aio_return() will be called;
26 *
27 *    aio_cancel() cannot cancel file AIO: it returns AIO_NOTCANCELED always.
28 */
29
30
31 extern int  ngx_kqueue;
32
33
34 static ssize_t ngx_file_aio_result(ngx_file_t *file, ngx_event_aio_t *aio,
35     ngx_event_t *ev);
36 static void ngx_file_aio_event_handler(ngx_event_t *ev);
37
38
39 ssize_t
40 ngx_file_aio_read(ngx_file_t *file, u_char *buf, size_t size, off_t offset,
41     ngx_pool_t *pool)
42 {
43     int               n;
44     ngx_event_t      *ev;
45     ngx_event_aio_t  *aio;
46
47     if (!ngx_file_aio) {
48         return ngx_read_file(file, buf, size, offset);
49     }
50
51     aio = file->aio;
52
53     if (aio == NULL) {
54         aio = ngx_pcalloc(pool, sizeof(ngx_event_aio_t));
55         if (aio == NULL) {
56             return NGX_ERROR;
57         }
58
```

```
        aio->file = file;
        aio->fd = file->fd;
        aio->event.data = aio;
        aio->event.ready = 1;
        aio->event.log = file->log;
#if (NGX_HAVE_AIO_SENDFILE)
        aio->last_offset = -1;
#endif
        file->aio = aio;
    }

    ev = &aio->event;

    if (!ev->ready) {
        ngx_log_error(NGX_LOG_ALERT, file->log, 0,
                      "second aio post for \"%V\"", &file->name);
        return NGX_AGAIN;
    }

    ngx_log_debug4(NGX_LOG_DEBUG_CORE, file->log, 0,
                   "aio complete:%d @%O:%z %V",
                   ev->complete, offset, size, &file->name);

    if (ev->complete) {
        ev->complete = 0;
        ngx_set_errno(aio->err);

        if (aio->err == 0) {
            return aio->nbytes;
        }

        ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                      "aio read \"%s\" failed", file->name.data);

        return NGX_ERROR;
    }

    ngx_memzero(&aio->aiocb, sizeof(struct aiocb));

    aio->aiocb.aio_fildes = file->fd;
    aio->aiocb.aio_offset = offset;
     aio->aiocb.aio_buf = buf;
     aio->aiocb.aio_nbytes = size;
 #if (NGX_HAVE_KQUEUE)
     aio->aiocb.aio_sigevent.sigev_notify_kqueue = ngx_kqueue;
     aio->aiocb.aio_sigevent.sigev_notify = SIGEV_KEVENT;
     aio->aiocb.aio_sigevent.sigev_value.sigval_ptr = ev;
 #endif
     ev->handler = ngx_file_aio_event_handler;

     n = aio_read(&aio->aiocb);

     if (n == -1) {
         n = ngx_errno;

         if (n == NGX_EAGAIN) {
             return ngx_read_file(file, buf, size, offset);
         }

         ngx_log_error(NGX_LOG_CRIT, file->log, n,
                       "aio_read(\"%V\") failed", &file->name);

         if (n == NGX_ENOSYS) {
             ngx_file_aio = 0;
             return ngx_read_file(file, buf, size, offset);
         }

         return NGX_ERROR;
     }

     ngx_log_debug2(NGX_LOG_DEBUG_CORE, file->log, 0,
                    "aio_read: fd:%d %d", file->fd, n);

     ev->active = 1;
     ev->ready = 0;
     ev->complete = 0;
```

```
135
136     return ngx_file_aio_result(aio->file, aio, ev);
137 }
138
139
140 static ssize_t
141 ngx_file_aio_result(ngx_file_t *file, ngx_event_aio_t *aio, ngx_event_t *ev)
142 {
143     int        n;
144     ngx_err_t  err;
145
146     n = aio_error(&aio->aiocb);
147
148     ngx_log_debug2(NGX_LOG_DEBUG_CORE, file->log, 0,
149                    "aio_error: fd:%d %d", file->fd, n);
150
151     if (n == -1) {
152         err = ngx_errno;
153         aio->err = err;
154
155         ngx_log_error(NGX_LOG_ALERT, file->log, err,
156                       "aio_error(\"%V\") failed", &file->name);
157         return NGX_ERROR;
158     }
159
160     if (n == NGX_EINPROGRESS) {
161         if (ev->ready) {
162             ev->ready = 0;
163             ngx_log_error(NGX_LOG_ALERT, file->log, n,
164                           "aio_read(\"%V\") still in progress",
165                           &file->name);
166         }
167
168         return NGX_AGAIN;
169     }
170
171     n = aio_return(&aio->aiocb);
172
173     if (n == -1) {
174         err = ngx_errno;
175         aio->err = err;
176         ev->ready = 1;
177
178         ngx_log_error(NGX_LOG_CRIT, file->log, err,
179                       "aio_return(\"%V\") failed", &file->name);
180         return NGX_ERROR;
181     }
182
183     aio->err = 0;
184     aio->nbytes = n;
185     ev->ready = 1;
186     ev->active = 0;
187
188     ngx_log_debug2(NGX_LOG_DEBUG_CORE, file->log, 0,
189                    "aio_return: fd:%d %d", file->fd, n);
190
191     return n;
192 }
193
194
195 static void
196 ngx_file_aio_event_handler(ngx_event_t *ev)
197 {
198     ngx_event_aio_t  *aio;
199
200     aio = ev->data;
201
202     ngx_log_debug2(NGX_LOG_DEBUG_CORE, ev->log, 0,
203                    "aio event handler fd:%d %V", aio->fd, &aio->file->name);
204
205     if (ngx_file_aio_result(aio->file, aio, ev) != NGX_AGAIN) {
206         aio->handler(ev);
207     }
208 }
```

One Level Up Top Level

src/http/ngx_http_upstream.c - nginx-1.7.10

Global variables defined

- ngx_http_upstream_cache_method_mask
- ngx_http_upstream_commands
- ngx_http_upstream_headers_in
- ngx_http_upstream_ignore_headers_masks
- ngx_http_upstream_module
- ngx_http_upstream_module_ctx
- ngx_http_upstream_next_errors
- ngx_http_upstream_vars

Functions defined

- ngx_http_upstream
- ngx_http_upstream_add
- ngx_http_upstream_add_variables
- ngx_http_upstream_addr_variable
- ngx_http_upstream_bind_set_slot
- ngx_http_upstream_cache
- ngx_http_upstream_cache_etag
- ngx_http_upstream_cache_get
- ngx_http_upstream_cache_last_modified
- ngx_http_upstream_cache_send
- ngx_http_upstream_cache_status
- ngx_http_upstream_check_broken_connection
- ngx_http_upstream_cleanup
- ngx_http_upstream_connect
- ngx_http_upstream_cookie_variable
- ngx_http_upstream_copy_allow_ranges
- ngx_http_upstream_copy_content_encoding
- ngx_http_upstream_copy_content_type
- ngx_http_upstream_copy_header_line
- ngx_http_upstream_copy_last_modified

- ngx_http_upstream_copy_multi_header_lines
- ngx_http_upstream_create
- ngx_http_upstream_create_main_conf
- ngx_http_upstream_dummy_handler
- ngx_http_upstream_finalize_request
- ngx_http_upstream_get_local
- ngx_http_upstream_handler
- ngx_http_upstream_header_variable
- ngx_http_upstream_hide_headers_hash
- ngx_http_upstream_ignore_header_line
- ngx_http_upstream_init
- ngx_http_upstream_init_main_conf
- ngx_http_upstream_init_request
- ngx_http_upstream_intercept_errors
- ngx_http_upstream_next
- ngx_http_upstream_non_buffered_filter
- ngx_http_upstream_non_buffered_filter_init
- ngx_http_upstream_param_set_slot
- ngx_http_upstream_process_accel_expires
- ngx_http_upstream_process_body_in_memory
- ngx_http_upstream_process_buffering
- ngx_http_upstream_process_cache_control
- ngx_http_upstream_process_charset
- ngx_http_upstream_process_connection
- ngx_http_upstream_process_content_length
- ngx_http_upstream_process_downstream
- ngx_http_upstream_process_expires
- ngx_http_upstream_process_header
- ngx_http_upstream_process_header_line
- ngx_http_upstream_process_headers
- ngx_http_upstream_process_last_modified
- ngx_http_upstream_process_limit_rate
- ngx_http_upstream_process_non_buffered_downstream

- ngx_http_upstream_process_non_buffered_request
- ngx_http_upstream_process_non_buffered_upstream
- ngx_http_upstream_process_request
- ngx_http_upstream_process_set_cookie
- ngx_http_upstream_process_transfer_encoding
- ngx_http_upstream_process_upgraded
- ngx_http_upstream_process_upstream
- ngx_http_upstream_process_vary
- ngx_http_upstream_rd_check_broken_connection
- ngx_http_upstream_reinit
- ngx_http_upstream_resolve_handler
- ngx_http_upstream_response_length_variable
- ngx_http_upstream_response_time_variable
- ngx_http_upstream_rewrite_location
- ngx_http_upstream_rewrite_refresh
- ngx_http_upstream_rewrite_set_cookie
- ngx_http_upstream_send_request
- ngx_http_upstream_send_request_handler
- ngx_http_upstream_send_response
- ngx_http_upstream_server
- ngx_http_upstream_ssl_handshake
- ngx_http_upstream_ssl_init_connection
- ngx_http_upstream_ssl_name
- ngx_http_upstream_status_variable
- ngx_http_upstream_store
- ngx_http_upstream_test_connect
- ngx_http_upstream_test_next
- ngx_http_upstream_upgrade
- ngx_http_upstream_upgraded_read_downstream
- ngx_http_upstream_upgraded_read_upstream
- ngx_http_upstream_upgraded_write_downstream
- ngx_http_upstream_upgraded_write_upstream
- ngx_http_upstream_wr_check_broken_connection

Source code

```

/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_http.h>


 #if (NGX_HTTP_CACHE)
 static ngx_int_t ngx_http_upstream_cache(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static ngx_int_t ngx_http_upstream_cache_get(ngx_http_request_t *r,
     ngx_http_upstream_t *u, ngx_http_file_cache_t **cache);
 static ngx_int_t ngx_http_upstream_cache_send(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static ngx_int_t ngx_http_upstream_cache_status(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
 static ngx_int_t ngx_http_upstream_cache_last_modified(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
 static ngx_int_t ngx_http_upstream_cache_etag(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
 #endif

 static void ngx_http_upstream_init_request(ngx_http_request_t *r);
 static void ngx_http_upstream_resolve_handler(ngx_resolver_ctx_t *ctx);
 static void ngx_http_upstream_rd_check_broken_connection(ngx_http_request_t *r);
 static void ngx_http_upstream_wr_check_broken_connection(ngx_http_request_t *r);
 static void ngx_http_upstream_check_broken_connection(ngx_http_request_t *r,
     ngx_event_t *ev);
 static void ngx_http_upstream_connect(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static ngx_int_t ngx_http_upstream_reinit(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_send_request(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_send_request_handler(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_process_header(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static ngx_int_t ngx_http_upstream_test_next(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static ngx_int_t ngx_http_upstream_intercept_errors(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static ngx_int_t ngx_http_upstream_test_connect(ngx_connection_t *c);
 static ngx_int_t ngx_http_upstream_process_headers(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_process_body_in_memory(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_send_response(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_upgrade(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_upgraded_read_downstream(ngx_http_request_t *r);
 static void ngx_http_upstream_upgraded_write_downstream(ngx_http_request_t *r);
 static void ngx_http_upstream_upgraded_read_upstream(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_upgraded_write_upstream(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void ngx_http_upstream_process_upgraded(ngx_http_request_t *r,
     ngx_uint_t from_upstream, ngx_uint_t do_write);
 static void
     ngx_http_upstream_process_non_buffered_downstream(ngx_http_request_t *r);
 static void
     ngx_http_upstream_process_non_buffered_upstream(ngx_http_request_t *r,
     ngx_http_upstream_t *u);
 static void
     ngx_http_upstream_process_non_buffered_request(ngx_http_request_t *r,
     ngx_uint_t do_write);
 static ngx_int_t ngx_http_upstream_non_buffered_filter_init(void *data);
```

```
static ngx_int_t ngx_http_upstream_non_buffered_filter(void *data,
    ssize_t bytes);
static void ngx_http_upstream_process_downstream(ngx_http_request_t *r);
static void ngx_http_upstream_process_upstream(ngx_http_request_t *r,
    ngx_http_upstream_t *u);
static void ngx_http_upstream_process_request(ngx_http_request_t *r);
static void ngx_http_upstream_store(ngx_http_request_t *r,
    ngx_http_upstream_t *u);
static void ngx_http_upstream_dummy_handler(ngx_http_request_t *r,
    ngx_http_upstream_t *u);
static void ngx_http_upstream_next(ngx_http_request_t *r,
    ngx_http_upstream_t *u, ngx_uint_t ft_type);
static void ngx_http_upstream_cleanup(void *data);
static void ngx_http_upstream_finalize_request(ngx_http_request_t *r,
    ngx_http_upstream_t *u, ngx_int_t rc);

static ngx_int_t ngx_http_upstream_process_header_line(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_content_length(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_last_modified(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_set_cookie(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t
    ngx_http_upstream_process_cache_control(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_ignore_header_line(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_expires(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_accel_expires(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_limit_rate(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_buffering(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_charset(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_connection(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t
     ngx_http_upstream_process_transfer_encoding(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_process_vary(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_copy_header_line(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t
     ngx_http_upstream_copy_multi_header_lines(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_copy_content_type(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_copy_last_modified(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_rewrite_location(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_rewrite_refresh(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_rewrite_set_cookie(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
static ngx_int_t ngx_http_upstream_copy_allow_ranges(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);

#if (NGX_HTTP_GZIP)
static ngx_int_t ngx_http_upstream_copy_content_encoding(ngx_http_request_t *r,
     ngx_table_elt_t *h, ngx_uint_t offset);
#endif

static ngx_int_t ngx_http_upstream_add_variables(ngx_conf_t *cf);
static ngx_int_t ngx_http_upstream_addr_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_upstream_status_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_upstream_response_time_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
```

```
150 static ngx_int_t ngx_http_upstream_response_length_variable(
151     ngx_http_request_t *r, ngx_http_variable_value_t *v, uintptr_t data);
152
153 static char *ngx_http_upstream(ngx_conf_t *cf, ngx_command_t *cmd, void *dummy);
154 static char *ngx_http_upstream_server(ngx_conf_t *cf, ngx_command_t *cmd,
155     void *conf);
156
157 static ngx_addr_t *ngx_http_upstream_get_local(ngx_http_request_t *r,
158     ngx_http_upstream_local_t *local);
159
160 static void *ngx_http_upstream_create_main_conf(ngx_conf_t *cf);
161 static char *ngx_http_upstream_init_main_conf(ngx_conf_t *cf, void *conf);
162
163 #if (NGX_HTTP_SSL)
164 static void ngx_http_upstream_ssl_init_connection(ngx_http_request_t *,
165     ngx_http_upstream_t *u, ngx_connection_t *c);
166 static void ngx_http_upstream_ssl_handshake(ngx_connection_t *c);
167 static ngx_int_t ngx_http_upstream_ssl_name(ngx_http_request_t *r,
168     ngx_http_upstream_t *u, ngx_connection_t *c);
169 #endif
170
171
172 ngx_http_upstream_header_t  ngx_http_upstream_headers_in[] = {
173
174     { ngx_string("Status"),
175                 ngx_http_upstream_process_header_line,
176                 offsetof(ngx_http_upstream_headers_in_t, status),
177                 ngx_http_upstream_copy_header_line, 0, 0 },
178
179     { ngx_string("Content-Type"),
180                 ngx_http_upstream_process_header_line,
181                 offsetof(ngx_http_upstream_headers_in_t, content_type),
182                 ngx_http_upstream_copy_content_type, 0, 1 },
183
184     { ngx_string("Content-Length"),
185                 ngx_http_upstream_process_content_length, 0,
186                 ngx_http_upstream_ignore_header_line, 0, 0 },
187
188     { ngx_string("Date"),
189                 ngx_http_upstream_process_header_line,
190                 offsetof(ngx_http_upstream_headers_in_t, date),
191                 ngx_http_upstream_copy_header_line,
192                 offsetof(ngx_http_headers_out_t, date), 0 },
193
194     { ngx_string("Last-Modified"),
195                 ngx_http_upstream_process_last_modified, 0,
196                 ngx_http_upstream_copy_last_modified, 0, 0 },
197
198     { ngx_string("ETag"),
199                 ngx_http_upstream_process_header_line,
200                 offsetof(ngx_http_upstream_headers_in_t, etag),
201                 ngx_http_upstream_copy_header_line,
202                 offsetof(ngx_http_headers_out_t, etag), 0 },
203
204     { ngx_string("Server"),
205                 ngx_http_upstream_process_header_line,
206                 offsetof(ngx_http_upstream_headers_in_t, server),
207                 ngx_http_upstream_copy_header_line,
208                 offsetof(ngx_http_headers_out_t, server), 0 },
209
210     { ngx_string("WWW-Authenticate"),
211                 ngx_http_upstream_process_header_line,
212                 offsetof(ngx_http_upstream_headers_in_t, www_authenticate),
213                 ngx_http_upstream_copy_header_line, 0, 0 },
214
215     { ngx_string("Location"),
216                 ngx_http_upstream_process_header_line,
217                 offsetof(ngx_http_upstream_headers_in_t, location),
218                 ngx_http_upstream_rewrite_location, 0, 0 },
219
220     { ngx_string("Refresh"),
221                 ngx_http_upstream_ignore_header_line, 0,
222                 ngx_http_upstream_rewrite_refresh, 0, 0 },
223
224     { ngx_string("Set-Cookie"),
225                 ngx_http_upstream_process_set_cookie,
```

```
                  offsetof(ngx_http_upstream_headers_in_t, cookies),
                  ngx_http_upstream_rewrite_set_cookie, 0, 1 },

    { ngx_string("Content-Disposition"),
                  ngx_http_upstream_ignore_header_line, 0,
                  ngx_http_upstream_copy_header_line, 0, 1 },

    { ngx_string("Cache-Control"),
                  ngx_http_upstream_process_cache_control, 0,
                  ngx_http_upstream_copy_multi_header_lines,
                  offsetof(ngx_http_headers_out_t, cache_control), 1 },

    { ngx_string("Expires"),
                  ngx_http_upstream_process_expires, 0,
                  ngx_http_upstream_copy_header_line,
                  offsetof(ngx_http_headers_out_t, expires), 1 },

    { ngx_string("Accept-Ranges"),
                  ngx_http_upstream_process_header_line,
                  offsetof(ngx_http_upstream_headers_in_t, accept_ranges),
                  ngx_http_upstream_copy_allow_ranges,
                  offsetof(ngx_http_headers_out_t, accept_ranges), 1 },

    { ngx_string("Connection"),
                  ngx_http_upstream_process_connection, 0,
                  ngx_http_upstream_ignore_header_line, 0, 0 },

    { ngx_string("Keep-Alive"),
                  ngx_http_upstream_ignore_header_line, 0,
                  ngx_http_upstream_ignore_header_line, 0, 0 },

    { ngx_string("Vary"),
                  ngx_http_upstream_process_vary, 0,
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("X-Powered-By"),
                  ngx_http_upstream_ignore_header_line, 0,
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("X-Accel-Expires"),
                  ngx_http_upstream_process_accel_expires, 0,
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("X-Accel-Redirect"),
                  ngx_http_upstream_process_header_line,
                  offsetof(ngx_http_upstream_headers_in_t, x_accel_redirect),
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("X-Accel-Limit-Rate"),
                  ngx_http_upstream_process_limit_rate, 0,
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("X-Accel-Buffering"),
                  ngx_http_upstream_process_buffering, 0,
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("X-Accel-Charset"),
                  ngx_http_upstream_process_charset, 0,
                  ngx_http_upstream_copy_header_line, 0, 0 },

    { ngx_string("Transfer-Encoding"),
                  ngx_http_upstream_process_transfer_encoding, 0,
                  ngx_http_upstream_ignore_header_line, 0, 0 },

#if (NGX_HTTP_GZIP)
    { ngx_string("Content-Encoding"),
                  ngx_http_upstream_process_header_line,
                  offsetof(ngx_http_upstream_headers_in_t, content_encoding),
                  ngx_http_upstream_copy_content_encoding, 0, 0 },
#endif

    { ngx_null_string, NULL, 0, NULL, 0, 0 }
};


static ngx_command_t  ngx_http_upstream_commands[] = {
```

```
302
303     { ngx_string("upstream"),
304       NGX_HTTP_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_TAKE1,
305       ngx_http_upstream,
306       0,
307       0,
308       NULL },
309
310     { ngx_string("server"),
311       NGX_HTTP_UPS_CONF|NGX_CONF_1MORE,
312       ngx_http_upstream_server,
313       NGX_HTTP_SRV_CONF_OFFSET,
314       0,
315       NULL },
316
317       ngx_null_command
318 };
319
320
321 static ngx_http_module_t  ngx_http_upstream_module_ctx = {
322     ngx_http_upstream_add_variables,       /* preconfiguration */
323     NULL,                                  /* postconfiguration */
324
325     ngx_http_upstream_create_main_conf,    /* create main configuration */
326     ngx_http_upstream_init_main_conf,      /* init main configuration */
327
328     NULL,                                  /* create server configuration */
329     NULL,                                  /* merge server configuration */
330
331     NULL,                                  /* create location configuration */
332     NULL                                   /* merge location configuration */
333 };
334
335
336 ngx_module_t  ngx_http_upstream_module = {
337     NGX_MODULE_V1,
338     &ngx_http_upstream_module_ctx,         /* module context */
339     ngx_http_upstream_commands,            /* module directives */
340     NGX_HTTP_MODULE,                       /* module type */
341     NULL,                                  /* init master */
342     NULL,                                  /* init module */
343     NULL,                                  /* init process */
344     NULL,                                  /* init thread */
345     NULL,                                  /* exit thread */
346     NULL,                                  /* exit process */
347     NULL,                                  /* exit master */
348     NGX_MODULE_V1_PADDING
349 };
350
351
352 static ngx_http_variable_t  ngx_http_upstream_vars[] = {
353
354     { ngx_string("upstream_addr"), NULL,
355       ngx_http_upstream_addr_variable, 0,
356       NGX_HTTP_VAR_NOCACHEABLE, 0 },
357
358     { ngx_string("upstream_status"), NULL,
359       ngx_http_upstream_status_variable, 0,
360       NGX_HTTP_VAR_NOCACHEABLE, 0 },
361
362     { ngx_string("upstream_header_time"), NULL,
363       ngx_http_upstream_response_time_variable, 1,
364       NGX_HTTP_VAR_NOCACHEABLE, 0 },
365
366     { ngx_string("upstream_response_time"), NULL,
367       ngx_http_upstream_response_time_variable, 0,
368       NGX_HTTP_VAR_NOCACHEABLE, 0 },
369
370     { ngx_string("upstream_response_length"), NULL,
371       ngx_http_upstream_response_length_variable, 0,
372       NGX_HTTP_VAR_NOCACHEABLE, 0 },
373
374 #if (NGX_HTTP_CACHE)
375
376     { ngx_string("upstream_cache_status"), NULL,
377       ngx_http_upstream_cache_status, 0,
```

```
      NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("upstream_cache_last_modified"), NULL,
      ngx_http_upstream_cache_last_modified, 0,
      NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

    { ngx_string("upstream_cache_etag"), NULL,
      ngx_http_upstream_cache_etag, 0,
      NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

#endif

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static ngx_http_upstream_next_t  ngx_http_upstream_next_errors[] = {
    { 500, NGX_HTTP_UPSTREAM_FT_HTTP_500 },
    { 502, NGX_HTTP_UPSTREAM_FT_HTTP_502 },
    { 503, NGX_HTTP_UPSTREAM_FT_HTTP_503 },
    { 504, NGX_HTTP_UPSTREAM_FT_HTTP_504 },
    { 403, NGX_HTTP_UPSTREAM_FT_HTTP_403 },
    { 404, NGX_HTTP_UPSTREAM_FT_HTTP_404 },
    { 0, 0 }
};


ngx_conf_bitmask_t  ngx_http_upstream_cache_method_mask[] = {
   { ngx_string("GET"),  NGX_HTTP_GET},
   { ngx_string("HEAD"), NGX_HTTP_HEAD },
   { ngx_string("POST"), NGX_HTTP_POST },
   { ngx_null_string, 0 }
};


ngx_conf_bitmask_t  ngx_http_upstream_ignore_headers_masks[] = {
    { ngx_string("X-Accel-Redirect"), NGX_HTTP_UPSTREAM_IGN_XA_REDIRECT },
    { ngx_string("X-Accel-Expires"), NGX_HTTP_UPSTREAM_IGN_XA_EXPIRES },
    { ngx_string("X-Accel-Limit-Rate"), NGX_HTTP_UPSTREAM_IGN_XA_LIMIT_RATE },
    { ngx_string("X-Accel-Buffering"), NGX_HTTP_UPSTREAM_IGN_XA_BUFFERING },
    { ngx_string("X-Accel-Charset"), NGX_HTTP_UPSTREAM_IGN_XA_CHARSET },
    { ngx_string("Expires"), NGX_HTTP_UPSTREAM_IGN_EXPIRES },
    { ngx_string("Cache-Control"), NGX_HTTP_UPSTREAM_IGN_CACHE_CONTROL },
    { ngx_string("Set-Cookie"), NGX_HTTP_UPSTREAM_IGN_SET_COOKIE },
    { ngx_string("Vary"), NGX_HTTP_UPSTREAM_IGN_VARY },
    { ngx_null_string, 0 }
};


ngx_int_t
ngx_http_upstream_create(ngx_http_request_t *r)
{
    ngx_http_upstream_t  *u;

    u = r->upstream;

    if (u && u->cleanup) {
        r->main->count++;
        ngx_http_upstream_cleanup(r);
    }

    u = ngx_pcalloc(r->pool, sizeof(ngx_http_upstream_t));
    if (u == NULL) {
        return NGX_ERROR;
    }

    r->upstream = u;

    u->peer.log = r->connection->log;
    u->peer.log_error = NGX_ERROR_ERR;
#if (NGX_THREADS)
    u->peer.lock = &r->connection->lock;
#endif

#if (NGX_HTTP_CACHE)
    r->cache = NULL;
```

```
#endif

    u->headers_in.content_length_n = -1;
    u->headers_in.last_modified_time = -1;

    return NGX_OK;
}


void
ngx_http_upstream_init(ngx_http_request_t *r)
{
    ngx_connection_t     *c;

    c = r->connection;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http init upstream, client timer: %d", c->read->timer_set);

#if (NGX_HTTP_SPDY)
    if (r->spdy_stream) {
        ngx_http_upstream_init_request(r);
        return;
    }
#endif

    if (c->read->timer_set) {
        ngx_del_timer(c->read);
    }

    if (ngx_event_flags & NGX_USE_CLEAR_EVENT) {

        if (!c->write->active) {
            if (ngx_add_event(c->write, NGX_WRITE_EVENT, NGX_CLEAR_EVENT)
                == NGX_ERROR)
            {
                ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
                return;
            }
        }
    }

    ngx_http_upstream_init_request(r);
}


static void
ngx_http_upstream_init_request(ngx_http_request_t *r)
{
    ngx_str_t                      *host;
    ngx_uint_t                      i;
    ngx_resolver_ctx_t             *ctx, temp;
    ngx_http_cleanup_t             *cln;
    ngx_http_upstream_t            *u;
    ngx_http_core_loc_conf_t       *clcf;
    ngx_http_upstream_srv_conf_t   *uscf, **uscfp;
    ngx_http_upstream_main_conf_t  *umcf;

    if (r->aio) {
        return;
    }

    u = r->upstream;

#if (NGX_HTTP_CACHE)

    if (u->conf->cache) {
        ngx_int_t  rc;

        rc = ngx_http_upstream_cache(r, u);

        if (rc == NGX_BUSY) {
            r->write_event_handler = ngx_http_upstream_init_request;
            return;
        }
```

```
        r->write_event_handler = ngx_http_request_empty_handler;

        if (rc == NGX_DONE) {
            return;
        }

        if (rc == NGX_ERROR) {
            ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        if (rc != NGX_DECLINED) {
            ngx_http_finalize_request(r, rc);
            return;
        }
    }

#endif

    u->store = u->conf->store;

    if (!u->store && !r->post_action && !u->conf->ignore_client_abort) {
        r->read_event_handler = ngx_http_upstream_rd_check_broken_connection;
        r->write_event_handler = ngx_http_upstream_wr_check_broken_connection;
    }

    if (r->request_body) {
        u->request_bufs = r->request_body->bufs;
    }

    if (u->create_request(r) != NGX_OK) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    u->peer.local = ngx_http_upstream_get_local(r, u->conf->local);

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    u->output.alignment = clcf->directio_alignment;
    u->output.pool = r->pool;
    u->output.bufs.num = 1;
    u->output.bufs.size = clcf->client_body_buffer_size;
    u->output.output_filter = ngx_chain_writer;
    u->output.filter_ctx = &u->writer;

    u->writer.pool = r->pool;

    if (r->upstream_states == NULL) {

        r->upstream_states = ngx_array_create(r->pool, 1,
                                              sizeof(ngx_http_upstream_state_t));
        if (r->upstream_states == NULL) {
            ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

    } else {

        u->state = ngx_array_push(r->upstream_states);
        if (u->state == NULL) {
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        ngx_memzero(u->state, sizeof(ngx_http_upstream_state_t));
    }

    cln = ngx_http_cleanup_add(r, 0);
    if (cln == NULL) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    cln->handler = ngx_http_upstream_cleanup;
```

```
    cln->data = r;
    u->cleanup = &cln->handler;

    if (u->resolved == NULL) {

        uscf = u->conf->upstream;

    } else {

#if (NGX_HTTP_SSL)
        u->ssl_name = u->resolved->host;
#endif

        if (u->resolved->sockaddr) {

            if (ngx_http_upstream_create_round_robin_peer(r, u->resolved)
                != NGX_OK)
            {
                ngx_http_upstream_finalize_request(r, u,
                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
                return;
            }

            ngx_http_upstream_connect(r, u);

            return;
        }

        host = &u->resolved->host;

        umcf = ngx_http_get_module_main_conf(r, ngx_http_upstream_module);

        uscfp = umcf->upstreams.elts;

        for (i = 0; i < umcf->upstreams.nelts; i++) {

            uscf = uscfp[i];

            if (uscf->host.len == host->len
                && ((uscf->port == 0 && u->resolved->no_port)
                     || uscf->port == u->resolved->port)
                && ngx_strncasecmp(uscf->host.data, host->data, host->len) == 0)
            {
                goto found;
            }
        }

        if (u->resolved->port == 0) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "no port in upstream \"%V\"", host);
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        temp.name = *host;

        ctx = ngx_resolve_start(clcf->resolver, &temp);
        if (ctx == NULL) {
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        if (ctx == NGX_NO_RESOLVER) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "no resolver defined to resolve %V", host);

            ngx_http_upstream_finalize_request(r, u, NGX_HTTP_BAD_GATEWAY);
            return;
        }

        ctx->name = *host;
        ctx->handler = ngx_http_upstream_resolve_handler;
        ctx->data = r;
        ctx->timeout = clcf->resolver_timeout;
```

```
682
683         u->resolved->ctx = ctx;
684
685         if (ngx_resolve_name(ctx) != NGX_OK) {
686             u->resolved->ctx = NULL;
687             ngx_http_upstream_finalize_request(r, u,
688                                                NGX_HTTP_INTERNAL_SERVER_ERROR);
689             return;
690         }
691
692         return;
693     }
694
695 found:
696
697     if (uscf == NULL) {
698         ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
699                       "no upstream configuration");
700         ngx_http_upstream_finalize_request(r, u,
701                                            NGX_HTTP_INTERNAL_SERVER_ERROR);
702         return;
703     }
704
705 #if (NGX_HTTP_SSL)
706     u->ssl_name = uscf->host;
707 #endif
708
709     if (uscf->peer.init(r, uscf) != NGX_OK) {
710         ngx_http_upstream_finalize_request(r, u,
711                                            NGX_HTTP_INTERNAL_SERVER_ERROR);
712         return;
713     }
714
715     u->peer.start_time = ngx_current_msec;
716
717     if (u->conf->next_upstream_tries
718         && u->peer.tries > u->conf->next_upstream_tries)
719     {
720         u->peer.tries = u->conf->next_upstream_tries;
721     }
722
723     ngx_http_upstream_connect(r, u);
724 }
725
726
727 #if (NGX_HTTP_CACHE)
728
729 static ngx_int_t
730 ngx_http_upstream_cache(ngx_http_request_t *r, ngx_http_upstream_t *u)
731 {
732     ngx_int_t               rc;
733     ngx_http_cache_t       *c;
734     ngx_http_file_cache_t  *cache;
735
736     c = r->cache;
737
738     if (c == NULL) {
739
740         if (!(r->method & u->conf->cache_methods)) {
741             return NGX_DECLINED;
742         }
743
744         rc = ngx_http_upstream_cache_get(r, u, &cache);
745
746         if (rc != NGX_OK) {
747             return rc;
748         }
749
750         if (r->method & NGX_HTTP_HEAD) {
751             u->method = ngx_http_core_get_method;
752         }
753
754         if (ngx_http_file_cache_new(r) != NGX_OK) {
755             return NGX_ERROR;
756         }
757
```

```
        if (u->create_key(r) != NGX_OK) {
            return NGX_ERROR;
        }

        /* TODO: add keys */

        ngx_http_file_cache_create_key(r);

        if (r->cache->header_start + 256 >= u->conf->buffer_size) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "%V_buffer_size %uz is not enough for cache key, "
                          "it should be increased to at least %uz",
                          &u->conf->module, u->conf->buffer_size,
                          ngx_align(r->cache->header_start + 256, 1024));

            r->cache = NULL;
            return NGX_DECLINED;
        }

        u->cacheable = 1;

        c = r->cache;

        c->body_start = u->conf->buffer_size;
        c->min_uses = u->conf->cache_min_uses;
        c->file_cache = cache;

        switch (ngx_http_test_predicates(r, u->conf->cache_bypass)) {

        case NGX_ERROR:
            return NGX_ERROR;

        case NGX_DECLINED:
            u->cache_status = NGX_HTTP_CACHE_BYPASS;
            return NGX_DECLINED;

        default: /* NGX_OK */
            break;
        }

        c->lock = u->conf->cache_lock;
        c->lock_timeout = u->conf->cache_lock_timeout;
        c->lock_age = u->conf->cache_lock_age;

        u->cache_status = NGX_HTTP_CACHE_MISS;
    }

    rc = ngx_http_file_cache_open(r);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http upstream cache: %i", rc);

    switch (rc) {

    case NGX_HTTP_CACHE_UPDATING:

        if (u->conf->cache_use_stale & NGX_HTTP_UPSTREAM_FT_UPDATING) {
            u->cache_status = rc;
            rc = NGX_OK;

        } else {
            rc = NGX_HTTP_CACHE_STALE;
        }

        break;

    case NGX_OK:
        u->cache_status = NGX_HTTP_CACHE_HIT;
    }

    switch (rc) {

    case NGX_OK:

        rc = ngx_http_upstream_cache_send(r, u);

```

```
        if (rc != NGX_HTTP_UPSTREAM_INVALID_HEADER) {
            return rc;
        }

        break;

    case NGX_HTTP_CACHE_STALE:

        c->valid_sec = 0;
        u->buffer.start = NULL;
        u->cache_status = NGX_HTTP_CACHE_EXPIRED;

        break;

    case NGX_DECLINED:

        if ((size_t) (u->buffer.end - u->buffer.start) < u->conf->buffer_size) {
            u->buffer.start = NULL;

        } else {
            u->buffer.pos = u->buffer.start + c->header_start;
            u->buffer.last = u->buffer.pos;
        }

        break;

    case NGX_HTTP_CACHE_SCARCE:

        u->cacheable = 0;

        break;

    case NGX_AGAIN:

        return NGX_BUSY;

    case NGX_ERROR:

        return NGX_ERROR;

    default:

        /* cached NGX_HTTP_BAD_GATEWAY, NGX_HTTP_GATEWAY_TIME_OUT, etc. */

        u->cache_status = NGX_HTTP_CACHE_HIT;

        return rc;
    }

    r->cached = 0;

    return NGX_DECLINED;
}


static ngx_int_t
ngx_http_upstream_cache_get(ngx_http_request_t *r, ngx_http_upstream_t *u,
    ngx_http_file_cache_t **cache)
{
    ngx_str_t               *name, val;
    ngx_uint_t               i;
    ngx_http_file_cache_t  **caches;

    if (u->conf->cache_zone) {
        *cache = u->conf->cache_zone->data;
        return NGX_OK;
    }

    if (ngx_http_complex_value(r, u->conf->cache_value, &val) != NGX_OK) {
        return NGX_ERROR;
    }

    if (val.len == 0
        || (val.len == 3 && ngx_strncmp(val.data, "off", 3) == 0))
    {
        return NGX_DECLINED;
```

```
    }

    caches = u->caches->elts;

    for (i = 0; i < u->caches->nelts; i++) {
        name = &caches[i]->shm_zone->shm.name;

        if (name->len == val.len
            && ngx_strncmp(name->data, val.data, val.len) == 0)
        {
            *cache = caches[i];
            return NGX_OK;
        }
    }

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "cache \"%V\" not found", &val);

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_upstream_cache_send(ngx_http_request_t *r, ngx_http_upstream_t *u)
{
    ngx_int_t          rc;
    ngx_http_cache_t  *c;

    r->cached = 1;
    c = r->cache;

    if (c->header_start == c->body_start) {
        r->http_version = NGX_HTTP_VERSION_9;
        return ngx_http_cache_send(r);
    }

    /* TODO: cache stack */

    u->buffer = *c->buf;
    u->buffer.pos += c->header_start;

    ngx_memzero(&u->headers_in, sizeof(ngx_http_upstream_headers_in_t));
    u->headers_in.content_length_n = -1;
    u->headers_in.last_modified_time = -1;

    if (ngx_list_init(&u->headers_in.headers, r->pool, 8,
                      sizeof(ngx_table_elt_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    rc = u->process_header(r);

    if (rc == NGX_OK) {

        if (ngx_http_upstream_process_headers(r, u) != NGX_OK) {
            return NGX_DONE;
        }

        return ngx_http_cache_send(r);
    }

    if (rc == NGX_ERROR) {
        return NGX_ERROR;
    }

    /* rc == NGX_HTTP_UPSTREAM_INVALID_HEADER */

    /* TODO: delete file */

    return rc;
}

#endif
```

```
986
987 static void
988 ngx_http_upstream_resolve_handler(ngx_resolver_ctx_t *ctx)
989 {
990     ngx_connection_t              *c;
991     ngx_http_request_t            *r;
992     ngx_http_upstream_t           *u;
993     ngx_http_upstream_resolved_t  *ur;
994
995     r = ctx->data;
996     c = r->connection;
997
998     u = r->upstream;
999     ur = u->resolved;
1000
1001     ngx_http_set_log_request(c->log, r);
1002
1003     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
1004                    "http upstream resolve: \"%V?%V\"", &r->uri, &r->args);
1005
1006     if (ctx->state) {
1007         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1008                       "%V could not be resolved (%i: %s)",
1009                       &ctx->name, ctx->state,
1010                       ngx_resolver_strerror(ctx->state));
1011
1012         ngx_http_upstream_finalize_request(r, u, NGX_HTTP_BAD_GATEWAY);
1013         goto failed;
1014     }
1015
1016     ur->naddrs = ctx->naddrs;
1017     ur->addrs = ctx->addrs;
1018
1019 #if (NGX_DEBUG)
1020     {
1021     u_char      text[NGX_SOCKADDR_STRLEN];
1022     ngx_str_t   addr;
1023     ngx_uint_t  i;
1024
1025     addr.data = text;
1026
1027     for (i = 0; i < ctx->naddrs; i++) {
1028         addr.len = ngx_sock_ntop(ur->addrs[i].sockaddr, ur->addrs[i].socklen,
1029                                  text, NGX_SOCKADDR_STRLEN, 0);
1030
1031         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1032                        "name was resolved to %V", &addr);
1033     }
1034     }
1035 #endif
1036
1037     if (ngx_http_upstream_create_round_robin_peer(r, ur) != NGX_OK) {
1038         ngx_http_upstream_finalize_request(r, u,
1039                                            NGX_HTTP_INTERNAL_SERVER_ERROR);
1040         goto failed;
1041     }
1042
1043     ngx_resolve_name_done(ctx);
1044     ur->ctx = NULL;
1045
1046     u->peer.start_time = ngx_current_msec;
1047
1048     if (u->conf->next_upstream_tries
1049         && u->peer.tries > u->conf->next_upstream_tries)
1050     {
1051         u->peer.tries = u->conf->next_upstream_tries;
1052     }
1053
1054     ngx_http_upstream_connect(r, u);
1055
1056 failed:
1057
1058     ngx_http_run_posted_requests(c);
1059 }
1060
1061
```

```c
static void
ngx_http_upstream_handler(ngx_event_t *ev)
{
    ngx_connection_t     *c;
    ngx_http_request_t   *r;
    ngx_http_upstream_t  *u;

    c = ev->data;
    r = c->data;

    u = r->upstream;
    c = r->connection;

    ngx_http_set_log_request(c->log, r);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http upstream request: \"%V?%V\"", &r->uri, &r->args);

    if (ev->write) {
        u->write_event_handler(r, u);

    } else {
        u->read_event_handler(r, u);
    }

    ngx_http_run_posted_requests(c);
}


static void
ngx_http_upstream_rd_check_broken_connection(ngx_http_request_t *r)
{
    ngx_http_upstream_check_broken_connection(r, r->connection->read);
}


static void
ngx_http_upstream_wr_check_broken_connection(ngx_http_request_t *r)
{
    ngx_http_upstream_check_broken_connection(r, r->connection->write);
}


static void
ngx_http_upstream_check_broken_connection(ngx_http_request_t *r,
    ngx_event_t *ev)
{
    int                  n;
    char                 buf[1];
    ngx_err_t            err;
    ngx_int_t            event;
    ngx_connection_t     *c;
    ngx_http_upstream_t  *u;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, ev->log, 0,
                   "http upstream check client, write event:%d, \"%V\"",
                   ev->write, &r->uri);

    c = r->connection;
    u = r->upstream;

    if (c->error) {
        if ((ngx_event_flags & NGX_USE_LEVEL_EVENT) && ev->active) {

            event = ev->write ? NGX_WRITE_EVENT : NGX_READ_EVENT;

            if (ngx_del_event(ev, event, 0) != NGX_OK) {
                ngx_http_upstream_finalize_request(r, u,
                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
                return;
            }
        }

        if (!u->cacheable) {
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
```

```
1138        }
1139
1140        return;
1141    }
1142
1143 #if (NGX_HTTP_SPDY)
1144    if (r->spdy_stream) {
1145        return;
1146    }
1147 #endif
1148
1149 #if (NGX_HAVE_KQUEUE)
1150
1151    if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
1152
1153        if (!ev->pending_eof) {
1154            return;
1155        }
1156
1157        ev->eof = 1;
1158        c->error = 1;
1159
1160        if (ev->kq_errno) {
1161            ev->error = 1;
1162        }
1163
1164        if (!u->cacheable && u->peer.connection) {
1165            ngx_log_error(NGX_LOG_INFO, ev->log, ev->kq_errno,
1166                          "kevent() reported that client prematurely closed "
1167                          "connection, so upstream connection is closed too");
1168            ngx_http_upstream_finalize_request(r, u,
1169                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
1170            return;
1171        }
1172
1173        ngx_log_error(NGX_LOG_INFO, ev->log, ev->kq_errno,
1174                      "kevent() reported that client prematurely closed "
1175                      "connection");
1176
1177        if (u->peer.connection == NULL) {
1178            ngx_http_upstream_finalize_request(r, u,
1179                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
1180        }
1181
1182        return;
1183    }
1184
1185 #endif
1186
1187 #if (NGX_HAVE_EPOLLRDHUP)
1188
1189    if ((ngx_event_flags & NGX_USE_EPOLL_EVENT) && ev->pending_eof) {
1190        socklen_t  len;
1191
1192        ev->eof = 1;
1193        c->error = 1;
1194
1195        err = 0;
1196        len = sizeof(ngx_err_t);
1197
1198        /*
1199         * BSDs and Linux return 0 and set a pending error in err
1200         * Solaris returns -1 and sets errno
1201         */
1202
1203        if (getsockopt(c->fd, SOL_SOCKET, SO_ERROR, (void *) &err, &len)
1204            == -1)
1205        {
1206            err = ngx_socket_errno;
1207        }
1208
1209        if (err) {
1210            ev->error = 1;
1211        }
1212
1213        if (!u->cacheable && u->peer.connection) {
```

```
            ngx_log_error(NGX_LOG_INFO, ev->log, err,
                          "epoll_wait() reported that client prematurely closed "
                          "connection, so upstream connection is closed too");
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
            return;
        }

        ngx_log_error(NGX_LOG_INFO, ev->log, err,
                      "epoll_wait() reported that client prematurely closed "
                      "connection");

        if (u->peer.connection == NULL) {
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
        }

        return;
    }

#endif

    n = recv(c->fd, buf, 1, MSG_PEEK);

    err = ngx_socket_errno;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ev->log, err,
                   "http upstream recv(): %d", n);

    if (ev->write && (n >= 0 || err == NGX_EAGAIN)) {
        return;
    }

    if ((ngx_event_flags & NGX_USE_LEVEL_EVENT) && ev->active) {

        event = ev->write ? NGX_WRITE_EVENT : NGX_READ_EVENT;

        if (ngx_del_event(ev, event, 0) != NGX_OK) {
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }
    }

    if (n > 0) {
        return;
    }

    if (n == -1) {
        if (err == NGX_EAGAIN) {
            return;
        }

        ev->error = 1;

    } else { /* n == 0 */
        err = 0;
    }

    ev->eof = 1;
    c->error = 1;

    if (!u->cacheable && u->peer.connection) {
        ngx_log_error(NGX_LOG_INFO, ev->log, err,
                      "client prematurely closed connection, "
                      "so upstream connection is closed too");
        ngx_http_upstream_finalize_request(r, u,
                                           NGX_HTTP_CLIENT_CLOSED_REQUEST);
        return;
    }

    ngx_log_error(NGX_LOG_INFO, ev->log, err,
                  "client prematurely closed connection");

    if (u->peer.connection == NULL) {
        ngx_http_upstream_finalize_request(r, u,
```

```
                                          NGX_HTTP_CLIENT_CLOSED_REQUEST);
    }
}


static void
ngx_http_upstream_connect(ngx_http_request_t *r, ngx_http_upstream_t *u)
{
    ngx_int_t          rc;
    ngx_time_t        *tp;
    ngx_connection_t  *c;

    r->connection->log->action = "connecting to upstream";

    if (u->state && u->state->response_sec) {
        tp = ngx_timeofday();
        u->state->response_sec = tp->sec - u->state->response_sec;
        u->state->response_msec = tp->msec - u->state->response_msec;
    }

    u->state = ngx_array_push(r->upstream_states);
    if (u->state == NULL) {
        ngx_http_upstream_finalize_request(r, u,
                                           NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    ngx_memzero(u->state, sizeof(ngx_http_upstream_state_t));

    tp = ngx_timeofday();
    u->state->response_sec = tp->sec;
    u->state->response_msec = tp->msec;
    u->state->header_sec = (time_t) NGX_ERROR;

    rc = ngx_event_connect_peer(&u->peer);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http upstream connect: %i", rc);

    if (rc == NGX_ERROR) {
        ngx_http_upstream_finalize_request(r, u,
                                           NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    u->state->peer = u->peer.name;

    if (rc == NGX_BUSY) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0, "no live upstreams");
        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_NOLIVE);
        return;
    }

    if (rc == NGX_DECLINED) {
        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);
        return;
    }

    /* rc == NGX_OK || rc == NGX_AGAIN || rc == NGX_DONE */

    c = u->peer.connection;

    c->data = r;

    c->write->handler = ngx_http_upstream_handler;
    c->read->handler = ngx_http_upstream_handler;

    u->write_event_handler = ngx_http_upstream_send_request_handler;
    u->read_event_handler = ngx_http_upstream_process_header;

    c->sendfile &= r->connection->sendfile;
    u->output.sendfile = c->sendfile;

    if (c->pool == NULL) {

        /* we need separate pool here to be able to cache SSL connections */
```

```
1366
1367            c->pool = ngx_create_pool(128, r->connection->log);
1368            if (c->pool == NULL) {
1369                ngx_http_upstream_finalize_request(r, u,
1370                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
1371                return;
1372            }
1373        }
1374
1375        c->log = r->connection->log;
1376        c->pool->log = c->log;
1377        c->read->log = c->log;
1378        c->write->log = c->log;
1379
1380        /* init or reinit the ngx_output_chain() and ngx_chain_writer() contexts */
1381
1382        u->writer.out = NULL;
1383        u->writer.last = &u->writer.out;
1384        u->writer.connection = c;
1385        u->writer.limit = 0;
1386
1387        if (u->request_sent) {
1388            if (ngx_http_upstream_reinit(r, u) != NGX_OK) {
1389                ngx_http_upstream_finalize_request(r, u,
1390                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
1391                return;
1392            }
1393        }
1394
1395        if (r->request_body
1396            && r->request_body->buf
1397            && r->request_body->temp_file
1398            && r == r->main)
1399        {
1400            /*
1401             * the r->request_body->buf can be reused for one request only,
1402             * the subrequests should allocate their own temporary bufs
1403             */
1404
1405            u->output.free = ngx_alloc_chain_link(r->pool);
1406            if (u->output.free == NULL) {
1407                ngx_http_upstream_finalize_request(r, u,
1408                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
1409                return;
1410            }
1411
1412            u->output.free->buf = r->request_body->buf;
1413            u->output.free->next = NULL;
1414            u->output.allocated = 1;
1415
1416            r->request_body->buf->pos = r->request_body->buf->start;
1417            r->request_body->buf->last = r->request_body->buf->start;
1418            r->request_body->buf->tag = u->output.tag;
1419        }
1420
1421        u->request_sent = 0;
1422
1423        if (rc == NGX_AGAIN) {
1424            ngx_add_timer(c->write, u->conf->connect_timeout);
1425            return;
1426        }
1427
1428    #if (NGX_HTTP_SSL)
1429
1430        if (u->ssl && c->ssl == NULL) {
1431            ngx_http_upstream_ssl_init_connection(r, u, c);
1432            return;
1433        }
1434
1435    #endif
1436
1437        ngx_http_upstream_send_request(r, u);
1438    }
1439
1440
1441    #if (NGX_HTTP_SSL)
```

```
1442
1443 static void
1444 ngx_http_upstream_ssl_init_connection(ngx_http_request_t *r,
1445     ngx_http_upstream_t *u, ngx_connection_t *c)
1446 {
1447     ngx_int_t   rc;
1448
1449     if (ngx_http_upstream_test_connect(c) != NGX_OK) {
1450         ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);
1451         return;
1452     }
1453
1454     if (ngx_ssl_create_connection(u->conf->ssl, c,
1455                                   NGX_SSL_BUFFER|NGX_SSL_CLIENT)
1456         != NGX_OK)
1457     {
1458         ngx_http_upstream_finalize_request(r, u,
1459                                            NGX_HTTP_INTERNAL_SERVER_ERROR);
1460         return;
1461     }
1462
1463     c->sendfile = 0;
1464     u->output.sendfile = 0;
1465
1466     if (u->conf->ssl_server_name || u->conf->ssl_verify) {
1467         if (ngx_http_upstream_ssl_name(r, u, c) != NGX_OK) {
1468             ngx_http_upstream_finalize_request(r, u,
1469                                                NGX_HTTP_INTERNAL_SERVER_ERROR);
1470             return;
1471         }
1472     }
1473
1474     if (u->conf->ssl_session_reuse) {
1475         if (u->peer.set_session(&u->peer, u->peer.data) != NGX_OK) {
1476             ngx_http_upstream_finalize_request(r, u,
1477                                                NGX_HTTP_INTERNAL_SERVER_ERROR);
1478             return;
1479         }
1480     }
1481
1482     r->connection->log->action = "SSL handshaking to upstream";
1483
1484     rc = ngx_ssl_handshake(c);
1485
1486     if (rc == NGX_AGAIN) {
1487
1488         if (!c->write->timer_set) {
1489             ngx_add_timer(c->write, u->conf->connect_timeout);
1490         }
1491
1492         c->ssl->handler = ngx_http_upstream_ssl_handshake;
1493         return;
1494     }
1495
1496     ngx_http_upstream_ssl_handshake(c);
1497 }
1498
1499
1500 static void
1501 ngx_http_upstream_ssl_handshake(ngx_connection_t *c)
1502 {
1503     long                 rc;
1504     ngx_http_request_t   *r;
1505     ngx_http_upstream_t  *u;
1506
1507     r = c->data;
1508     u = r->upstream;
1509
1510     ngx_http_set_log_request(c->log, r);
1511
1512     if (c->ssl->handshaked) {
1513
1514         if (u->conf->ssl_verify) {
1515             rc = SSL_get_verify_result(c->ssl->connection);
1516
1517             if (rc != X509_V_OK) {
```

```
                ngx_log_error(NGX_LOG_ERR, c->log, 0,
                              "upstream SSL certificate verify error: (%l:%s)",
                              rc, X509_verify_cert_error_string(rc));
                goto failed;
            }

            if (ngx_ssl_check_host(c, &u->ssl_name) != NGX_OK) {
                ngx_log_error(NGX_LOG_ERR, c->log, 0,
                              "upstream SSL certificate does not match \"%V\"",
                              &u->ssl_name);
                goto failed;
            }
        }

        if (u->conf->ssl_session_reuse) {
            u->peer.save_session(&u->peer, u->peer.data);
        }

        c->write->handler = ngx_http_upstream_handler;
        c->read->handler = ngx_http_upstream_handler;

        c = r->connection;

        ngx_http_upstream_send_request(r, u);

        ngx_http_run_posted_requests(c);
        return;
    }

failed:

    c = r->connection;

    ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);

    ngx_http_run_posted_requests(c);
}


static ngx_int_t
ngx_http_upstream_ssl_name(ngx_http_request_t *r, ngx_http_upstream_t *u,
    ngx_connection_t *c)
{
    u_char     *p, *last;
    ngx_str_t   name;

    if (u->conf->ssl_name) {
        if (ngx_http_complex_value(r, u->conf->ssl_name, &name) != NGX_OK) {
            return NGX_ERROR;
        }

    } else {
        name = u->ssl_name;
    }

    if (name.len == 0) {
        goto done;
    }

    /*
     * ssl name here may contain port, notably if derived from $proxy_host
     * or $http_host; we have to strip it
     */

    p = name.data;
    last = name.data + name.len;

    if (*p == '[') {
        p = ngx_strlchr(p, last, ']');

        if (p == NULL) {
            p = name.data;
        }
    }

    p = ngx_strlchr(p, last, ':');
```

```
1594
1595        if (p != NULL) {
1596            name.len = p - name.data;
1597        }
1598
1599        if (!u->conf->ssl_server_name) {
1600            goto done;
1601        }
1602
1603    #ifdef SSL_CTRL_SET_TLSEXT_HOSTNAME
1604
1605        /* as per RFC 6066, literal IPv4 and IPv6 addresses are not permitted */
1606
1607        if (name.len == 0 || *name.data == '[') {
1608            goto done;
1609        }
1610
1611        if (ngx_inet_addr(name.data, name.len) != INADDR_NONE) {
1612            goto done;
1613        }
1614
1615        /*
1616         * SSL_set_tlsext_host_name() needs a null-terminated string,
1617         * hence we explicitly null-terminate name here
1618         */
1619
1620        p = ngx_pnalloc(r->pool, name.len + 1);
1621        if (p == NULL) {
1622            return NGX_ERROR;
1623        }
1624
1625        (void) ngx_cpystrn(p, name.data, name.len + 1);
1626
1627        name.data = p;
1628
1629        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1630                       "upstream SSL server name: \"%s\"", name.data);
1631
1632        if (SSL_set_tlsext_host_name(c->ssl->connection, name.data) == 0) {
1633            ngx_ssl_error(NGX_LOG_ERR, r->connection->log, 0,
1634                          "SSL_set_tlsext_host_name(\"%s\") failed", name.data);
1635            return NGX_ERROR;
1636        }
1637
1638    #endif
1639
1640    done:
1641
1642        u->ssl_name = name;
1643
1644        return NGX_OK;
1645    }
1646
1647    #endif
1648
1649
1650    static ngx_int_t
1651    ngx_http_upstream_reinit(ngx_http_request_t *r, ngx_http_upstream_t *u)
1652    {
1653        off_t         file_pos;
1654        ngx_chain_t  *cl;
1655
1656        if (u->reinit_request(r) != NGX_OK) {
1657            return NGX_ERROR;
1658        }
1659
1660        u->keepalive = 0;
1661        u->upgrade = 0;
1662
1663        ngx_memzero(&u->headers_in, sizeof(ngx_http_upstream_headers_in_t));
1664        u->headers_in.content_length_n = -1;
1665        u->headers_in.last_modified_time = -1;
1666
1667        if (ngx_list_init(&u->headers_in.headers, r->pool, 8,
1668                          sizeof(ngx_table_elt_t))
1669            != NGX_OK)
```

```
    {
        return NGX_ERROR;
    }

    /* reinit the request chain */

    file_pos = 0;

    for (cl = u->request_bufs; cl; cl = cl->next) {
        cl->buf->pos = cl->buf->start;

        /* there is at most one file */

        if (cl->buf->in_file) {
            cl->buf->file_pos = file_pos;
            file_pos = cl->buf->file_last;
        }
    }

    /* reinit the subrequest's ngx_output_chain() context */

    if (r->request_body && r->request_body->temp_file
        && r != r->main && u->output.buf)
    {
        u->output.free = ngx_alloc_chain_link(r->pool);
        if (u->output.free == NULL) {
            return NGX_ERROR;
        }

        u->output.free->buf = u->output.buf;
        u->output.free->next = NULL;

        u->output.buf->pos = u->output.buf->start;
        u->output.buf->last = u->output.buf->start;
    }

    u->output.buf = NULL;
    u->output.in = NULL;
    u->output.busy = NULL;

    /* reinit u->buffer */

    u->buffer.pos = u->buffer.start;

#if (NGX_HTTP_CACHE)

    if (r->cache) {
        u->buffer.pos += r->cache->header_start;
    }

#endif

    u->buffer.last = u->buffer.pos;

    return NGX_OK;
}


static void
ngx_http_upstream_send_request(ngx_http_request_t *r, ngx_http_upstream_t *u)
{
    ngx_int_t          rc;
    ngx_connection_t  *c;

    c = u->peer.connection;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http upstream send request");

    if (!u->request_sent && ngx_http_upstream_test_connect(c) != NGX_OK) {
        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);
        return;
    }

    c->log->action = "sending request to upstream";

```

```
    rc = ngx_output_chain(&u->output, u->request_sent ? NULL : u->request_bufs);

    u->request_sent = 1;

    if (rc == NGX_ERROR) {
        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);
        return;
    }

    if (c->write->timer_set) {
        ngx_del_timer(c->write);
    }

    if (rc == NGX_AGAIN) {
        ngx_add_timer(c->write, u->conf->send_timeout);

        if (ngx_handle_write_event(c->write, u->conf->send_lowat) != NGX_OK) {
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        return;
    }

    /* rc == NGX_OK */

    if (c->tcp_nopush == NGX_TCP_NOPUSH_SET) {
        if (ngx_tcp_push(c->fd) == NGX_ERROR) {
            ngx_log_error(NGX_LOG_CRIT, c->log, ngx_socket_errno,
                          ngx_tcp_push_n " failed");
            ngx_http_upstream_finalize_request(r, u,
                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        c->tcp_nopush = NGX_TCP_NOPUSH_UNSET;
    }

    u->write_event_handler = ngx_http_upstream_dummy_handler;

    if (ngx_handle_write_event(c->write, 0) != NGX_OK) {
        ngx_http_upstream_finalize_request(r, u,
                                           NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    ngx_add_timer(c->read, u->conf->read_timeout);

    if (c->read->ready) {
        ngx_http_upstream_process_header(r, u);
        return;
    }
}


static void
ngx_http_upstream_send_request_handler(ngx_http_request_t *r,
    ngx_http_upstream_t *u)
{
    ngx_connection_t  *c;

    c = u->peer.connection;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http upstream send request handler");

    if (c->write->timedout) {
        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_TIMEOUT);
        return;
    }

#if (NGX_HTTP_SSL)

    if (u->ssl && c->ssl == NULL) {
        ngx_http_upstream_ssl_init_connection(r, u, c);
```

```
1822        return;
1823    }
1824
1825 #endif
1826
1827    if (u->header_sent) {
1828        u->write_event_handler = ngx_http_upstream_dummy_handler;
1829
1830        (void) ngx_handle_write_event(c->write, 0);
1831
1832        return;
1833    }
1834
1835    ngx_http_upstream_send_request(r, u);
1836 }
1837
1838
1839 static void
1840 ngx_http_upstream_process_header(ngx_http_request_t *r, ngx_http_upstream_t *u)
1841 {
1842    ssize_t            n;
1843    ngx_int_t          rc;
1844    ngx_time_t        *tp;
1845    ngx_connection_t  *c;
1846
1847    c = u->peer.connection;
1848
1849    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
1850                   "http upstream process header");
1851
1852    c->log->action = "reading response header from upstream";
1853
1854    if (c->read->timedout) {
1855        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_TIMEOUT);
1856        return;
1857    }
1858
1859    if (!u->request_sent && ngx_http_upstream_test_connect(c) != NGX_OK) {
1860        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);
1861        return;
1862    }
1863
1864    if (u->buffer.start == NULL) {
1865        u->buffer.start = ngx_palloc(r->pool, u->conf->buffer_size);
1866        if (u->buffer.start == NULL) {
1867            ngx_http_upstream_finalize_request(r, u,
1868                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
1869            return;
1870        }
1871
1872        u->buffer.pos = u->buffer.start;
1873        u->buffer.last = u->buffer.start;
1874        u->buffer.end = u->buffer.start + u->conf->buffer_size;
1875        u->buffer.temporary = 1;
1876
1877        u->buffer.tag = u->output.tag;
1878
1879        if (ngx_list_init(&u->headers_in.headers, r->pool, 8,
1880                          sizeof(ngx_table_elt_t))
1881            != NGX_OK)
1882        {
1883            ngx_http_upstream_finalize_request(r, u,
1884                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
1885            return;
1886        }
1887
1888 #if (NGX_HTTP_CACHE)
1889
1890        if (r->cache) {
1891            u->buffer.pos += r->cache->header_start;
1892            u->buffer.last = u->buffer.pos;
1893        }
1894 #endif
1895    }
1896
1897    for ( ;; ) {
```

```
1898
1899        n = c->recv(c, u->buffer.last, u->buffer.end - u->buffer.last);
1900
1901        if (n == NGX_AGAIN) {
1902    #if 0
1903            ngx_add_timer(rev, u->read_timeout);
1904    #endif
1905
1906            if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
1907                ngx_http_upstream_finalize_request(r, u,
1908                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
1909                return;
1910            }
1911
1912            return;
1913        }
1914
1915        if (n == 0) {
1916            ngx_log_error(NGX_LOG_ERR, c->log, 0,
1917                          "upstream prematurely closed connection");
1918        }
1919
1920        if (n == NGX_ERROR || n == 0) {
1921            ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_ERROR);
1922            return;
1923        }
1924
1925        u->buffer.last += n;
1926
1927    #if 0
1928        u->valid_header_in = 0;
1929
1930        u->peer.cached = 0;
1931    #endif
1932
1933        rc = u->process_header(r);
1934
1935        if (rc == NGX_AGAIN) {
1936
1937            if (u->buffer.last == u->buffer.end) {
1938                ngx_log_error(NGX_LOG_ERR, c->log, 0,
1939                              "upstream sent too big header");
1940
1941                ngx_http_upstream_next(r, u,
1942                                       NGX_HTTP_UPSTREAM_FT_INVALID_HEADER);
1943                return;
1944            }
1945
1946            continue;
1947        }
1948
1949        break;
1950    }
1951
1952    if (rc == NGX_HTTP_UPSTREAM_INVALID_HEADER) {
1953        ngx_http_upstream_next(r, u, NGX_HTTP_UPSTREAM_FT_INVALID_HEADER);
1954        return;
1955    }
1956
1957    if (rc == NGX_ERROR) {
1958        ngx_http_upstream_finalize_request(r, u,
1959                                           NGX_HTTP_INTERNAL_SERVER_ERROR);
1960        return;
1961    }
1962
1963    /* rc == NGX_OK */
1964
1965    tp = ngx_timeofday();
1966    u->state->header_sec = tp->sec - u->state->response_sec;
1967    u->state->header_msec = tp->msec - u->state->response_msec;
1968
1969    if (u->headers_in.status_n >= NGX_HTTP_SPECIAL_RESPONSE) {
1970
1971        if (ngx_http_upstream_test_next(r, u) == NGX_OK) {
1972            return;
1973        }
```

```
1974
1975            if (ngx_http_upstream_intercept_errors(r, u) == NGX_OK) {
1976                return;
1977            }
1978        }
1979
1980        if (ngx_http_upstream_process_headers(r, u) != NGX_OK) {
1981            return;
1982        }
1983
1984        if (!r->subrequest_in_memory) {
1985            ngx_http_upstream_send_response(r, u);
1986            return;
1987        }
1988
1989        /* subrequest content in memory */
1990
1991        if (u->input_filter == NULL) {
1992            u->input_filter_init = ngx_http_upstream_non_buffered_filter_init;
1993            u->input_filter = ngx_http_upstream_non_buffered_filter;
1994            u->input_filter_ctx = r;
1995        }
1996
1997        if (u->input_filter_init(u->input_filter_ctx) == NGX_ERROR) {
1998            ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
1999            return;
2000        }
2001
2002        n = u->buffer.last - u->buffer.pos;
2003
2004        if (n) {
2005            u->buffer.last = u->buffer.pos;
2006
2007            u->state->response_length += n;
2008
2009            if (u->input_filter(u->input_filter_ctx, n) == NGX_ERROR) {
2010                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2011                return;
2012            }
2013        }
2014
2015        if (u->length == 0) {
2016            ngx_http_upstream_finalize_request(r, u, 0);
2017            return;
2018        }
2019
2020        u->read_event_handler = ngx_http_upstream_process_body_in_memory;
2021
2022        ngx_http_upstream_process_body_in_memory(r, u);
2023    }
2024
2025
2026    static ngx_int_t
2027    ngx_http_upstream_test_next(ngx_http_request_t *r, ngx_http_upstream_t *u)
2028    {
2029        ngx_uint_t                 status;
2030        ngx_http_upstream_next_t  *un;
2031
2032        status = u->headers_in.status_n;
2033
2034        for (un = ngx_http_upstream_next_errors; un->status; un++) {
2035
2036            if (status != un->status) {
2037                continue;
2038            }
2039
2040            if (u->peer.tries > 1 && (u->conf->next_upstream & un->mask)) {
2041                ngx_http_upstream_next(r, u, un->mask);
2042                return NGX_OK;
2043            }
2044
2045    #if (NGX_HTTP_CACHE)
2046
2047            if (u->cache_status == NGX_HTTP_CACHE_EXPIRED
2048                && (u->conf->cache_use_stale & un->mask))
2049            {
```

```
2050            ngx_int_t  rc;
2051
2052            rc = u->reinit_request(r);
2053
2054            if (rc == NGX_OK) {
2055                u->cache_status = NGX_HTTP_CACHE_STALE;
2056                rc = ngx_http_upstream_cache_send(r, u);
2057            }
2058
2059            ngx_http_upstream_finalize_request(r, u, rc);
2060            return NGX_OK;
2061        }
2062
2063 #endif
2064    }
2065
2066 #if (NGX_HTTP_CACHE)
2067
2068    if (status == NGX_HTTP_NOT_MODIFIED
2069        && u->cache_status == NGX_HTTP_CACHE_EXPIRED
2070        && u->conf->cache_revalidate)
2071    {
2072        time_t     now, valid;
2073        ngx_int_t  rc;
2074
2075        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2076                       "http upstream not modified");
2077
2078        now = ngx_time();
2079        valid = r->cache->valid_sec;
2080
2081        rc = u->reinit_request(r);
2082
2083        if (rc != NGX_OK) {
2084            ngx_http_upstream_finalize_request(r, u, rc);
2085            return NGX_OK;
2086        }
2087
2088        u->cache_status = NGX_HTTP_CACHE_REVALIDATED;
2089        rc = ngx_http_upstream_cache_send(r, u);
2090
2091        if (valid == 0) {
2092            valid = r->cache->valid_sec;
2093        }
2094
2095        if (valid == 0) {
2096            valid = ngx_http_file_cache_valid(u->conf->cache_valid,
2097                                              u->headers_in.status_n);
2098            if (valid) {
2099                valid = now + valid;
2100            }
2101        }
2102
2103        if (valid) {
2104            r->cache->valid_sec = valid;
2105            r->cache->date = now;
2106
2107            ngx_http_file_cache_update_header(r);
2108        }
2109
2110        ngx_http_upstream_finalize_request(r, u, rc);
2111        return NGX_OK;
2112    }
2113
2114 #endif
2115
2116    return NGX_DECLINED;
2117 }
2118
2119
2120 static ngx_int_t
2121 ngx_http_upstream_intercept_errors(ngx_http_request_t *r,
2122     ngx_http_upstream_t *u)
2123 {
2124     ngx_int_t                  status;
2125     ngx_uint_t                 i;
```

```
    ngx_table_elt_t           *h;
    ngx_http_err_page_t       *err_page;
    ngx_http_core_loc_conf_t  *clcf;

    status = u->headers_in.status_n;

    if (status == NGX_HTTP_NOT_FOUND && u->conf->intercept_404) {
        ngx_http_upstream_finalize_request(r, u, NGX_HTTP_NOT_FOUND);
        return NGX_OK;
    }

    if (!u->conf->intercept_errors) {
        return NGX_DECLINED;
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (clcf->error_pages == NULL) {
        return NGX_DECLINED;
    }

    err_page = clcf->error_pages->elts;
    for (i = 0; i < clcf->error_pages->nelts; i++) {

        if (err_page[i].status == status) {

            if (status == NGX_HTTP_UNAUTHORIZED
                && u->headers_in.www_authenticate)
            {
                h = ngx_list_push(&r->headers_out.headers);

                if (h == NULL) {
                    ngx_http_upstream_finalize_request(r, u,
                                                       NGX_HTTP_INTERNAL_SERVER_ERROR);
                    return NGX_OK;
                }

                *h = *u->headers_in.www_authenticate;

                r->headers_out.www_authenticate = h;
            }

#if (NGX_HTTP_CACHE)

            if (r->cache) {
                time_t  valid;

                valid = ngx_http_file_cache_valid(u->conf->cache_valid, status);

                if (valid) {
                    r->cache->valid_sec = ngx_time() + valid;
                    r->cache->error = status;
                }

                ngx_http_file_cache_free(r->cache, u->pipe->temp_file);
            }
#endif
            ngx_http_upstream_finalize_request(r, u, status);

            return NGX_OK;
        }
    }

    return NGX_DECLINED;
}


static ngx_int_t
ngx_http_upstream_test_connect(ngx_connection_t *c)
{
    int        err;
    socklen_t  len;

#if (NGX_HAVE_KQUEUE)

    if (ngx_event_flags & NGX_USE_KQUEUE_EVENT)  {
```

```
2202         if (c->write->pending_eof || c->read->pending_eof) {
2203             if (c->write->pending_eof) {
2204                 err = c->write->kq_errno;
2205
2206             } else {
2207                 err = c->read->kq_errno;
2208             }
2209
2210             c->log->action = "connecting to upstream";
2211             (void) ngx_connection_error(c, err,
2212                                         "kevent() reported that connect() failed");
2213             return NGX_ERROR;
2214         }
2215
2216     } else
2217 #endif
2218     {
2219         err = 0;
2220         len = sizeof(int);
2221
2222         /*
2223          * BSDs and Linux return 0 and set a pending error in err
2224          * Solaris returns -1 and sets errno
2225          */
2226
2227         if (getsockopt(c->fd, SOL_SOCKET, SO_ERROR, (void *) &err, &len)
2228             == -1)
2229         {
2230             err = ngx_socket_errno;
2231         }
2232
2233         if (err) {
2234             c->log->action = "connecting to upstream";
2235             (void) ngx_connection_error(c, err, "connect() failed");
2236             return NGX_ERROR;
2237         }
2238     }
2239
2240     return NGX_OK;
2241 }
2242
2243
2244 static ngx_int_t
2245 ngx_http_upstream_process_headers(ngx_http_request_t *r, ngx_http_upstream_t *u)
2246 {
2247     ngx_str_t                      uri, args;
2248     ngx_uint_t                     i, flags;
2249     ngx_list_part_t               *part;
2250     ngx_table_elt_t               *h;
2251     ngx_http_upstream_header_t    *hh;
2252     ngx_http_upstream_main_conf_t  *umcf;
2253
2254     umcf = ngx_http_get_module_main_conf(r, ngx_http_upstream_module);
2255
2256     if (u->headers_in.x_accel_redirect
2257         && !(u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_XA_REDIRECT))
2258     {
2259         ngx_http_upstream_finalize_request(r, u, NGX_DECLINED);
2260
2261         part = &u->headers_in.headers.part;
2262         h = part->elts;
2263
2264         for (i = 0; /* void */; i++) {
2265
2266             if (i >= part->nelts) {
2267                 if (part->next == NULL) {
2268                     break;
2269                 }
2270
2271                 part = part->next;
2272                 h = part->elts;
2273                 i = 0;
2274             }
2275
2276             hh = ngx_hash_find(&umcf->headers_in_hash, h[i].hash,
2277                                h[i].lowcase_key, h[i].key.len);
```

```
2278
2279            if (hh && hh->redirect) {
2280                if (hh->copy_handler(r, &h[i], hh->conf) != NGX_OK) {
2281                    ngx_http_finalize_request(r,
2282                                              NGX_HTTP_INTERNAL_SERVER_ERROR);
2283                    return NGX_DONE;
2284                }
2285            }
2286        }
2287
2288        uri = u->headers_in.x_accel_redirect->value;
2289
2290        if (uri.data[0] == '@') {
2291            ngx_http_named_location(r, &uri);
2292
2293        } else {
2294            ngx_str_null(&args);
2295            flags = NGX_HTTP_LOG_UNSAFE;
2296
2297            if (ngx_http_parse_unsafe_uri(r, &uri, &args, &flags) != NGX_OK) {
2298                ngx_http_finalize_request(r, NGX_HTTP_NOT_FOUND);
2299                return NGX_DONE;
2300            }
2301
2302            if (r->method != NGX_HTTP_HEAD) {
2303                r->method = NGX_HTTP_GET;
2304            }
2305
2306            ngx_http_internal_redirect(r, &uri, &args);
2307        }
2308
2309        ngx_http_finalize_request(r, NGX_DONE);
2310        return NGX_DONE;
2311    }
2312
2313    part = &u->headers_in.headers.part;
2314    h = part->elts;
2315
2316    for (i = 0; /* void */; i++) {
2317
2318        if (i >= part->nelts) {
2319            if (part->next == NULL) {
2320                break;
2321            }
2322
2323            part = part->next;
2324            h = part->elts;
2325            i = 0;
2326        }
2327
2328        if (ngx_hash_find(&u->conf->hide_headers_hash, h[i].hash,
2329                          h[i].lowcase_key, h[i].key.len))
2330        {
2331            continue;
2332        }
2333
2334        hh = ngx_hash_find(&umcf->headers_in_hash, h[i].hash,
2335                           h[i].lowcase_key, h[i].key.len);
2336
2337        if (hh) {
2338            if (hh->copy_handler(r, &h[i], hh->conf) != NGX_OK) {
2339                ngx_http_upstream_finalize_request(r, u,
2340                                                   NGX_HTTP_INTERNAL_SERVER_ERROR);
2341                return NGX_DONE;
2342            }
2343
2344            continue;
2345        }
2346
2347        if (ngx_http_upstream_copy_header_line(r, &h[i], 0) != NGX_OK) {
2348            ngx_http_upstream_finalize_request(r, u,
2349                                               NGX_HTTP_INTERNAL_SERVER_ERROR);
2350            return NGX_DONE;
2351        }
2352    }
2353
```

```
    if (r->headers_out.server && r->headers_out.server->value.data == NULL) {
        r->headers_out.server->hash = 0;
    }

    if (r->headers_out.date && r->headers_out.date->value.data == NULL) {
        r->headers_out.date->hash = 0;
    }

    r->headers_out.status = u->headers_in.status_n;
    r->headers_out.status_line = u->headers_in.status_line;

    r->headers_out.content_length_n = u->headers_in.content_length_n;

    r->disable_not_modified = !u->cacheable;

    if (u->conf->force_ranges) {
        r->allow_ranges = 1;
        r->single_range = 1;

#if (NGX_HTTP_CACHE)
        if (r->cached) {
            r->single_range = 0;
        }
#endif
    }

    u->length = -1;

    return NGX_OK;
}


static void
ngx_http_upstream_process_body_in_memory(ngx_http_request_t *r,
    ngx_http_upstream_t *u)
{
    size_t             size;
    ssize_t            n;
    ngx_buf_t         *b;
    ngx_event_t       *rev;
    ngx_connection_t  *c;

    c = u->peer.connection;
    rev = c->read;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http upstream process body on memory");

    if (rev->timedout) {
        ngx_connection_error(c, NGX_ETIMEDOUT, "upstream timed out");
        ngx_http_upstream_finalize_request(r, u, NGX_HTTP_GATEWAY_TIME_OUT);
        return;
    }

    b = &u->buffer;

    for ( ;; ) {

        size = b->end - b->last;

        if (size == 0) {
            ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                          "upstream buffer is too small to read response");
            ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
            return;
        }

        n = c->recv(c, b->last, size);

        if (n == NGX_AGAIN) {
            break;
        }

        if (n == 0 || n == NGX_ERROR) {
            ngx_http_upstream_finalize_request(r, u, n);
            return;
```

```
2430         }
2431
2432         u->state->response_length += n;
2433
2434         if (u->input_filter(u->input_filter_ctx, n) == NGX_ERROR) {
2435             ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2436             return;
2437         }
2438
2439         if (!rev->ready) {
2440             break;
2441         }
2442     }
2443
2444     if (u->length == 0) {
2445         ngx_http_upstream_finalize_request(r, u, 0);
2446         return;
2447     }
2448
2449     if (ngx_handle_read_event(rev, 0) != NGX_OK) {
2450         ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2451         return;
2452     }
2453
2454     if (rev->active) {
2455         ngx_add_timer(rev, u->conf->read_timeout);
2456
2457     } else if (rev->timer_set) {
2458         ngx_del_timer(rev);
2459     }
2460 }
2461
2462
2463 static void
2464 ngx_http_upstream_send_response(ngx_http_request_t *r, ngx_http_upstream_t *u)
2465 {
2466     int                        tcp_nodelay;
2467     ssize_t                    n;
2468     ngx_int_t                  rc;
2469     ngx_event_pipe_t          *p;
2470     ngx_connection_t          *c;
2471     ngx_http_core_loc_conf_t  *clcf;
2472
2473     rc = ngx_http_send_header(r);
2474
2475     if (rc == NGX_ERROR || rc > NGX_OK || r->post_action) {
2476         ngx_http_upstream_finalize_request(r, u, rc);
2477         return;
2478     }
2479
2480     u->header_sent = 1;
2481
2482     if (u->upgrade) {
2483         ngx_http_upstream_upgrade(r, u);
2484         return;
2485     }
2486
2487     c = r->connection;
2488
2489     if (r->header_only) {
2490
2491         if (!u->buffering) {
2492             ngx_http_upstream_finalize_request(r, u, rc);
2493             return;
2494         }
2495
2496         if (!u->cacheable && !u->store) {
2497             ngx_http_upstream_finalize_request(r, u, rc);
2498             return;
2499         }
2500
2501         u->pipe->downstream_error = 1;
2502     }
2503
2504     if (r->request_body && r->request_body->temp_file) {
2505         ngx_pool_run_cleanup_file(r->pool, r->request_body->temp_file->file.fd);
```

```
2506         r->request_body->temp_file->file.fd = NGX_INVALID_FILE;
2507     }
2508 
2509     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2510 
2511     if (!u->buffering) {
2512 
2513         if (u->input_filter == NULL) {
2514             u->input_filter_init = ngx_http_upstream_non_buffered_filter_init;
2515             u->input_filter = ngx_http_upstream_non_buffered_filter;
2516             u->input_filter_ctx = r;
2517         }
2518 
2519         u->read_event_handler = ngx_http_upstream_process_non_buffered_upstream;
2520         r->write_event_handler =
2521                             ngx_http_upstream_process_non_buffered_downstream;
2522 
2523         r->limit_rate = 0;
2524 
2525         if (u->input_filter_init(u->input_filter_ctx) == NGX_ERROR) {
2526             ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2527             return;
2528         }
2529 
2530         if (clcf->tcp_nodelay && c->tcp_nodelay == NGX_TCP_NODELAY_UNSET) {
2531             ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "tcp_nodelay");
2532 
2533             tcp_nodelay = 1;
2534 
2535             if (setsockopt(c->fd, IPPROTO_TCP, TCP_NODELAY,
2536                            (const void *) &tcp_nodelay, sizeof(int)) == -1)
2537             {
2538                 ngx_connection_error(c, ngx_socket_errno,
2539                                      "setsockopt(TCP_NODELAY) failed");
2540                 ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2541                 return;
2542             }
2543 
2544             c->tcp_nodelay = NGX_TCP_NODELAY_SET;
2545         }
2546 
2547         n = u->buffer.last - u->buffer.pos;
2548 
2549         if (n) {
2550             u->buffer.last = u->buffer.pos;
2551 
2552             u->state->response_length += n;
2553 
2554             if (u->input_filter(u->input_filter_ctx, n) == NGX_ERROR) {
2555                 ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2556                 return;
2557             }
2558 
2559             ngx_http_upstream_process_non_buffered_downstream(r);
2560 
2561         } else {
2562             u->buffer.pos = u->buffer.start;
2563             u->buffer.last = u->buffer.start;
2564 
2565             if (ngx_http_send_special(r, NGX_HTTP_FLUSH) == NGX_ERROR) {
2566                 ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2567                 return;
2568             }
2569 
2570             if (u->peer.connection->read->ready || u->length == 0) {
2571                 ngx_http_upstream_process_non_buffered_upstream(r, u);
2572             }
2573         }
2574 
2575         return;
2576     }
2577 
2578     /* TODO: preallocate event_pipe bufs, look "Content-Length" */
2579 
2580 #if (NGX_HTTP_CACHE)
2581 
```

```
    if (r->cache && r->cache->file.fd != NGX_INVALID_FILE) {
        ngx_pool_run_cleanup_file(r->pool, r->cache->file.fd);
        r->cache->file.fd = NGX_INVALID_FILE;
    }

    switch (ngx_http_test_predicates(r, u->conf->no_cache)) {

    case NGX_ERROR:
        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
        return;

    case NGX_DECLINED:
        u->cacheable = 0;
        break;

    default: /* NGX_OK */

        if (u->cache_status == NGX_HTTP_CACHE_BYPASS) {

            /* create cache if previously bypassed */

            if (ngx_http_file_cache_create(r) != NGX_OK) {
                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                return;
            }
        }

        break;
    }

    if (u->cacheable) {
        time_t  now, valid;

        now = ngx_time();

        valid = r->cache->valid_sec;

        if (valid == 0) {
            valid = ngx_http_file_cache_valid(u->conf->cache_valid,
                                              u->headers_in.status_n);
            if (valid) {
                r->cache->valid_sec = now + valid;
            }
        }

        if (valid) {
            r->cache->date = now;
            r->cache->body_start = (u_short) (u->buffer.pos - u->buffer.start);

            if (u->headers_in.status_n == NGX_HTTP_OK
                || u->headers_in.status_n == NGX_HTTP_PARTIAL_CONTENT)
            {
                r->cache->last_modified = u->headers_in.last_modified_time;

                if (u->headers_in.etag) {
                    r->cache->etag = u->headers_in.etag->value;
                }
            }

            if (ngx_http_file_cache_set_header(r, u->buffer.start) != NGX_OK) {
                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                return;
            }

        } else {
            u->cacheable = 0;
        }
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http cacheable: %d", u->cacheable);

    if (u->cacheable == 0 && r->cache) {
        ngx_http_file_cache_free(r->cache, u->pipe->temp_file);
    }
```

```
#endif

    p = u->pipe;

    p->output_filter = (ngx_event_pipe_output_filter_pt) ngx_http_output_filter;
    p->output_ctx = r;
    p->tag = u->output.tag;
    p->bufs = u->conf->bufs;
    p->busy_size = u->conf->busy_buffers_size;
    p->upstream = u->peer.connection;
    p->downstream = c;
    p->pool = r->pool;
    p->log = c->log;
    p->limit_rate = u->conf->limit_rate;
    p->start_sec = ngx_time();

    p->cacheable = u->cacheable || u->store;

    p->temp_file = ngx_pcalloc(r->pool, sizeof(ngx_temp_file_t));
    if (p->temp_file == NULL) {
        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
        return;
    }

    p->temp_file->file.fd = NGX_INVALID_FILE;
    p->temp_file->file.log = c->log;
    p->temp_file->path = u->conf->temp_path;
    p->temp_file->pool = r->pool;

    if (p->cacheable) {
        p->temp_file->persistent = 1;

#if (NGX_HTTP_CACHE)
        if (r->cache && r->cache->file_cache->temp_path) {
            p->temp_file->path = r->cache->file_cache->temp_path;
        }
#endif

    } else {
        p->temp_file->log_level = NGX_LOG_WARN;
        p->temp_file->warn = "an upstream response is buffered "
                             "to a temporary file";
    }

    p->max_temp_file_size = u->conf->max_temp_file_size;
    p->temp_file_write_size = u->conf->temp_file_write_size;

    p->preread_bufs = ngx_alloc_chain_link(r->pool);
    if (p->preread_bufs == NULL) {
        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
        return;
    }

    p->preread_bufs->buf = &u->buffer;
    p->preread_bufs->next = NULL;
    u->buffer.recycled = 1;

    p->preread_size = u->buffer.last - u->buffer.pos;

    if (u->cacheable) {

        p->buf_to_file = ngx_calloc_buf(r->pool);
        if (p->buf_to_file == NULL) {
            ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
            return;
        }

        p->buf_to_file->start = u->buffer.start;
        p->buf_to_file->pos = u->buffer.start;
        p->buf_to_file->last = u->buffer.pos;
        p->buf_to_file->temporary = 1;
    }

    if (ngx_event_flags & NGX_USE_AIO_EVENT) {
        /* the posted aio operation may corrupt a shadow buffer */
        p->single_buf = 1;
```

```
2734     }
2735
2736     /* TODO: p->free_bufs = 0 if use ngx_create_chain_of_bufs() */
2737     p->free_bufs = 1;
2738
2739     /*
2740      * event_pipe would do u->buffer.last += p->preread_size
2741      * as though these bytes were read
2742      */
2743     u->buffer.last = u->buffer.pos;
2744
2745     if (u->conf->cyclic_temp_file) {
2746
2747         /*
2748          * we need to disable the use of sendfile() if we use cyclic temp file
2749          * because the writing a new data may interfere with sendfile()
2750          * that uses the same kernel file pages (at least on FreeBSD)
2751          */
2752
2753         p->cyclic_temp_file = 1;
2754         c->sendfile = 0;
2755
2756     } else {
2757         p->cyclic_temp_file = 0;
2758     }
2759
2760     p->read_timeout = u->conf->read_timeout;
2761     p->send_timeout = clcf->send_timeout;
2762     p->send_lowat = clcf->send_lowat;
2763
2764     p->length = -1;
2765
2766     if (u->input_filter_init
2767         && u->input_filter_init(p->input_ctx) != NGX_OK)
2768     {
2769         ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
2770         return;
2771     }
2772
2773     u->read_event_handler = ngx_http_upstream_process_upstream;
2774     r->write_event_handler = ngx_http_upstream_process_downstream;
2775
2776     ngx_http_upstream_process_upstream(r, u);
2777 }
2778
2779
2780 static void
2781 ngx_http_upstream_upgrade(ngx_http_request_t *r, ngx_http_upstream_t *u)
2782 {
2783     int                        tcp_nodelay;
2784     ngx_connection_t          *c;
2785     ngx_http_core_loc_conf_t  *clcf;
2786
2787     c = r->connection;
2788     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
2789
2790     /* TODO: prevent upgrade if not requested or not possible */
2791
2792     r->keepalive = 0;
2793     c->log->action = "proxying upgraded connection";
2794
2795     u->read_event_handler = ngx_http_upstream_upgraded_read_upstream;
2796     u->write_event_handler = ngx_http_upstream_upgraded_write_upstream;
2797     r->read_event_handler = ngx_http_upstream_upgraded_read_downstream;
2798     r->write_event_handler = ngx_http_upstream_upgraded_write_downstream;
2799
2800     if (clcf->tcp_nodelay) {
2801         tcp_nodelay = 1;
2802
2803         if (c->tcp_nodelay == NGX_TCP_NODELAY_UNSET) {
2804             ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0, "tcp_nodelay");
2805
2806             if (setsockopt(c->fd, IPPROTO_TCP, TCP_NODELAY,
2807                            (const void *) &tcp_nodelay, sizeof(int)) == -1)
2808             {
2809                 ngx_connection_error(c, ngx_socket_errno,
```

```
                                    "setsockopt(TCP_NODELAY) failed");
                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                return;
            }

            c->tcp_nodelay = NGX_TCP_NODELAY_SET;
        }

        if (u->peer.connection->tcp_nodelay == NGX_TCP_NODELAY_UNSET) {
            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, u->peer.connection->log, 0,
                           "tcp_nodelay");

            if (setsockopt(u->peer.connection->fd, IPPROTO_TCP, TCP_NODELAY,
                           (const void *) &tcp_nodelay, sizeof(int)) == -1)
            {
                ngx_connection_error(u->peer.connection, ngx_socket_errno,
                                     "setsockopt(TCP_NODELAY) failed");
                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                return;
            }

            u->peer.connection->tcp_nodelay = NGX_TCP_NODELAY_SET;
        }
    }

    if (ngx_http_send_special(r, NGX_HTTP_FLUSH) == NGX_ERROR) {
        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
        return;
    }

    if (u->peer.connection->read->ready
        || u->buffer.pos != u->buffer.last)
    {
        ngx_post_event(c->read, &ngx_posted_events);
        ngx_http_upstream_process_upgraded(r, 1, 1);
        return;
    }

    ngx_http_upstream_process_upgraded(r, 0, 1);
}


static void
ngx_http_upstream_upgraded_read_downstream(ngx_http_request_t *r)
{
    ngx_http_upstream_process_upgraded(r, 0, 0);
}


static void
ngx_http_upstream_upgraded_write_downstream(ngx_http_request_t *r)
{
    ngx_http_upstream_process_upgraded(r, 1, 1);
}


static void
ngx_http_upstream_upgraded_read_upstream(ngx_http_request_t *r,
    ngx_http_upstream_t *u)
{
    ngx_http_upstream_process_upgraded(r, 1, 0);
}


static void
ngx_http_upstream_upgraded_write_upstream(ngx_http_request_t *r,
    ngx_http_upstream_t *u)
{
    ngx_http_upstream_process_upgraded(r, 0, 1);
}


static void
ngx_http_upstream_process_upgraded(ngx_http_request_t *r,
    ngx_uint_t from_upstream, ngx_uint_t do_write)
{
```

```
    size_t                     size;
    ssize_t                    n;
    ngx_buf_t                 *b;
    ngx_connection_t          *c, *downstream, *upstream, *dst, *src;
    ngx_http_upstream_t       *u;
    ngx_http_core_loc_conf_t  *clcf;

    c = r->connection;
    u = r->upstream;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http upstream process upgraded, fu:%ui", from_upstream);

    downstream = c;
    upstream = u->peer.connection;

    if (downstream->write->timedout) {
        c->timedout = 1;
        ngx_connection_error(c, NGX_ETIMEDOUT, "client timed out");
        ngx_http_upstream_finalize_request(r, u, NGX_HTTP_REQUEST_TIME_OUT);
        return;
    }

    if (upstream->read->timedout || upstream->write->timedout) {
        ngx_connection_error(c, NGX_ETIMEDOUT, "upstream timed out");
        ngx_http_upstream_finalize_request(r, u, NGX_HTTP_GATEWAY_TIME_OUT);
        return;
    }

    if (from_upstream) {
        src = upstream;
        dst = downstream;
        b = &u->buffer;

    } else {
        src = downstream;
        dst = upstream;
        b = &u->from_client;

        if (r->header_in->last > r->header_in->pos) {
            b = r->header_in;
            b->end = b->last;
            do_write = 1;
        }

        if (b->start == NULL) {
            b->start = ngx_palloc(r->pool, u->conf->buffer_size);
            if (b->start == NULL) {
                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                return;
            }

            b->pos = b->start;
            b->last = b->start;
            b->end = b->start + u->conf->buffer_size;
            b->temporary = 1;
            b->tag = u->output.tag;
        }
    }

    for ( ;; ) {

        if (do_write) {

            size = b->last - b->pos;

            if (size && dst->write->ready) {

                n = dst->send(dst, b->pos, size);

                if (n == NGX_ERROR) {
                    ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                    return;
                }

                if (n > 0) {
```

```
                b->pos += n;

                if (b->pos == b->last) {
                    b->pos = b->start;
                    b->last = b->start;
                }
            }
        }
    }

    size = b->end - b->last;

    if (size && src->read->ready) {

        n = src->recv(src, b->last, size);

        if (n == NGX_AGAIN || n == 0) {
            break;
        }

        if (n > 0) {
            do_write = 1;
            b->last += n;

            continue;
        }

        if (n == NGX_ERROR) {
            src->read->eof = 1;
        }
    }

    break;
}

if ((upstream->read->eof && u->buffer.pos == u->buffer.last)
    || (downstream->read->eof && u->from_client.pos == u->from_client.last)
    || (downstream->read->eof && upstream->read->eof))
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http upstream upgraded done");
    ngx_http_upstream_finalize_request(r, u, 0);
    return;
}

clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

if (ngx_handle_write_event(upstream->write, u->conf->send_lowat)
    != NGX_OK)
{
    ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
    return;
}

if (upstream->write->active && !upstream->write->ready) {
    ngx_add_timer(upstream->write, u->conf->send_timeout);

} else if (upstream->write->timer_set) {
    ngx_del_timer(upstream->write);
}

if (ngx_handle_read_event(upstream->read, 0) != NGX_OK) {
    ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
    return;
}

if (upstream->read->active && !upstream->read->ready) {
    ngx_add_timer(upstream->read, u->conf->read_timeout);

} else if (upstream->read->timer_set) {
    ngx_del_timer(upstream->read);
}

if (ngx_handle_write_event(downstream->write, clcf->send_lowat)
    != NGX_OK)
{
```

```
3038        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3039        return;
3040    }
3041
3042    if (ngx_handle_read_event(downstream->read, 0) != NGX_OK) {
3043        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3044        return;
3045    }
3046
3047    if (downstream->write->active && !downstream->write->ready) {
3048        ngx_add_timer(downstream->write, clcf->send_timeout);
3049
3050    } else if (downstream->write->timer_set) {
3051        ngx_del_timer(downstream->write);
3052    }
3053 }
3054
3055
3056 static void
3057 ngx_http_upstream_process_non_buffered_downstream(ngx_http_request_t *r)
3058 {
3059    ngx_event_t          *wev;
3060    ngx_connection_t     *c;
3061    ngx_http_upstream_t  *u;
3062
3063    c = r->connection;
3064    u = r->upstream;
3065    wev = c->write;
3066
3067    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3068                   "http upstream process non buffered downstream");
3069
3070    c->log->action = "sending to client";
3071
3072    if (wev->timedout) {
3073        c->timedout = 1;
3074        ngx_connection_error(c, NGX_ETIMEDOUT, "client timed out");
3075        ngx_http_upstream_finalize_request(r, u, NGX_HTTP_REQUEST_TIME_OUT);
3076        return;
3077    }
3078
3079    ngx_http_upstream_process_non_buffered_request(r, 1);
3080 }
3081
3082
3083 static void
3084 ngx_http_upstream_process_non_buffered_upstream(ngx_http_request_t *r,
3085    ngx_http_upstream_t *u)
3086 {
3087    ngx_connection_t  *c;
3088
3089    c = u->peer.connection;
3090
3091    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3092                   "http upstream process non buffered upstream");
3093
3094    c->log->action = "reading upstream";
3095
3096    if (c->read->timedout) {
3097        ngx_connection_error(c, NGX_ETIMEDOUT, "upstream timed out");
3098        ngx_http_upstream_finalize_request(r, u, NGX_HTTP_GATEWAY_TIME_OUT);
3099        return;
3100    }
3101
3102    ngx_http_upstream_process_non_buffered_request(r, 0);
3103 }
3104
3105
3106 static void
3107 ngx_http_upstream_process_non_buffered_request(ngx_http_request_t *r,
3108    ngx_uint_t do_write)
3109 {
3110    size_t                     size;
3111    ssize_t                    n;
3112    ngx_buf_t                 *b;
3113    ngx_int_t                  rc;
```

```
    ngx_connection_t          *downstream, *upstream;
    ngx_http_upstream_t       *u;
    ngx_http_core_loc_conf_t  *clcf;

    u = r->upstream;
    downstream = r->connection;
    upstream = u->peer.connection;

    b = &u->buffer;

    do_write = do_write || u->length == 0;

    for ( ;; ) {

        if (do_write) {

            if (u->out_bufs || u->busy_bufs) {
                rc = ngx_http_output_filter(r, u->out_bufs);

                if (rc == NGX_ERROR) {
                    ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                    return;
                }

                ngx_chain_update_chains(r->pool, &u->free_bufs, &u->busy_bufs,
                                        &u->out_bufs, u->output.tag);
            }

            if (u->busy_bufs == NULL) {

                if (u->length == 0
                    || (upstream->read->eof && u->length == -1))
                {
                    ngx_http_upstream_finalize_request(r, u, 0);
                    return;
                }

                if (upstream->read->eof) {
                    ngx_log_error(NGX_LOG_ERR, upstream->log, 0,
                                  "upstream prematurely closed connection");

                    ngx_http_upstream_finalize_request(r, u,
                                                       NGX_HTTP_BAD_GATEWAY);
                    return;
                }

                if (upstream->read->error) {
                    ngx_http_upstream_finalize_request(r, u,
                                                       NGX_HTTP_BAD_GATEWAY);
                    return;
                }

                b->pos = b->start;
                b->last = b->start;
            }
        }

        size = b->end - b->last;

        if (size && upstream->read->ready) {

            n = upstream->recv(upstream, b->last, size);

            if (n == NGX_AGAIN) {
                break;
            }

            if (n > 0) {
                u->state->response_length += n;

                if (u->input_filter(u->input_filter_ctx, n) == NGX_ERROR) {
                    ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
                    return;
                }
            }

```

```
3190            do_write = 1;
3191
3192            continue;
3193        }
3194
3195        break;
3196    }
3197
3198    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
3199
3200    if (downstream->data == r) {
3201        if (ngx_handle_write_event(downstream->write, clcf->send_lowat)
3202            != NGX_OK)
3203        {
3204            ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3205            return;
3206        }
3207    }
3208
3209    if (downstream->write->active && !downstream->write->ready) {
3210        ngx_add_timer(downstream->write, clcf->send_timeout);
3211
3212    } else if (downstream->write->timer_set) {
3213        ngx_del_timer(downstream->write);
3214    }
3215
3216    if (ngx_handle_read_event(upstream->read, 0) != NGX_OK) {
3217        ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3218        return;
3219    }
3220
3221    if (upstream->read->active && !upstream->read->ready) {
3222        ngx_add_timer(upstream->read, u->conf->read_timeout);
3223
3224    } else if (upstream->read->timer_set) {
3225        ngx_del_timer(upstream->read);
3226    }
3227}
3228
3229
3230static ngx_int_t
3231ngx_http_upstream_non_buffered_filter_init(void *data)
3232{
3233    return NGX_OK;
3234}
3235
3236
3237static ngx_int_t
3238ngx_http_upstream_non_buffered_filter(void *data, ssize_t bytes)
3239{
3240    ngx_http_request_t  *r = data;
3241
3242    ngx_buf_t            *b;
3243    ngx_chain_t          *cl, **ll;
3244    ngx_http_upstream_t  *u;
3245
3246    u = r->upstream;
3247
3248    for (cl = u->out_bufs, ll = &u->out_bufs; cl; cl = cl->next) {
3249        ll = &cl->next;
3250    }
3251
3252    cl = ngx_chain_get_free_buf(r->pool, &u->free_bufs);
3253    if (cl == NULL) {
3254        return NGX_ERROR;
3255    }
3256
3257    *ll = cl;
3258
3259    cl->buf->flush = 1;
3260    cl->buf->memory = 1;
3261
3262    b = &u->buffer;
3263
3264    cl->buf->pos = b->last;
3265    b->last += bytes;
```

```
3266      cl->buf->last = b->last;
3267      cl->buf->tag = u->output.tag;
3268
3269      if (u->length == -1) {
3270          return NGX_OK;
3271      }
3272
3273      u->length -= bytes;
3274
3275      return NGX_OK;
3276  }
3277
3278
3279  static void
3280  ngx_http_upstream_process_downstream(ngx_http_request_t *r)
3281  {
3282      ngx_event_t          *wev;
3283      ngx_connection_t     *c;
3284      ngx_event_pipe_t     *p;
3285      ngx_http_upstream_t  *u;
3286
3287      c = r->connection;
3288      u = r->upstream;
3289      p = u->pipe;
3290      wev = c->write;
3291
3292      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3293                     "http upstream process downstream");
3294
3295      c->log->action = "sending to client";
3296
3297      if (wev->timedout) {
3298
3299          if (wev->delayed) {
3300
3301              wev->timedout = 0;
3302              wev->delayed = 0;
3303
3304              if (!wev->ready) {
3305                  ngx_add_timer(wev, p->send_timeout);
3306
3307                  if (ngx_handle_write_event(wev, p->send_lowat) != NGX_OK) {
3308                      ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3309                  }
3310
3311                  return;
3312              }
3313
3314              if (ngx_event_pipe(p, wev->write) == NGX_ABORT) {
3315                  ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3316                  return;
3317              }
3318
3319          } else {
3320              p->downstream_error = 1;
3321              c->timedout = 1;
3322              ngx_connection_error(c, NGX_ETIMEDOUT, "client timed out");
3323          }
3324
3325      } else {
3326
3327          if (wev->delayed) {
3328
3329              ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3330                             "http downstream delayed");
3331
3332              if (ngx_handle_write_event(wev, p->send_lowat) != NGX_OK) {
3333                  ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3334              }
3335
3336              return;
3337          }
3338
3339          if (ngx_event_pipe(p, 1) == NGX_ABORT) {
3340              ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3341              return;
```

```
3342        }
3343    }
3344
3345    ngx_http_upstream_process_request(r);
3346}
3347
3348
3349static void
3350ngx_http_upstream_process_upstream(ngx_http_request_t *r,
3351    ngx_http_upstream_t *u)
3352{
3353    ngx_event_t       *rev;
3354    ngx_event_pipe_t  *p;
3355    ngx_connection_t  *c;
3356
3357    c = u->peer.connection;
3358    p = u->pipe;
3359    rev = c->read;
3360
3361    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3362                   "http upstream process upstream");
3363
3364    c->log->action = "reading upstream";
3365
3366    if (rev->timedout) {
3367
3368        if (rev->delayed) {
3369
3370            rev->timedout = 0;
3371            rev->delayed = 0;
3372
3373            if (!rev->ready) {
3374                ngx_add_timer(rev, p->read_timeout);
3375
3376                if (ngx_handle_read_event(rev, 0) != NGX_OK) {
3377                    ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3378                }
3379
3380                return;
3381            }
3382
3383            if (ngx_event_pipe(p, 0) == NGX_ABORT) {
3384                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3385                return;
3386            }
3387
3388        } else {
3389            p->upstream_error = 1;
3390            ngx_connection_error(c, NGX_ETIMEDOUT, "upstream timed out");
3391        }
3392
3393    } else {
3394
3395        if (rev->delayed) {
3396
3397            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->log, 0,
3398                           "http upstream delayed");
3399
3400            if (ngx_handle_read_event(rev, 0) != NGX_OK) {
3401                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3402            }
3403
3404            return;
3405        }
3406
3407        if (ngx_event_pipe(p, 0) == NGX_ABORT) {
3408            ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3409            return;
3410        }
3411    }
3412
3413    ngx_http_upstream_process_request(r);
3414}
3415
3416
3417static void
```

```
3418 ngx_http_upstream_process_request(ngx_http_request_t *r)
3419 {
3420     ngx_temp_file_t      *tf;
3421     ngx_event_pipe_t     *p;
3422     ngx_http_upstream_t  *u;
3423
3424     u = r->upstream;
3425     p = u->pipe;
3426
3427     if (u->peer.connection) {
3428
3429         if (u->store) {
3430
3431             if (p->upstream_eof || p->upstream_done) {
3432
3433                 tf = p->temp_file;
3434
3435                 if (u->headers_in.status_n == NGX_HTTP_OK
3436                     && (p->upstream_done || p->length == -1)
3437                     && (u->headers_in.content_length_n == -1
3438                         || u->headers_in.content_length_n == tf->offset))
3439                 {
3440                     ngx_http_upstream_store(r, u);
3441                 }
3442             }
3443         }
3444
3445 #if (NGX_HTTP_CACHE)
3446
3447         if (u->cacheable) {
3448
3449             if (p->upstream_done) {
3450                 ngx_http_file_cache_update(r, p->temp_file);
3451
3452             } else if (p->upstream_eof) {
3453
3454                 tf = p->temp_file;
3455
3456                 if (p->length == -1
3457                     && (u->headers_in.content_length_n == -1
3458                         || u->headers_in.content_length_n
3459                            == tf->offset - (off_t) r->cache->body_start))
3460                 {
3461                     ngx_http_file_cache_update(r, tf);
3462
3463                 } else {
3464                     ngx_http_file_cache_free(r->cache, tf);
3465                 }
3466
3467             } else if (p->upstream_error) {
3468                 ngx_http_file_cache_free(r->cache, p->temp_file);
3469             }
3470         }
3471
3472 #endif
3473
3474         if (p->upstream_done || p->upstream_eof || p->upstream_error) {
3475             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3476                            "http upstream exit: %p", p->out);
3477
3478             if (p->upstream_done
3479                 || (p->upstream_eof && p->length == -1))
3480             {
3481                 ngx_http_upstream_finalize_request(r, u, 0);
3482                 return;
3483             }
3484
3485             if (p->upstream_eof) {
3486                 ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
3487                               "upstream prematurely closed connection");
3488             }
3489
3490             ngx_http_upstream_finalize_request(r, u, NGX_HTTP_BAD_GATEWAY);
3491             return;
3492         }
3493     }
```

```
3494
3495        if (p->downstream_error) {
3496            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3497                           "http upstream downstream error");
3498
3499            if (!u->cacheable && !u->store && u->peer.connection) {
3500                ngx_http_upstream_finalize_request(r, u, NGX_ERROR);
3501            }
3502        }
3503    }
3504
3505
3506    static void
3507    ngx_http_upstream_store(ngx_http_request_t *r, ngx_http_upstream_t *u)
3508    {
3509        size_t                  root;
3510        time_t                  lm;
3511        ngx_str_t               path;
3512        ngx_temp_file_t        *tf;
3513        ngx_ext_rename_file_t   ext;
3514
3515        tf = u->pipe->temp_file;
3516
3517        if (tf->file.fd == NGX_INVALID_FILE) {
3518
3519            /* create file for empty 200 response */
3520
3521            tf = ngx_pcalloc(r->pool, sizeof(ngx_temp_file_t));
3522            if (tf == NULL) {
3523                return;
3524            }
3525
3526            tf->file.fd = NGX_INVALID_FILE;
3527            tf->file.log = r->connection->log;
3528            tf->path = u->conf->temp_path;
3529            tf->pool = r->pool;
3530            tf->persistent = 1;
3531
3532            if (ngx_create_temp_file(&tf->file, tf->path, tf->pool,
3533                                     tf->persistent, tf->clean, tf->access)
3534                != NGX_OK)
3535            {
3536                return;
3537            }
3538
3539            u->pipe->temp_file = tf;
3540        }
3541
3542        ext.access = u->conf->store_access;
3543        ext.path_access = u->conf->store_access;
3544        ext.time = -1;
3545        ext.create_path = 1;
3546        ext.delete_file = 1;
3547        ext.log = r->connection->log;
3548
3549        if (u->headers_in.last_modified) {
3550
3551            lm = ngx_http_parse_time(u->headers_in.last_modified->value.data,
3552                                     u->headers_in.last_modified->value.len);
3553
3554            if (lm != NGX_ERROR) {
3555                ext.time = lm;
3556                ext.fd = tf->file.fd;
3557            }
3558        }
3559
3560        if (u->conf->store_lengths == NULL) {
3561
3562            if (ngx_http_map_uri_to_path(r, &path, &root, 0) == NULL) {
3563                return;
3564            }
3565
3566        } else {
3567            if (ngx_http_script_run(r, &path, u->conf->store_lengths->elts, 0,
3568                                    u->conf->store_values->elts)
3569                == NULL)
```

```
3570         {
3571             return;
3572         }
3573     }
3574 
3575     path.len--;
3576 
3577     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3578                    "upstream stores \"%s\" to \"%s\"",
3579                    tf->file.name.data, path.data);
3580 
3581     (void) ngx_ext_rename_file(&tf->file.name, &path, &ext);
3582 
3583     u->store = 0;
3584 }
3585 
3586 
3587 static void
3588 ngx_http_upstream_dummy_handler(ngx_http_request_t *r, ngx_http_upstream_t *u)
3589 {
3590     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3591                    "http upstream dummy handler");
3592 }
3593 
3594 
3595 static void
3596 ngx_http_upstream_next(ngx_http_request_t *r, ngx_http_upstream_t *u,
3597     ngx_uint_t ft_type)
3598 {
3599     ngx_msec_t  timeout;
3600     ngx_uint_t  status, state;
3601 
3602     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3603                    "http next upstream, %xi", ft_type);
3604 
3605     if (u->peer.sockaddr) {
3606 
3607         if (ft_type == NGX_HTTP_UPSTREAM_FT_HTTP_403
3608             || ft_type == NGX_HTTP_UPSTREAM_FT_HTTP_404)
3609         {
3610             state = NGX_PEER_NEXT;
3611 
3612         } else {
3613             state = NGX_PEER_FAILED;
3614         }
3615 
3616         u->peer.free(&u->peer, u->peer.data, state);
3617         u->peer.sockaddr = NULL;
3618     }
3619 
3620     if (ft_type == NGX_HTTP_UPSTREAM_FT_TIMEOUT) {
3621         ngx_log_error(NGX_LOG_ERR, r->connection->log, NGX_ETIMEDOUT,
3622                       "upstream timed out");
3623     }
3624 
3625     if (u->peer.cached && ft_type == NGX_HTTP_UPSTREAM_FT_ERROR) {
3626         status = 0;
3627 
3628         /* TODO: inform balancer instead */
3629 
3630         u->peer.tries++;
3631 
3632     } else {
3633         switch (ft_type) {
3634 
3635         case NGX_HTTP_UPSTREAM_FT_TIMEOUT:
3636             status = NGX_HTTP_GATEWAY_TIME_OUT;
3637             break;
3638 
3639         case NGX_HTTP_UPSTREAM_FT_HTTP_500:
3640             status = NGX_HTTP_INTERNAL_SERVER_ERROR;
3641             break;
3642 
3643         case NGX_HTTP_UPSTREAM_FT_HTTP_403:
3644             status = NGX_HTTP_FORBIDDEN;
3645             break;
```

```
3646
3647            case NGX_HTTP_UPSTREAM_FT_HTTP_404:
3648                status = NGX_HTTP_NOT_FOUND;
3649                break;
3650
3651            /*
3652             * NGX_HTTP_UPSTREAM_FT_BUSY_LOCK and NGX_HTTP_UPSTREAM_FT_MAX_WAITING
3653             * never reach here
3654             */
3655
3656            default:
3657                status = NGX_HTTP_BAD_GATEWAY;
3658            }
3659        }
3660
3661        if (r->connection->error) {
3662            ngx_http_upstream_finalize_request(r, u,
3663                                               NGX_HTTP_CLIENT_CLOSED_REQUEST);
3664            return;
3665        }
3666
3667        if (status) {
3668            u->state->status = status;
3669            timeout = u->conf->next_upstream_timeout;
3670
3671            if (u->peer.tries == 0
3672                || !(u->conf->next_upstream & ft_type)
3673                || (timeout && ngx_current_msec - u->peer.start_time >= timeout))
3674            {
3675    #if (NGX_HTTP_CACHE)
3676
3677                if (u->cache_status == NGX_HTTP_CACHE_EXPIRED
3678                    && (u->conf->cache_use_stale & ft_type))
3679                {
3680                    ngx_int_t  rc;
3681
3682                    rc = u->reinit_request(r);
3683
3684                    if (rc == NGX_OK) {
3685                        u->cache_status = NGX_HTTP_CACHE_STALE;
3686                        rc = ngx_http_upstream_cache_send(r, u);
3687                    }
3688
3689                    ngx_http_upstream_finalize_request(r, u, rc);
3690                    return;
3691                }
3692    #endif
3693
3694                ngx_http_upstream_finalize_request(r, u, status);
3695                return;
3696            }
3697        }
3698
3699        if (u->peer.connection) {
3700            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3701                           "close http upstream connection: %d",
3702                           u->peer.connection->fd);
3703    #if (NGX_HTTP_SSL)
3704
3705            if (u->peer.connection->ssl) {
3706                u->peer.connection->ssl->no_wait_shutdown = 1;
3707                u->peer.connection->ssl->no_send_shutdown = 1;
3708
3709                (void) ngx_ssl_shutdown(u->peer.connection);
3710            }
3711    #endif
3712
3713            if (u->peer.connection->pool) {
3714                ngx_destroy_pool(u->peer.connection->pool);
3715            }
3716
3717            ngx_close_connection(u->peer.connection);
3718            u->peer.connection = NULL;
3719        }
3720
3721        ngx_http_upstream_connect(r, u);
```

```
}


static void
ngx_http_upstream_cleanup(void *data)
{
    ngx_http_request_t *r = data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "cleanup http upstream request: \"%V\"", &r->uri);

    ngx_http_upstream_finalize_request(r, r->upstream, NGX_DONE);
}


static void
ngx_http_upstream_finalize_request(ngx_http_request_t *r,
    ngx_http_upstream_t *u, ngx_int_t rc)
{
    ngx_uint_t   flush;
    ngx_time_t  *tp;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "finalize http upstream request: %i", rc);

    if (u->cleanup) {
        *u->cleanup = NULL;
        u->cleanup = NULL;
    }

    if (u->resolved && u->resolved->ctx) {
        ngx_resolve_name_done(u->resolved->ctx);
        u->resolved->ctx = NULL;
    }

    if (u->state && u->state->response_sec) {
        tp = ngx_timeofday();
        u->state->response_sec = tp->sec - u->state->response_sec;
        u->state->response_msec = tp->msec - u->state->response_msec;

        if (u->pipe && u->pipe->read_length) {
            u->state->response_length = u->pipe->read_length;
        }
    }

    u->finalize_request(r, rc);

    if (u->peer.free && u->peer.sockaddr) {
        u->peer.free(&u->peer, u->peer.data, 0);
        u->peer.sockaddr = NULL;
    }

    if (u->peer.connection) {

#if (NGX_HTTP_SSL)

        /* TODO: do not shutdown persistent connection */

        if (u->peer.connection->ssl) {

            /*
             * We send the "close notify" shutdown alert to the upstream only
             * and do not wait its "close notify" shutdown alert.
             * It is acceptable according to the TLS standard.
             */

            u->peer.connection->ssl->no_wait_shutdown = 1;

            (void) ngx_ssl_shutdown(u->peer.connection);
        }
#endif

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "close http upstream connection: %d",
                       u->peer.connection->fd);

```

```
3798         if (u->peer.connection->pool) {
3799             ngx_destroy_pool(u->peer.connection->pool);
3800         }
3801 
3802         ngx_close_connection(u->peer.connection);
3803     }
3804 
3805     u->peer.connection = NULL;
3806 
3807     if (u->pipe && u->pipe->temp_file) {
3808         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
3809                        "http upstream temp fd: %d",
3810                        u->pipe->temp_file->file.fd);
3811     }
3812 
3813     if (u->store && u->pipe && u->pipe->temp_file
3814         && u->pipe->temp_file->file.fd != NGX_INVALID_FILE)
3815     {
3816         if (ngx_delete_file(u->pipe->temp_file->file.name.data)
3817             == NGX_FILE_ERROR)
3818         {
3819             ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
3820                           ngx_delete_file_n " \"%s\" failed",
3821                           u->pipe->temp_file->file.name.data);
3822         }
3823     }
3824 
3825 #if (NGX_HTTP_CACHE)
3826 
3827     if (r->cache) {
3828 
3829         if (u->cacheable) {
3830 
3831             if (rc == NGX_HTTP_BAD_GATEWAY || rc == NGX_HTTP_GATEWAY_TIME_OUT) {
3832                 time_t  valid;
3833 
3834                 valid = ngx_http_file_cache_valid(u->conf->cache_valid, rc);
3835 
3836                 if (valid) {
3837                     r->cache->valid_sec = ngx_time() + valid;
3838                     r->cache->error = rc;
3839                 }
3840             }
3841         }
3842 
3843         ngx_http_file_cache_free(r->cache, u->pipe->temp_file);
3844     }
3845 
3846 #endif
3847 
3848     if (r->subrequest_in_memory
3849         && u->headers_in.status_n >= NGX_HTTP_SPECIAL_RESPONSE)
3850     {
3851         u->buffer.last = u->buffer.pos;
3852     }
3853 
3854     if (rc == NGX_DECLINED) {
3855         return;
3856     }
3857 
3858     r->connection->log->action = "sending to client";
3859 
3860     if (!u->header_sent
3861         || rc == NGX_HTTP_REQUEST_TIME_OUT
3862         || rc == NGX_HTTP_CLIENT_CLOSED_REQUEST)
3863     {
3864         ngx_http_finalize_request(r, rc);
3865         return;
3866     }
3867 
3868     flush = 0;
3869 
3870     if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
3871         rc = NGX_ERROR;
3872         flush = 1;
3873     }
```

```
3874
3875        if (r->header_only) {
3876            ngx_http_finalize_request(r, rc);
3877            return;
3878        }
3879
3880        if (rc == 0) {
3881            rc = ngx_http_send_special(r, NGX_HTTP_LAST);
3882
3883        } else if (flush) {
3884            r->keepalive = 0;
3885            rc = ngx_http_send_special(r, NGX_HTTP_FLUSH);
3886        }
3887
3888        ngx_http_finalize_request(r, rc);
3889    }
3890
3891
3892    static ngx_int_t
3893    ngx_http_upstream_process_header_line(ngx_http_request_t *r, ngx_table_elt_t *h,
3894        ngx_uint_t offset)
3895    {
3896        ngx_table_elt_t  **ph;
3897
3898        ph = (ngx_table_elt_t **) ((char *) &r->upstream->headers_in + offset);
3899
3900        if (*ph == NULL) {
3901            *ph = h;
3902        }
3903
3904        return NGX_OK;
3905    }
3906
3907
3908    static ngx_int_t
3909    ngx_http_upstream_ignore_header_line(ngx_http_request_t *r, ngx_table_elt_t *h,
3910        ngx_uint_t offset)
3911    {
3912        return NGX_OK;
3913    }
3914
3915
3916    static ngx_int_t
3917    ngx_http_upstream_process_content_length(ngx_http_request_t *r,
3918        ngx_table_elt_t *h, ngx_uint_t offset)
3919    {
3920        ngx_http_upstream_t  *u;
3921
3922        u = r->upstream;
3923
3924        u->headers_in.content_length = h;
3925        u->headers_in.content_length_n = ngx_atoof(h->value.data, h->value.len);
3926
3927        return NGX_OK;
3928    }
3929
3930
3931    static ngx_int_t
3932    ngx_http_upstream_process_last_modified(ngx_http_request_t *r,
3933        ngx_table_elt_t *h, ngx_uint_t offset)
3934    {
3935        ngx_http_upstream_t  *u;
3936
3937        u = r->upstream;
3938
3939        u->headers_in.last_modified = h;
3940
3941    #if (NGX_HTTP_CACHE)
3942
3943        if (u->cacheable) {
3944            u->headers_in.last_modified_time = ngx_http_parse_time(h->value.data,
3945                                                                   h->value.len);
3946        }
3947
3948    #endif
3949
```

```
3950     return NGX_OK;
3951 }
3952
3953
3954 static ngx_int_t
3955 ngx_http_upstream_process_set_cookie(ngx_http_request_t *r, ngx_table_elt_t *h,
3956     ngx_uint_t offset)
3957 {
3958     ngx_array_t           *pa;
3959     ngx_table_elt_t      **ph;
3960     ngx_http_upstream_t   *u;
3961
3962     u = r->upstream;
3963     pa = &u->headers_in.cookies;
3964
3965     if (pa->elts == NULL) {
3966         if (ngx_array_init(pa, r->pool, 1, sizeof(ngx_table_elt_t *)) != NGX_OK)
3967         {
3968             return NGX_ERROR;
3969         }
3970     }
3971
3972     ph = ngx_array_push(pa);
3973     if (ph == NULL) {
3974         return NGX_ERROR;
3975     }
3976
3977     *ph = h;
3978
3979 #if (NGX_HTTP_CACHE)
3980     if (!(u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_SET_COOKIE)) {
3981         u->cacheable = 0;
3982     }
3983 #endif
3984
3985     return NGX_OK;
3986 }
3987
3988
3989 static ngx_int_t
3990 ngx_http_upstream_process_cache_control(ngx_http_request_t *r,
3991     ngx_table_elt_t *h, ngx_uint_t offset)
3992 {
3993     ngx_array_t           *pa;
3994     ngx_table_elt_t      **ph;
3995     ngx_http_upstream_t   *u;
3996
3997     u = r->upstream;
3998     pa = &u->headers_in.cache_control;
3999
4000     if (pa->elts == NULL) {
4001        if (ngx_array_init(pa, r->pool, 2, sizeof(ngx_table_elt_t *)) != NGX_OK)
4002        {
4003            return NGX_ERROR;
4004        }
4005     }
4006
4007     ph = ngx_array_push(pa);
4008     if (ph == NULL) {
4009         return NGX_ERROR;
4010     }
4011
4012     *ph = h;
4013
4014 #if (NGX_HTTP_CACHE)
4015     {
4016     u_char     *p, *start, *last;
4017     ngx_int_t   n;
4018
4019     if (u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_CACHE_CONTROL) {
4020         return NGX_OK;
4021     }
4022
4023     if (r->cache == NULL) {
4024         return NGX_OK;
4025     }
```

```
4026
4027        if (r->cache->valid_sec != 0 && u->headers_in.x_accel_expires != NULL) {
4028            return NGX_OK;
4029        }
4030
4031        start = h->value.data;
4032        last = start + h->value.len;
4033
4034        if (ngx_strlcasestrn(start, last, (u_char *) "no-cache", 8 - 1) != NULL
4035            || ngx_strlcasestrn(start, last, (u_char *) "no-store", 8 - 1) != NULL
4036            || ngx_strlcasestrn(start, last, (u_char *) "private", 7 - 1) != NULL)
4037        {
4038            u->cacheable = 0;
4039            return NGX_OK;
4040        }
4041
4042        p = ngx_strlcasestrn(start, last, (u_char *) "s-maxage=", 9 - 1);
4043        offset = 9;
4044
4045        if (p == NULL) {
4046            p = ngx_strlcasestrn(start, last, (u_char *) "max-age=", 8 - 1);
4047            offset = 8;
4048        }
4049
4050        if (p == NULL) {
4051            return NGX_OK;
4052        }
4053
4054        n = 0;
4055
4056        for (p += offset; p < last; p++) {
4057            if (*p == ',' || *p == ';' || *p == ' ') {
4058                break;
4059            }
4060
4061            if (*p >= '0' && *p <= '9') {
4062                n = n * 10 + *p - '0';
4063                continue;
4064            }
4065
4066            u->cacheable = 0;
4067            return NGX_OK;
4068        }
4069
4070        if (n == 0) {
4071            u->cacheable = 0;
4072            return NGX_OK;
4073        }
4074
4075        r->cache->valid_sec = ngx_time() + n;
4076        }
4077    #endif
4078
4079        return NGX_OK;
4080    }
4081
4082
4083    static ngx_int_t
4084    ngx_http_upstream_process_expires(ngx_http_request_t *r, ngx_table_elt_t *h,
4085        ngx_uint_t offset)
4086    {
4087        ngx_http_upstream_t  *u;
4088
4089        u = r->upstream;
4090        u->headers_in.expires = h;
4091
4092    #if (NGX_HTTP_CACHE)
4093        {
4094        time_t  expires;
4095
4096        if (u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_EXPIRES) {
4097            return NGX_OK;
4098        }
4099
4100        if (r->cache == NULL) {
4101            return NGX_OK;
```

```
4102     }
4103
4104     if (r->cache->valid_sec != 0) {
4105         return NGX_OK;
4106     }
4107
4108     expires = ngx_http_parse_time(h->value.data, h->value.len);
4109
4110     if (expires == NGX_ERROR || expires < ngx_time()) {
4111         u->cacheable = 0;
4112         return NGX_OK;
4113     }
4114
4115     r->cache->valid_sec = expires;
4116     }
4117 #endif
4118
4119     return NGX_OK;
4120 }
4121
4122
4123 static ngx_int_t
4124 ngx_http_upstream_process_accel_expires(ngx_http_request_t *r,
4125     ngx_table_elt_t *h, ngx_uint_t offset)
4126 {
4127     ngx_http_upstream_t  *u;
4128
4129     u = r->upstream;
4130     u->headers_in.x_accel_expires = h;
4131
4132 #if (NGX_HTTP_CACHE)
4133     {
4134     u_char     *p;
4135     size_t      len;
4136     ngx_int_t   n;
4137
4138     if (u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_XA_EXPIRES) {
4139         return NGX_OK;
4140     }
4141
4142     if (r->cache == NULL) {
4143         return NGX_OK;
4144     }
4145
4146     len = h->value.len;
4147     p = h->value.data;
4148
4149     if (p[0] != '@') {
4150         n = ngx_atoi(p, len);
4151
4152         switch (n) {
4153         case 0:
4154             u->cacheable = 0;
4155             /* fall through */
4156
4157         case NGX_ERROR:
4158             return NGX_OK;
4159
4160         default:
4161             r->cache->valid_sec = ngx_time() + n;
4162             return NGX_OK;
4163         }
4164     }
4165
4166     p++;
4167     len--;
4168
4169     n = ngx_atoi(p, len);
4170
4171     if (n != NGX_ERROR) {
4172         r->cache->valid_sec = n;
4173     }
4174     }
4175 #endif
4176
4177     return NGX_OK;
```

```
}


static ngx_int_t
ngx_http_upstream_process_limit_rate(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    ngx_int_t             n;
    ngx_http_upstream_t  *u;

    u = r->upstream;
    u->headers_in.x_accel_limit_rate = h;

    if (u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_XA_LIMIT_RATE) {
        return NGX_OK;
    }

    n = ngx_atoi(h->value.data, h->value.len);

    if (n != NGX_ERROR) {
        r->limit_rate = (size_t) n;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_process_buffering(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    u_char                c0, c1, c2;
    ngx_http_upstream_t  *u;

    u = r->upstream;

    if (u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_XA_BUFFERING) {
        return NGX_OK;
    }

    if (u->conf->change_buffering) {

        if (h->value.len == 2) {
            c0 = ngx_tolower(h->value.data[0]);
            c1 = ngx_tolower(h->value.data[1]);

            if (c0 == 'n' && c1 == 'o') {
                u->buffering = 0;
            }

        } else if (h->value.len == 3) {
            c0 = ngx_tolower(h->value.data[0]);
            c1 = ngx_tolower(h->value.data[1]);
            c2 = ngx_tolower(h->value.data[2]);

            if (c0 == 'y' && c1 == 'e' && c2 == 's') {
                u->buffering = 1;
            }
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_process_charset(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    if (r->upstream->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_XA_CHARSET) {
        return NGX_OK;
    }

    r->headers_out.override_charset = &h->value;

    return NGX_OK;
```

```
}


static ngx_int_t
ngx_http_upstream_process_connection(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    r->upstream->headers_in.connection = h;

    if (ngx_strlcasestrn(h->value.data, h->value.data + h->value.len,
                         (u_char *) "close", 5 - 1)
        != NULL)
    {
        r->upstream->headers_in.connection_close = 1;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_process_transfer_encoding(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset)
{
    r->upstream->headers_in.transfer_encoding = h;

    if (ngx_strlcasestrn(h->value.data, h->value.data + h->value.len,
                         (u_char *) "chunked", 7 - 1)
        != NULL)
    {
        r->upstream->headers_in.chunked = 1;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_process_vary(ngx_http_request_t *r,
    ngx_table_elt_t *h, ngx_uint_t offset)
{
    ngx_http_upstream_t  *u;

    u = r->upstream;
    u->headers_in.vary = h;

#if (NGX_HTTP_CACHE)

    if (u->conf->ignore_headers & NGX_HTTP_UPSTREAM_IGN_VARY) {
        return NGX_OK;
    }

    if (r->cache == NULL) {
        return NGX_OK;
    }

    if (h->value.len > NGX_HTTP_CACHE_VARY_LEN
        || (h->value.len == 1 && h->value.data[0] == '*'))
    {
        u->cacheable = 0;
    }

    r->cache->vary = h->value;

#endif

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_copy_header_line(ngx_http_request_t *r, ngx_table_elt_t *h,
    ngx_uint_t offset)
{
    ngx_table_elt_t  *ho, **ph;

```

```
4330      ho = ngx_list_push(&r->headers_out.headers);
4331      if (ho == NULL) {
4332          return NGX_ERROR;
4333      }
4334
4335      *ho = *h;
4336
4337      if (offset) {
4338          ph = (ngx_table_elt_t **) ((char *) &r->headers_out + offset);
4339          *ph = ho;
4340      }
4341
4342      return NGX_OK;
4343  }
4344
4345
4346  static ngx_int_t
4347  ngx_http_upstream_copy_multi_header_lines(ngx_http_request_t *r,
4348      ngx_table_elt_t *h, ngx_uint_t offset)
4349  {
4350      ngx_array_t      *pa;
4351      ngx_table_elt_t  *ho, **ph;
4352
4353      pa = (ngx_array_t *) ((char *) &r->headers_out + offset);
4354
4355      if (pa->elts == NULL) {
4356          if (ngx_array_init(pa, r->pool, 2, sizeof(ngx_table_elt_t *)) != NGX_OK)
4357          {
4358              return NGX_ERROR;
4359          }
4360      }
4361
4362      ph = ngx_array_push(pa);
4363      if (ph == NULL) {
4364          return NGX_ERROR;
4365      }
4366
4367      ho = ngx_list_push(&r->headers_out.headers);
4368      if (ho == NULL) {
4369          return NGX_ERROR;
4370      }
4371
4372      *ho = *h;
4373      *ph = ho;
4374
4375      return NGX_OK;
4376  }
4377
4378
4379  static ngx_int_t
4380  ngx_http_upstream_copy_content_type(ngx_http_request_t *r, ngx_table_elt_t *h,
4381      ngx_uint_t offset)
4382  {
4383      u_char  *p, *last;
4384
4385      r->headers_out.content_type_len = h->value.len;
4386      r->headers_out.content_type = h->value;
4387      r->headers_out.content_type_lowcase = NULL;
4388
4389      for (p = h->value.data; *p; p++) {
4390
4391          if (*p != ';') {
4392              continue;
4393          }
4394
4395          last = p;
4396
4397          while (*++p == ' ') { /* void */ }
4398
4399          if (*p == '\0') {
4400              return NGX_OK;
4401          }
4402
4403          if (ngx_strncasecmp(p, (u_char *) "charset=", 8) != 0) {
4404              continue;
4405          }
```

```
4406
4407            p += 8;
4408
4409            r->headers_out.content_type_len = last - h->value.data;
4410
4411            if (*p == '"') {
4412                p++;
4413            }
4414
4415            last = h->value.data + h->value.len;
4416
4417            if (*(last - 1) == '"') {
4418                last--;
4419            }
4420
4421            r->headers_out.charset.len = last - p;
4422            r->headers_out.charset.data = p;
4423
4424            return NGX_OK;
4425        }
4426
4427        return NGX_OK;
4428    }
4429
4430
4431    static ngx_int_t
4432    ngx_http_upstream_copy_last_modified(ngx_http_request_t *r, ngx_table_elt_t *h,
4433        ngx_uint_t offset)
4434    {
4435        ngx_table_elt_t  *ho;
4436
4437        ho = ngx_list_push(&r->headers_out.headers);
4438        if (ho == NULL) {
4439            return NGX_ERROR;
4440        }
4441
4442        *ho = *h;
4443
4444        r->headers_out.last_modified = ho;
4445
4446    #if (NGX_HTTP_CACHE)
4447
4448        if (r->upstream->cacheable) {
4449            r->headers_out.last_modified_time =
4450                                        r->upstream->headers_in.last_modified_time;
4451        }
4452
4453    #endif
4454
4455        return NGX_OK;
4456    }
4457
4458
4459    static ngx_int_t
4460    ngx_http_upstream_rewrite_location(ngx_http_request_t *r, ngx_table_elt_t *h,
4461        ngx_uint_t offset)
4462    {
4463        ngx_int_t         rc;
4464        ngx_table_elt_t  *ho;
4465
4466        ho = ngx_list_push(&r->headers_out.headers);
4467        if (ho == NULL) {
4468            return NGX_ERROR;
4469        }
4470
4471        *ho = *h;
4472
4473        if (r->upstream->rewrite_redirect) {
4474            rc = r->upstream->rewrite_redirect(r, ho, 0);
4475
4476            if (rc == NGX_DECLINED) {
4477                return NGX_OK;
4478            }
4479
4480            if (rc == NGX_OK) {
4481                r->headers_out.location = ho;
```

```
4482
4483            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
4484                           "rewritten location: \"%V\"", &ho->value);
4485        }
4486
4487        return rc;
4488    }
4489
4490    if (ho->value.data[0] != '/') {
4491        r->headers_out.location = ho;
4492    }
4493
4494    /*
4495     * we do not set r->headers_out.location here to avoid the handling
4496     * the local redirects without a host name by ngx_http_header_filter()
4497     */
4498
4499    return NGX_OK;
4500 }
4501
4502
4503 static ngx_int_t
4504 ngx_http_upstream_rewrite_refresh(ngx_http_request_t *r, ngx_table_elt_t *h,
4505     ngx_uint_t offset)
4506 {
4507     u_char           *p;
4508     ngx_int_t         rc;
4509     ngx_table_elt_t  *ho;
4510
4511     ho = ngx_list_push(&r->headers_out.headers);
4512     if (ho == NULL) {
4513         return NGX_ERROR;
4514     }
4515
4516     *ho = *h;
4517
4518     if (r->upstream->rewrite_redirect) {
4519
4520         p = ngx_strcasestrn(ho->value.data, "url=", 4 - 1);
4521
4522         if (p) {
4523             rc = r->upstream->rewrite_redirect(r, ho, p + 4 - ho->value.data);
4524
4525         } else {
4526             return NGX_OK;
4527         }
4528
4529         if (rc == NGX_DECLINED) {
4530             return NGX_OK;
4531         }
4532
4533         if (rc == NGX_OK) {
4534             r->headers_out.refresh = ho;
4535
4536             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
4537                            "rewritten refresh: \"%V\"", &ho->value);
4538         }
4539
4540         return rc;
4541     }
4542
4543     r->headers_out.refresh = ho;
4544
4545     return NGX_OK;
4546 }
4547
4548
4549 static ngx_int_t
4550 ngx_http_upstream_rewrite_set_cookie(ngx_http_request_t *r, ngx_table_elt_t *h,
4551     ngx_uint_t offset)
4552 {
4553     ngx_int_t         rc;
4554     ngx_table_elt_t  *ho;
4555
4556     ho = ngx_list_push(&r->headers_out.headers);
4557     if (ho == NULL) {
```

```
4558            return NGX_ERROR;
4559        }
4560
4561        *ho = *h;
4562
4563        if (r->upstream->rewrite_cookie) {
4564            rc = r->upstream->rewrite_cookie(r, ho);
4565
4566            if (rc == NGX_DECLINED) {
4567                return NGX_OK;
4568            }
4569
4570    #if (NGX_DEBUG)
4571            if (rc == NGX_OK) {
4572                ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
4573                               "rewritten cookie: \"%V\"", &ho->value);
4574            }
4575    #endif
4576
4577            return rc;
4578        }
4579
4580        return NGX_OK;
4581    }
4582
4583
4584    static ngx_int_t
4585    ngx_http_upstream_copy_allow_ranges(ngx_http_request_t *r,
4586        ngx_table_elt_t *h, ngx_uint_t offset)
4587    {
4588        ngx_table_elt_t  *ho;
4589
4590        if (r->upstream->conf->force_ranges) {
4591            return NGX_OK;
4592        }
4593
4594    #if (NGX_HTTP_CACHE)
4595
4596        if (r->cached) {
4597            r->allow_ranges = 1;
4598            return NGX_OK;
4599        }
4600
4601        if (r->upstream->cacheable) {
4602            r->allow_ranges = 1;
4603            r->single_range = 1;
4604            return NGX_OK;
4605        }
4606
4607    #endif
4608
4609        ho = ngx_list_push(&r->headers_out.headers);
4610        if (ho == NULL) {
4611            return NGX_ERROR;
4612        }
4613
4614        *ho = *h;
4615
4616        r->headers_out.accept_ranges = ho;
4617
4618        return NGX_OK;
4619    }
4620
4621
4622    #if (NGX_HTTP_GZIP)
4623
4624    static ngx_int_t
4625    ngx_http_upstream_copy_content_encoding(ngx_http_request_t *r,
4626        ngx_table_elt_t *h, ngx_uint_t offset)
4627    {
4628        ngx_table_elt_t  *ho;
4629
4630        ho = ngx_list_push(&r->headers_out.headers);
4631        if (ho == NULL) {
4632            return NGX_ERROR;
4633        }
```

```
4634
4635        *ho = *h;
4636
4637        r->headers_out.content_encoding = ho;
4638
4639        return NGX_OK;
4640    }
4641
4642    #endif
4643
4644
4645    static ngx_int_t
4646    ngx_http_upstream_add_variables(ngx_conf_t *cf)
4647    {
4648        ngx_http_variable_t  *var, *v;
4649
4650        for (v = ngx_http_upstream_vars; v->name.len; v++) {
4651            var = ngx_http_add_variable(cf, &v->name, v->flags);
4652            if (var == NULL) {
4653                return NGX_ERROR;
4654            }
4655
4656            var->get_handler = v->get_handler;
4657            var->data = v->data;
4658        }
4659
4660        return NGX_OK;
4661    }
4662
4663
4664    static ngx_int_t
4665    ngx_http_upstream_addr_variable(ngx_http_request_t *r,
4666        ngx_http_variable_value_t *v, uintptr_t data)
4667    {
4668        u_char                     *p;
4669        size_t                      len;
4670        ngx_uint_t                  i;
4671        ngx_http_upstream_state_t  *state;
4672
4673        v->valid = 1;
4674        v->no_cacheable = 0;
4675        v->not_found = 0;
4676
4677        if (r->upstream_states == NULL || r->upstream_states->nelts == 0) {
4678            v->not_found = 1;
4679            return NGX_OK;
4680        }
4681
4682        len = 0;
4683        state = r->upstream_states->elts;
4684
4685        for (i = 0; i < r->upstream_states->nelts; i++) {
4686            if (state[i].peer) {
4687                len += state[i].peer->len + 2;
4688
4689            } else {
4690                len += 3;
4691            }
4692        }
4693
4694        p = ngx_pnalloc(r->pool, len);
4695        if (p == NULL) {
4696            return NGX_ERROR;
4697        }
4698
4699        v->data = p;
4700
4701        i = 0;
4702
4703        for ( ;; ) {
4704            if (state[i].peer) {
4705                p = ngx_cpymem(p, state[i].peer->data, state[i].peer->len);
4706            }
4707
4708            if (++i == r->upstream_states->nelts) {
4709                break;
```

```
4710        }
4711
4712        if (state[i].peer) {
4713            *p++ = ',';
4714            *p++ = ' ';
4715
4716        } else {
4717            *p++ = ' ';
4718            *p++ = ':';
4719            *p++ = ' ';
4720
4721            if (++i == r->upstream_states->nelts) {
4722                break;
4723            }
4724
4725            continue;
4726        }
4727    }
4728
4729    v->len = p - v->data;
4730
4731    return NGX_OK;
4732 }
4733
4734
4735 static ngx_int_t
4736 ngx_http_upstream_status_variable(ngx_http_request_t *r,
4737     ngx_http_variable_value_t *v, uintptr_t data)
4738 {
4739     u_char                     *p;
4740     size_t                      len;
4741     ngx_uint_t                  i;
4742     ngx_http_upstream_state_t  *state;
4743
4744     v->valid = 1;
4745     v->no_cacheable = 0;
4746     v->not_found = 0;
4747
4748     if (r->upstream_states == NULL || r->upstream_states->nelts == 0) {
4749         v->not_found = 1;
4750         return NGX_OK;
4751     }
4752
4753     len = r->upstream_states->nelts * (3 + 2);
4754
4755     p = ngx_pnalloc(r->pool, len);
4756     if (p == NULL) {
4757         return NGX_ERROR;
4758     }
4759
4760     v->data = p;
4761
4762     i = 0;
4763     state = r->upstream_states->elts;
4764
4765     for ( ;; ) {
4766         if (state[i].status) {
4767             p = ngx_sprintf(p, "%ui", state[i].status);
4768
4769         } else {
4770             *p++ = '-';
4771         }
4772
4773         if (++i == r->upstream_states->nelts) {
4774             break;
4775         }
4776
4777         if (state[i].peer) {
4778             *p++ = ',';
4779             *p++ = ' ';
4780
4781         } else {
4782             *p++ = ' ';
4783             *p++ = ':';
4784             *p++ = ' ';
4785
```

```
            if (++i == r->upstream_states->nelts) {
                break;
            }

            continue;
        }
    }

    v->len = p - v->data;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_response_time_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char                     *p;
    size_t                      len;
    ngx_uint_t                  i;
    ngx_msec_int_t              ms;
    ngx_http_upstream_state_t  *state;

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    if (r->upstream_states == NULL || r->upstream_states->nelts == 0) {
        v->not_found = 1;
        return NGX_OK;
    }

    len = r->upstream_states->nelts * (NGX_TIME_T_LEN + 4 + 2);

    p = ngx_pnalloc(r->pool, len);
    if (p == NULL) {
        return NGX_ERROR;
    }

    v->data = p;

    i = 0;
    state = r->upstream_states->elts;

    for ( ;; ) {
        if (state[i].status) {

            if (data
                && state[i].header_sec != (time_t) NGX_ERROR)
            {
                ms = (ngx_msec_int_t)
                      (state[i].header_sec * 1000 + state[i].header_msec);

            } else {
                ms = (ngx_msec_int_t)
                      (state[i].response_sec * 1000 + state[i].response_msec);
            }

            ms = ngx_max(ms, 0);
            p = ngx_sprintf(p, "%T.%03M", (time_t) ms / 1000, ms % 1000);

        } else {
            *p++ = '-';
        }

        if (++i == r->upstream_states->nelts) {
            break;
        }

        if (state[i].peer) {
            *p++ = ',';
            *p++ = ' ';

        } else {
            *p++ = ' ';
```

```
            *p++ = ':';
            *p++ = ' ';

            if (++i == r->upstream_states->nelts) {
                break;
            }

            continue;
        }
    }

    v->len = p - v->data;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_response_length_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char                     *p;
    size_t                      len;
    ngx_uint_t                  i;
    ngx_http_upstream_state_t  *state;

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    if (r->upstream_states == NULL || r->upstream_states->nelts == 0) {
        v->not_found = 1;
        return NGX_OK;
    }

    len = r->upstream_states->nelts * (NGX_OFF_T_LEN + 2);

    p = ngx_pnalloc(r->pool, len);
    if (p == NULL) {
        return NGX_ERROR;
    }

    v->data = p;

    i = 0;
    state = r->upstream_states->elts;

    for ( ;; ) {
        p = ngx_sprintf(p, "%O", state[i].response_length);

        if (++i == r->upstream_states->nelts) {
            break;
        }

        if (state[i].peer) {
            *p++ = ',';
            *p++ = ' ';

        } else {
            *p++ = ' ';
            *p++ = ':';
            *p++ = ' ';

            if (++i == r->upstream_states->nelts) {
                break;
            }

            continue;
        }
    }

    v->len = p - v->data;

    return NGX_OK;
}

```

```
4938
4939 ngx_int_t
4940 ngx_http_upstream_header_variable(ngx_http_request_t *r,
4941     ngx_http_variable_value_t *v, uintptr_t data)
4942 {
4943     if (r->upstream == NULL) {
4944         v->not_found = 1;
4945         return NGX_OK;
4946     }
4947
4948     return ngx_http_variable_unknown_header(v, (ngx_str_t *) data,
4949                                             &r->upstream->headers_in.headers.part,
4950                                             sizeof("upstream_http_") - 1);
4951 }
4952
4953
4954 ngx_int_t
4955 ngx_http_upstream_cookie_variable(ngx_http_request_t *r,
4956     ngx_http_variable_value_t *v, uintptr_t data)
4957 {
4958     ngx_str_t  *name = (ngx_str_t *) data;
4959
4960     ngx_str_t   cookie, s;
4961
4962     if (r->upstream == NULL) {
4963         v->not_found = 1;
4964         return NGX_OK;
4965     }
4966
4967     s.len = name->len - (sizeof("upstream_cookie_") - 1);
4968     s.data = name->data + sizeof("upstream_cookie_") - 1;
4969
4970     if (ngx_http_parse_set_cookie_lines(&r->upstream->headers_in.cookies,
4971                                         &s, &cookie)
4972         == NGX_DECLINED)
4973     {
4974         v->not_found = 1;
4975         return NGX_OK;
4976     }
4977
4978     v->len = cookie.len;
4979     v->valid = 1;
4980     v->no_cacheable = 0;
4981     v->not_found = 0;
4982     v->data = cookie.data;
4983
4984     return NGX_OK;
4985 }
4986
4987
4988 #if (NGX_HTTP_CACHE)
4989
4990 ngx_int_t
4991 ngx_http_upstream_cache_status(ngx_http_request_t *r,
4992     ngx_http_variable_value_t *v, uintptr_t data)
4993 {
4994     ngx_uint_t  n;
4995
4996     if (r->upstream == NULL || r->upstream->cache_status == 0) {
4997         v->not_found = 1;
4998         return NGX_OK;
4999     }
5000
5001     n = r->upstream->cache_status - 1;
5002
5003     v->valid = 1;
5004     v->no_cacheable = 0;
5005     v->not_found = 0;
5006     v->len = ngx_http_cache_status[n].len;
5007     v->data = ngx_http_cache_status[n].data;
5008
5009     return NGX_OK;
5010 }
5011
5012
5013 static ngx_int_t
```

```
5014 ngx_http_upstream_cache_last_modified(ngx_http_request_t *r,
5015     ngx_http_variable_value_t *v, uintptr_t data)
5016 {
5017     u_char  *p;
5018
5019     if (r->upstream == NULL
5020         || !r->upstream->conf->cache_revalidate
5021         || r->upstream->cache_status != NGX_HTTP_CACHE_EXPIRED
5022         || r->cache->last_modified == -1)
5023     {
5024         v->not_found = 1;
5025         return NGX_OK;
5026     }
5027
5028     p = ngx_pnalloc(r->pool, sizeof("Mon, 28 Sep 1970 06:00:00 GMT") - 1);
5029     if (p == NULL) {
5030         return NGX_ERROR;
5031     }
5032
5033     v->len = ngx_http_time(p, r->cache->last_modified) - p;
5034     v->valid = 1;
5035     v->no_cacheable = 0;
5036     v->not_found = 0;
5037     v->data = p;
5038
5039     return NGX_OK;
5040 }
5041
5042
5043 static ngx_int_t
5044 ngx_http_upstream_cache_etag(ngx_http_request_t *r,
5045     ngx_http_variable_value_t *v, uintptr_t data)
5046 {
5047     if (r->upstream == NULL
5048         || !r->upstream->conf->cache_revalidate
5049         || r->upstream->cache_status != NGX_HTTP_CACHE_EXPIRED
5050         || r->cache->etag.len == 0)
5051     {
5052         v->not_found = 1;
5053         return NGX_OK;
5054     }
5055
5056     v->valid = 1;
5057     v->no_cacheable = 0;
5058     v->not_found = 0;
5059     v->len = r->cache->etag.len;
5060     v->data = r->cache->etag.data;
5061
5062     return NGX_OK;
5063 }
5064
5065 #endif
5066
5067
5068 static char *
5069 ngx_http_upstream(ngx_conf_t *cf, ngx_command_t *cmd, void *dummy)
5070 {
5071     char                        *rv;
5072     void                        *mconf;
5073     ngx_str_t                   *value;
5074     ngx_url_t                    u;
5075     ngx_uint_t                   m;
5076     ngx_conf_t                   pcf;
5077     ngx_http_module_t           *module;
5078     ngx_http_conf_ctx_t         *ctx, *http_ctx;
5079     ngx_http_upstream_srv_conf_t  *uscf;
5080
5081     ngx_memzero(&u, sizeof(ngx_url_t));
5082
5083     value = cf->args->elts;
5084     u.host = value[1];
5085     u.no_resolve = 1;
5086     u.no_port = 1;
5087
5088     uscf = ngx_http_upstream_add(cf, &u, NGX_HTTP_UPSTREAM_CREATE
5089                                        |NGX_HTTP_UPSTREAM_WEIGHT
```

```
                                      |NGX_HTTP_UPSTREAM_MAX_FAILS
                                      |NGX_HTTP_UPSTREAM_FAIL_TIMEOUT
                                      |NGX_HTTP_UPSTREAM_DOWN
                                      |NGX_HTTP_UPSTREAM_BACKUP);
    if (uscf == NULL) {
        return NGX_CONF_ERROR;
    }


    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    http_ctx = cf->ctx;
    ctx->main_conf = http_ctx->main_conf;

    /* the upstream{}'s srv_conf */

    ctx->srv_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->srv_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    ctx->srv_conf[ngx_http_upstream_module.ctx_index] = uscf;

    uscf->srv_conf = ctx->srv_conf;


    /* the upstream{}'s loc_conf */

    ctx->loc_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->loc_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    for (m = 0; ngx_modules[m]; m++) {
        if (ngx_modules[m]->type != NGX_HTTP_MODULE) {
            continue;
        }

        module = ngx_modules[m]->ctx;

        if (module->create_srv_conf) {
            mconf = module->create_srv_conf(cf);
            if (mconf == NULL) {
                return NGX_CONF_ERROR;
            }

            ctx->srv_conf[ngx_modules[m]->ctx_index] = mconf;
        }

        if (module->create_loc_conf) {
            mconf = module->create_loc_conf(cf);
            if (mconf == NULL) {
                return NGX_CONF_ERROR;
            }

            ctx->loc_conf[ngx_modules[m]->ctx_index] = mconf;
        }
    }

    uscf->servers = ngx_array_create(cf->pool, 4,
                                     sizeof(ngx_http_upstream_server_t));
    if (uscf->servers == NULL) {
        return NGX_CONF_ERROR;
    }


    /* parse inside upstream{} */

    pcf = *cf;
    cf->ctx = ctx;
    cf->cmd_type = NGX_HTTP_UPS_CONF;

    rv = ngx_conf_parse(cf, NULL);
```

```
5166
5167        *cf = pcf;
5168
5169        if (rv != NGX_CONF_OK) {
5170            return rv;
5171        }
5172
5173        if (uscf->servers->nelts == 0) {
5174            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5175                               "no servers are inside upstream");
5176            return NGX_CONF_ERROR;
5177        }
5178
5179        return rv;
5180    }
5181
5182
5183    static char *
5184    ngx_http_upstream_server(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
5185    {
5186        ngx_http_upstream_srv_conf_t  *uscf = conf;
5187
5188        time_t                       fail_timeout;
5189        ngx_str_t                   *value, s;
5190        ngx_url_t                    u;
5191        ngx_int_t                    weight, max_fails;
5192        ngx_uint_t                   i;
5193        ngx_http_upstream_server_t  *us;
5194
5195        us = ngx_array_push(uscf->servers);
5196        if (us == NULL) {
5197            return NGX_CONF_ERROR;
5198        }
5199
5200        ngx_memzero(us, sizeof(ngx_http_upstream_server_t));
5201
5202        value = cf->args->elts;
5203
5204        weight = 1;
5205        max_fails = 1;
5206        fail_timeout = 10;
5207
5208        for (i = 2; i < cf->args->nelts; i++) {
5209
5210            if (ngx_strncmp(value[i].data, "weight=", 7) == 0) {
5211
5212                if (!(uscf->flags & NGX_HTTP_UPSTREAM_WEIGHT)) {
5213                    goto not_supported;
5214                }
5215
5216                weight = ngx_atoi(&value[i].data[7], value[i].len - 7);
5217
5218                if (weight == NGX_ERROR || weight == 0) {
5219                    goto invalid;
5220                }
5221
5222                continue;
5223            }
5224
5225            if (ngx_strncmp(value[i].data, "max_fails=", 10) == 0) {
5226
5227                if (!(uscf->flags & NGX_HTTP_UPSTREAM_MAX_FAILS)) {
5228                    goto not_supported;
5229                }
5230
5231                max_fails = ngx_atoi(&value[i].data[10], value[i].len - 10);
5232
5233                if (max_fails == NGX_ERROR) {
5234                    goto invalid;
5235                }
5236
5237                continue;
5238            }
5239
5240            if (ngx_strncmp(value[i].data, "fail_timeout=", 13) == 0) {
5241
```

```
5242            if (!(uscf->flags & NGX_HTTP_UPSTREAM_FAIL_TIMEOUT)) {
5243                goto not_supported;
5244            }
5245
5246            s.len = value[i].len - 13;
5247            s.data = &value[i].data[13];
5248
5249            fail_timeout = ngx_parse_time(&s, 1);
5250
5251            if (fail_timeout == (time_t) NGX_ERROR) {
5252                goto invalid;
5253            }
5254
5255            continue;
5256        }
5257
5258        if (ngx_strcmp(value[i].data, "backup") == 0) {
5259
5260            if (!(uscf->flags & NGX_HTTP_UPSTREAM_BACKUP)) {
5261                goto not_supported;
5262            }
5263
5264            us->backup = 1;
5265
5266            continue;
5267        }
5268
5269        if (ngx_strcmp(value[i].data, "down") == 0) {
5270
5271            if (!(uscf->flags & NGX_HTTP_UPSTREAM_DOWN)) {
5272                goto not_supported;
5273            }
5274
5275            us->down = 1;
5276
5277            continue;
5278        }
5279
5280        goto invalid;
5281    }
5282
5283    ngx_memzero(&u, sizeof(ngx_url_t));
5284
5285    u.url = value[1];
5286    u.default_port = 80;
5287
5288    if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
5289        if (u.err) {
5290            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5291                               "%s in upstream \"%V\"", u.err, &u.url);
5292        }
5293
5294        return NGX_CONF_ERROR;
5295    }
5296
5297    us->name = u.url;
5298    us->addrs = u.addrs;
5299    us->naddrs = u.naddrs;
5300    us->weight = weight;
5301    us->max_fails = max_fails;
5302    us->fail_timeout = fail_timeout;
5303
5304    return NGX_CONF_OK;
5305
5306 invalid:
5307
5308    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5309                       "invalid parameter \"%V\"", &value[i]);
5310
5311    return NGX_CONF_ERROR;
5312
5313 not_supported:
5314
5315    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5316                       "balancing method does not support parameter \"%V\"",
5317                       &value[i]);
```

```
5318
5319        return NGX_CONF_ERROR;
5320    }
5321
5322
5323    ngx_http_upstream_srv_conf_t *
5324    ngx_http_upstream_add(ngx_conf_t *cf, ngx_url_t *u, ngx_uint_t flags)
5325    {
5326        ngx_uint_t                      i;
5327        ngx_http_upstream_server_t     *us;
5328        ngx_http_upstream_srv_conf_t   *uscf, **uscfp;
5329        ngx_http_upstream_main_conf_t  *umcf;
5330
5331        if (!(flags & NGX_HTTP_UPSTREAM_CREATE)) {
5332
5333            if (ngx_parse_url(cf->pool, u) != NGX_OK) {
5334                if (u->err) {
5335                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5336                                       "%s in upstream \"%V\"", u->err, &u->url);
5337                }
5338
5339                return NULL;
5340            }
5341        }
5342
5343        umcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_upstream_module);
5344
5345        uscfp = umcf->upstreams.elts;
5346
5347        for (i = 0; i < umcf->upstreams.nelts; i++) {
5348
5349            if (uscfp[i]->host.len != u->host.len
5350                || ngx_strncasecmp(uscfp[i]->host.data, u->host.data, u->host.len)
5351                   != 0)
5352            {
5353                continue;
5354            }
5355
5356            if ((flags & NGX_HTTP_UPSTREAM_CREATE)
5357                 && (uscfp[i]->flags & NGX_HTTP_UPSTREAM_CREATE))
5358            {
5359                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5360                                   "duplicate upstream \"%V\"", &u->host);
5361                return NULL;
5362            }
5363
5364            if ((uscfp[i]->flags & NGX_HTTP_UPSTREAM_CREATE) && !u->no_port) {
5365                ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
5366                                   "upstream \"%V\" may not have port %d",
5367                                   &u->host, u->port);
5368                return NULL;
5369            }
5370
5371            if ((flags & NGX_HTTP_UPSTREAM_CREATE) && !uscfp[i]->no_port) {
5372                ngx_log_error(NGX_LOG_WARN, cf->log, 0,
5373                              "upstream \"%V\" may not have port %d in %s:%ui",
5374                              &u->host, uscfp[i]->port,
5375                              uscfp[i]->file_name, uscfp[i]->line);
5376                return NULL;
5377            }
5378
5379            if (uscfp[i]->port && u->port
5380                && uscfp[i]->port != u->port)
5381            {
5382                continue;
5383            }
5384
5385            if (uscfp[i]->default_port && u->default_port
5386                && uscfp[i]->default_port != u->default_port)
5387            {
5388                continue;
5389            }
5390
5391            if (flags & NGX_HTTP_UPSTREAM_CREATE) {
5392                uscfp[i]->flags = flags;
5393            }
```

```
5394
5395            return uscfp[i];
5396        }
5397
5398        uscf = ngx_pcalloc(cf->pool, sizeof(ngx_http_upstream_srv_conf_t));
5399        if (uscf == NULL) {
5400            return NULL;
5401        }
5402
5403        uscf->flags = flags;
5404        uscf->host = u->host;
5405        uscf->file_name = cf->conf_file->file.name.data;
5406        uscf->line = cf->conf_file->line;
5407        uscf->port = u->port;
5408        uscf->default_port = u->default_port;
5409        uscf->no_port = u->no_port;
5410
5411        if (u->naddrs == 1) {
5412            uscf->servers = ngx_array_create(cf->pool, 1,
5413                                             sizeof(ngx_http_upstream_server_t));
5414            if (uscf->servers == NULL) {
5415                return NULL;
5416            }
5417
5418            us = ngx_array_push(uscf->servers);
5419            if (us == NULL) {
5420                return NULL;
5421            }
5422
5423            ngx_memzero(us, sizeof(ngx_http_upstream_server_t));
5424
5425            us->addrs = u->addrs;
5426            us->naddrs = 1;
5427        }
5428
5429        uscfp = ngx_array_push(&umcf->upstreams);
5430        if (uscfp == NULL) {
5431            return NULL;
5432        }
5433
5434        *uscfp = uscf;
5435
5436        return uscf;
5437    }
5438
5439
5440    char *
5441    ngx_http_upstream_bind_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
5442        void *conf)
5443    {
5444        char  *p = conf;
5445
5446        ngx_int_t                           rc;
5447        ngx_str_t                          *value;
5448        ngx_http_complex_value_t            cv;
5449        ngx_http_upstream_local_t         **plocal, *local;
5450        ngx_http_compile_complex_value_t    ccv;
5451
5452        plocal = (ngx_http_upstream_local_t **) (p + cmd->offset);
5453
5454        if (*plocal != NGX_CONF_UNSET_PTR) {
5455            return "is duplicate";
5456        }
5457
5458        value = cf->args->elts;
5459
5460        if (ngx_strcmp(value[1].data, "off") == 0) {
5461            *plocal = NULL;
5462            return NGX_CONF_OK;
5463        }
5464
5465        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
5466
5467        ccv.cf = cf;
5468        ccv.value = &value[1];
5469        ccv.complex_value = &cv;
```

```
5470
5471        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
5472            return NGX_CONF_ERROR;
5473        }
5474
5475        local = ngx_pcalloc(cf->pool, sizeof(ngx_http_upstream_local_t));
5476        if (local == NULL) {
5477            return NGX_CONF_ERROR;
5478        }
5479
5480        *plocal = local;
5481
5482        if (cv.lengths) {
5483            local->value = ngx_palloc(cf->pool, sizeof(ngx_http_complex_value_t));
5484            if (local->value == NULL) {
5485                return NGX_CONF_ERROR;
5486            }
5487
5488            *local->value = cv;
5489
5490            return NGX_CONF_OK;
5491        }
5492
5493        local->addr = ngx_palloc(cf->pool, sizeof(ngx_addr_t));
5494        if (local->addr == NULL) {
5495            return NGX_CONF_ERROR;
5496        }
5497
5498        rc = ngx_parse_addr(cf->pool, local->addr, value[1].data, value[1].len);
5499
5500        switch (rc) {
5501        case NGX_OK:
5502            local->addr->name = value[1];
5503            return NGX_CONF_OK;
5504
5505        case NGX_DECLINED:
5506            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5507                               "invalid address \"%V\"", &value[1]);
5508            /* fall through */
5509
5510        default:
5511            return NGX_CONF_ERROR;
5512        }
5513    }
5514
5515
5516    static ngx_addr_t *
5517    ngx_http_upstream_get_local(ngx_http_request_t *r,
5518        ngx_http_upstream_local_t *local)
5519    {
5520        ngx_int_t    rc;
5521        ngx_str_t    val;
5522        ngx_addr_t  *addr;
5523
5524        if (local == NULL) {
5525            return NULL;
5526        }
5527
5528        if (local->value == NULL) {
5529            return local->addr;
5530        }
5531
5532        if (ngx_http_complex_value(r, local->value, &val) != NGX_OK) {
5533            return NULL;
5534        }
5535
5536        if (val.len == 0) {
5537            return NULL;
5538        }
5539
5540        addr = ngx_palloc(r->pool, sizeof(ngx_addr_t));
5541        if (addr == NULL) {
5542            return NULL;
5543        }
5544
5545        rc = ngx_parse_addr(r->pool, addr, val.data, val.len);
```

```
5546
5547        switch (rc) {
5548        case NGX_OK:
5549            addr->name = val;
5550            return addr;
5551
5552        case NGX_DECLINED:
5553            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
5554                          "invalid local address \"%V\"", &val);
5555            /* fall through */
5556
5557        default:
5558            return NULL;
5559        }
5560    }
5561
5562
5563    char *
5564    ngx_http_upstream_param_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
5565        void *conf)
5566    {
5567        char  *p = conf;
5568
5569        ngx_str_t                   *value;
5570        ngx_array_t                **a;
5571        ngx_http_upstream_param_t   *param;
5572
5573        a = (ngx_array_t **) (p + cmd->offset);
5574
5575        if (*a == NULL) {
5576            *a = ngx_array_create(cf->pool, 4, sizeof(ngx_http_upstream_param_t));
5577            if (*a == NULL) {
5578                return NGX_CONF_ERROR;
5579            }
5580        }
5581
5582        param = ngx_array_push(*a);
5583        if (param == NULL) {
5584            return NGX_CONF_ERROR;
5585        }
5586
5587        value = cf->args->elts;
5588
5589        param->key = value[1];
5590        param->value = value[2];
5591        param->skip_empty = 0;
5592
5593        if (cf->args->nelts == 4) {
5594            if (ngx_strcmp(value[3].data, "if_not_empty") != 0) {
5595                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
5596                                   "invalid parameter \"%V\"", &value[3]);
5597                return NGX_CONF_ERROR;
5598            }
5599
5600            param->skip_empty = 1;
5601        }
5602
5603        return NGX_CONF_OK;
5604    }
5605
5606
5607    ngx_int_t
5608    ngx_http_upstream_hide_headers_hash(ngx_conf_t *cf,
5609        ngx_http_upstream_conf_t *conf, ngx_http_upstream_conf_t *prev,
5610        ngx_str_t *default_hide_headers, ngx_hash_init_t *hash)
5611    {
5612        ngx_str_t       *h;
5613        ngx_uint_t       i, j;
5614        ngx_array_t      hide_headers;
5615        ngx_hash_key_t  *hk;
5616
5617        if (conf->hide_headers == NGX_CONF_UNSET_PTR
5618            && conf->pass_headers == NGX_CONF_UNSET_PTR)
5619        {
5620            conf->hide_headers = prev->hide_headers;
5621            conf->pass_headers = prev->pass_headers;
```

```
5622
5623            conf->hide_headers_hash = prev->hide_headers_hash;
5624
5625            if (conf->hide_headers_hash.buckets
5626    #if (NGX_HTTP_CACHE)
5627                && ((conf->cache == 0) == (prev->cache == 0))
5628    #endif
5629               )
5630            {
5631                return NGX_OK;
5632            }
5633
5634        } else {
5635            if (conf->hide_headers == NGX_CONF_UNSET_PTR) {
5636                conf->hide_headers = prev->hide_headers;
5637            }
5638
5639            if (conf->pass_headers == NGX_CONF_UNSET_PTR) {
5640                conf->pass_headers = prev->pass_headers;
5641            }
5642        }
5643
5644        if (ngx_array_init(&hide_headers, cf->temp_pool, 4, sizeof(ngx_hash_key_t))
5645            != NGX_OK)
5646        {
5647            return NGX_ERROR;
5648        }
5649
5650        for (h = default_hide_headers; h->len; h++) {
5651            hk = ngx_array_push(&hide_headers);
5652            if (hk == NULL) {
5653                return NGX_ERROR;
5654            }
5655
5656            hk->key = *h;
5657            hk->key_hash = ngx_hash_key_lc(h->data, h->len);
5658            hk->value = (void *) 1;
5659        }
5660
5661        if (conf->hide_headers != NGX_CONF_UNSET_PTR) {
5662
5663            h = conf->hide_headers->elts;
5664
5665            for (i = 0; i < conf->hide_headers->nelts; i++) {
5666
5667                hk = hide_headers.elts;
5668
5669                for (j = 0; j < hide_headers.nelts; j++) {
5670                    if (ngx_strcasecmp(h[i].data, hk[j].key.data) == 0) {
5671                        goto exist;
5672                    }
5673                }
5674
5675                hk = ngx_array_push(&hide_headers);
5676                if (hk == NULL) {
5677                    return NGX_ERROR;
5678                }
5679
5680                hk->key = h[i];
5681                hk->key_hash = ngx_hash_key_lc(h[i].data, h[i].len);
5682                hk->value = (void *) 1;
5683
5684            exist:
5685
5686                continue;
5687            }
5688        }
5689
5690        if (conf->pass_headers != NGX_CONF_UNSET_PTR) {
5691
5692            h = conf->pass_headers->elts;
5693            hk = hide_headers.elts;
5694
5695            for (i = 0; i < conf->pass_headers->nelts; i++) {
5696                for (j = 0; j < hide_headers.nelts; j++) {
5697
```

```
                if (hk[j].key.data == NULL) {
                    continue;
                }

                if (ngx_strcasecmp(h[i].data, hk[j].key.data) == 0) {
                    hk[j].key.data = NULL;
                    break;
                }
            }
        }
    }

    hash->hash = &conf->hide_headers_hash;
    hash->key = ngx_hash_key_lc;
    hash->pool = cf->pool;
    hash->temp_pool = NULL;

    return ngx_hash_init(hash, hide_headers.elts, hide_headers.nelts);
}


static void *
ngx_http_upstream_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_upstream_main_conf_t  *umcf;

    umcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_upstream_main_conf_t));
    if (umcf == NULL) {
        return NULL;
    }

    if (ngx_array_init(&umcf->upstreams, cf->pool, 4,
                       sizeof(ngx_http_upstream_srv_conf_t *))
        != NGX_OK)
    {
        return NULL;
    }

    return umcf;
}


static char *
ngx_http_upstream_init_main_conf(ngx_conf_t *cf, void *conf)
{
    ngx_http_upstream_main_conf_t  *umcf = conf;

    ngx_uint_t                      i;
    ngx_array_t                     headers_in;
    ngx_hash_key_t                 *hk;
    ngx_hash_init_t                 hash;
    ngx_http_upstream_init_pt       init;
    ngx_http_upstream_header_t     *header;
    ngx_http_upstream_srv_conf_t  **uscfp;

    uscfp = umcf->upstreams.elts;

    for (i = 0; i < umcf->upstreams.nelts; i++) {

        init = uscfp[i]->peer.init_upstream ? uscfp[i]->peer.init_upstream:
                                              ngx_http_upstream_init_round_robin;

        if (init(cf, uscfp[i]) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }


    /* upstream_headers_in_hash */

    if (ngx_array_init(&headers_in, cf->temp_pool, 32, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

```

```
5774     for (header = ngx_http_upstream_headers_in; header->name.len; header++) {
5775         hk = ngx_array_push(&headers_in);
5776         if (hk == NULL) {
5777             return NGX_CONF_ERROR;
5778         }
5779 
5780         hk->key = header->name;
5781         hk->key_hash = ngx_hash_key_lc(header->name.data, header->name.len);
5782         hk->value = header;
5783     }
5784 
5785     hash.hash = &umcf->headers_in_hash;
5786     hash.key = ngx_hash_key_lc;
5787     hash.max_size = 512;
5788     hash.bucket_size = ngx_align(64, ngx_cacheline_size);
5789     hash.name = "upstream_headers_in_hash";
5790     hash.pool = cf->pool;
5791     hash.temp_pool = NULL;
5792 
5793     if (ngx_hash_init(&hash, headers_in.elts, headers_in.nelts) != NGX_OK) {
5794         return NGX_CONF_ERROR;
5795     }
5796 
5797     return NGX_CONF_OK;
5798 }
```

One Level Up Top Level

src/http/ngx_http_upstream.h - nginx-1.7.10

Data types defined

- ngx_http_upstream_conf_t
- ngx_http_upstream_handler_pt
- ngx_http_upstream_header_t
- ngx_http_upstream_headers_in_t
- ngx_http_upstream_init_peer_pt
- ngx_http_upstream_init_pt
- ngx_http_upstream_local_t
- ngx_http_upstream_main_conf_t
- ngx_http_upstream_next_t
- ngx_http_upstream_param_t
- ngx_http_upstream_peer_t
- ngx_http_upstream_resolved_t
- ngx_http_upstream_s
- ngx_http_upstream_server_t
- ngx_http_upstream_srv_conf_s
- ngx_http_upstream_srv_conf_t
- ngx_http_upstream_state_t

Macros defined

- NGX_HTTP_UPSTREAM_BACKUP
- NGX_HTTP_UPSTREAM_CREATE
- NGX_HTTP_UPSTREAM_DOWN
- NGX_HTTP_UPSTREAM_FAIL_TIMEOUT
- NGX_HTTP_UPSTREAM_FT_BUSY_LOCK
- NGX_HTTP_UPSTREAM_FT_ERROR
- NGX_HTTP_UPSTREAM_FT_HTTP_403
- NGX_HTTP_UPSTREAM_FT_HTTP_404
- NGX_HTTP_UPSTREAM_FT_HTTP_500
- NGX_HTTP_UPSTREAM_FT_HTTP_502
- NGX_HTTP_UPSTREAM_FT_HTTP_503

- NGX_HTTP_UPSTREAM_FT_HTTP_504
- NGX_HTTP_UPSTREAM_FT_INVALID_HEADER
- NGX_HTTP_UPSTREAM_FT_MAX_WAITING
- NGX_HTTP_UPSTREAM_FT_NOLIVE
- NGX_HTTP_UPSTREAM_FT_OFF
- NGX_HTTP_UPSTREAM_FT_STATUS
- NGX_HTTP_UPSTREAM_FT_TIMEOUT
- NGX_HTTP_UPSTREAM_FT_UPDATING
- NGX_HTTP_UPSTREAM_IGN_CACHE_CONTROL
- NGX_HTTP_UPSTREAM_IGN_EXPIRES
- NGX_HTTP_UPSTREAM_IGN_SET_COOKIE
- NGX_HTTP_UPSTREAM_IGN_VARY
- NGX_HTTP_UPSTREAM_IGN_XA_BUFFERING
- NGX_HTTP_UPSTREAM_IGN_XA_CHARSET
- NGX_HTTP_UPSTREAM_IGN_XA_EXPIRES
- NGX_HTTP_UPSTREAM_IGN_XA_LIMIT_RATE
- NGX_HTTP_UPSTREAM_IGN_XA_REDIRECT
- NGX_HTTP_UPSTREAM_INVALID_HEADER
- NGX_HTTP_UPSTREAM_MAX_FAILS
- NGX_HTTP_UPSTREAM_WEIGHT
- _NGX_HTTP_UPSTREAM_H_INCLUDED_
- ngx_http_conf_upstream_srv_conf

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_HTTP_UPSTREAM_H_INCLUDED_
9  #define _NGX_HTTP_UPSTREAM_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15  #include <ngx_event_connect.h>
16  #include <ngx_event_pipe.h>
17  #include <ngx_http.h>
18
19
20  #define NGX_HTTP_UPSTREAM_FT_ERROR           0x00000002
21  #define NGX_HTTP_UPSTREAM_FT_TIMEOUT         0x00000004
22  #define NGX_HTTP_UPSTREAM_FT_INVALID_HEADER  0x00000008
```

```
#define NGX_HTTP_UPSTREAM_FT_HTTP_500        0x00000010
#define NGX_HTTP_UPSTREAM_FT_HTTP_502        0x00000020
#define NGX_HTTP_UPSTREAM_FT_HTTP_503        0x00000040
#define NGX_HTTP_UPSTREAM_FT_HTTP_504        0x00000080
#define NGX_HTTP_UPSTREAM_FT_HTTP_403        0x00000100
#define NGX_HTTP_UPSTREAM_FT_HTTP_404        0x00000200
#define NGX_HTTP_UPSTREAM_FT_UPDATING        0x00000400
#define NGX_HTTP_UPSTREAM_FT_BUSY_LOCK       0x00000800
#define NGX_HTTP_UPSTREAM_FT_MAX_WAITING     0x00001000
#define NGX_HTTP_UPSTREAM_FT_NOLIVE          0x40000000
#define NGX_HTTP_UPSTREAM_FT_OFF             0x80000000

#define NGX_HTTP_UPSTREAM_FT_STATUS          (NGX_HTTP_UPSTREAM_FT_HTTP_500  \
                                             |NGX_HTTP_UPSTREAM_FT_HTTP_502  \
                                             |NGX_HTTP_UPSTREAM_FT_HTTP_503  \
                                             |NGX_HTTP_UPSTREAM_FT_HTTP_504  \
                                             |NGX_HTTP_UPSTREAM_FT_HTTP_403  \
                                             |NGX_HTTP_UPSTREAM_FT_HTTP_404)

#define NGX_HTTP_UPSTREAM_INVALID_HEADER     40


#define NGX_HTTP_UPSTREAM_IGN_XA_REDIRECT    0x00000002
#define NGX_HTTP_UPSTREAM_IGN_XA_EXPIRES     0x00000004
#define NGX_HTTP_UPSTREAM_IGN_EXPIRES        0x00000008
#define NGX_HTTP_UPSTREAM_IGN_CACHE_CONTROL  0x00000010
#define NGX_HTTP_UPSTREAM_IGN_SET_COOKIE     0x00000020
#define NGX_HTTP_UPSTREAM_IGN_XA_LIMIT_RATE  0x00000040
#define NGX_HTTP_UPSTREAM_IGN_XA_BUFFERING   0x00000080
#define NGX_HTTP_UPSTREAM_IGN_XA_CHARSET     0x00000100
#define NGX_HTTP_UPSTREAM_IGN_VARY           0x00000200


typedef struct {
    ngx_msec_t                       bl_time;
    ngx_uint_t                       bl_state;

    ngx_uint_t                       status;
    time_t                           response_sec;
    ngx_uint_t                       response_msec;
    time_t                           header_sec;
    ngx_uint_t                       header_msec;
    off_t                            response_length;

    ngx_str_t                       *peer;
} ngx_http_upstream_state_t;


typedef struct {
    ngx_hash_t                       headers_in_hash;
    ngx_array_t                      upstreams;
                                     /* ngx_http_upstream_srv_conf_t */
} ngx_http_upstream_main_conf_t;

typedef struct ngx_http_upstream_srv_conf_s  ngx_http_upstream_srv_conf_t;

typedef ngx_int_t (*ngx_http_upstream_init_pt)(ngx_conf_t *cf,
    ngx_http_upstream_srv_conf_t *us);
typedef ngx_int_t (*ngx_http_upstream_init_peer_pt)(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us);


typedef struct {
    ngx_http_upstream_init_pt        init_upstream;
    ngx_http_upstream_init_peer_pt   init;
    void                            *data;
} ngx_http_upstream_peer_t;


typedef struct {
    ngx_str_t                        name;
    ngx_addr_t                      *addrs;
    ngx_uint_t                       naddrs;
    ngx_uint_t                       weight;
    ngx_uint_t                       max_fails;
    time_t                           fail_timeout;
```

```
99
100     unsigned                         down:1;
101     unsigned                         backup:1;
102 } ngx_http_upstream_server_t;
103
104
105 #define NGX_HTTP_UPSTREAM_CREATE        0x0001
106 #define NGX_HTTP_UPSTREAM_WEIGHT        0x0002
107 #define NGX_HTTP_UPSTREAM_MAX_FAILS     0x0004
108 #define NGX_HTTP_UPSTREAM_FAIL_TIMEOUT  0x0008
109 #define NGX_HTTP_UPSTREAM_DOWN          0x0010
110 #define NGX_HTTP_UPSTREAM_BACKUP        0x0020
111
112
113 struct ngx_http_upstream_srv_conf_s {
114     ngx_http_upstream_peer_t         peer;
115     void                           **srv_conf;
116
117     ngx_array_t                     *servers;  /* ngx_http_upstream_server_t */
118
119     ngx_uint_t                       flags;
120     ngx_str_t                        host;
121     u_char                          *file_name;
122     ngx_uint_t                       line;
123     in_port_t                        port;
124     in_port_t                        default_port;
125     ngx_uint_t                       no_port;  /* unsigned no_port:1 */
126 };
127
128
129 typedef struct {
130     ngx_addr_t                      *addr;
131     ngx_http_complex_value_t        *value;
132 } ngx_http_upstream_local_t;
133
134
135 typedef struct {
136     ngx_http_upstream_srv_conf_t    *upstream;
137
138     ngx_msec_t                       connect_timeout;
139     ngx_msec_t                       send_timeout;
140     ngx_msec_t                       read_timeout;
141     ngx_msec_t                       timeout;
142     ngx_msec_t                       next_upstream_timeout;
143
144     size_t                           send_lowat;
145     size_t                           buffer_size;
146     size_t                           limit_rate;
147
148     size_t                           busy_buffers_size;
149     size_t                           max_temp_file_size;
150     size_t                           temp_file_write_size;
151
152     size_t                           busy_buffers_size_conf;
153     size_t                           max_temp_file_size_conf;
154     size_t                           temp_file_write_size_conf;
155
156     ngx_bufs_t                       bufs;
157
158     ngx_uint_t                       ignore_headers;
159     ngx_uint_t                       next_upstream;
160     ngx_uint_t                       store_access;
161     ngx_uint_t                       next_upstream_tries;
162     ngx_flag_t                       buffering;
163     ngx_flag_t                       pass_request_headers;
164     ngx_flag_t                       pass_request_body;
165
166     ngx_flag_t                       ignore_client_abort;
167     ngx_flag_t                       intercept_errors;
168     ngx_flag_t                       cyclic_temp_file;
169     ngx_flag_t                       force_ranges;
170
171     ngx_path_t                      *temp_path;
172
173     ngx_hash_t                       hide_headers_hash;
174     ngx_array_t                     *hide_headers;
```

```
    ngx_array_t                     *pass_headers;

    ngx_http_upstream_local_t       *local;

#if (NGX_HTTP_CACHE)
    ngx_shm_zone_t                  *cache_zone;
    ngx_http_complex_value_t        *cache_value;

    ngx_uint_t                       cache_min_uses;
    ngx_uint_t                       cache_use_stale;
    ngx_uint_t                       cache_methods;

    ngx_flag_t                       cache_lock;
    ngx_msec_t                       cache_lock_timeout;
    ngx_msec_t                       cache_lock_age;

    ngx_flag_t                       cache_revalidate;

    ngx_array_t                     *cache_valid;
    ngx_array_t                     *cache_bypass;
    ngx_array_t                     *no_cache;
#endif

    ngx_array_t                     *store_lengths;
    ngx_array_t                     *store_values;

#if (NGX_HTTP_CACHE)
    signed                           cache:2;
#endif
    signed                           store:2;
    unsigned                         intercept_404:1;
    unsigned                         change_buffering:1;

#if (NGX_HTTP_SSL)
    ngx_ssl_t                       *ssl;
    ngx_flag_t                       ssl_session_reuse;

    ngx_http_complex_value_t        *ssl_name;
    ngx_flag_t                       ssl_server_name;
    ngx_flag_t                       ssl_verify;
#endif

    ngx_str_t                        module;
} ngx_http_upstream_conf_t;


typedef struct {
    ngx_str_t                        name;
    ngx_http_header_handler_pt       handler;
    ngx_uint_t                       offset;
    ngx_http_header_handler_pt       copy_handler;
    ngx_uint_t                       conf;
    ngx_uint_t                       redirect;  /* unsigned   redirect:1; */
} ngx_http_upstream_header_t;


typedef struct {
    ngx_list_t                       headers;

    ngx_uint_t                       status_n;
    ngx_str_t                        status_line;

    ngx_table_elt_t                 *status;
    ngx_table_elt_t                 *date;
    ngx_table_elt_t                 *server;
    ngx_table_elt_t                 *connection;

    ngx_table_elt_t                 *expires;
    ngx_table_elt_t                 *etag;
    ngx_table_elt_t                 *x_accel_expires;
    ngx_table_elt_t                 *x_accel_redirect;
    ngx_table_elt_t                 *x_accel_limit_rate;

    ngx_table_elt_t                 *content_type;
    ngx_table_elt_t                 *content_length;

```

```
    ngx_table_elt_t                 *last_modified;
    ngx_table_elt_t                 *location;
    ngx_table_elt_t                 *accept_ranges;
    ngx_table_elt_t                 *www_authenticate;
    ngx_table_elt_t                 *transfer_encoding;
    ngx_table_elt_t                 *vary;

#if (NGX_HTTP_GZIP)
    ngx_table_elt_t                 *content_encoding;
#endif

    ngx_array_t                      cache_control;
    ngx_array_t                      cookies;

    off_t                            content_length_n;
    time_t                           last_modified_time;

    unsigned                         connection_close:1;
    unsigned                         chunked:1;
} ngx_http_upstream_headers_in_t;


typedef struct {
    ngx_str_t                        host;
    in_port_t                        port;
    ngx_uint_t                       no_port; /* unsigned no_port:1 */

    ngx_uint_t                       naddrs;
    ngx_addr_t                      *addrs;

    struct sockaddr                 *sockaddr;
    socklen_t                        socklen;

    ngx_resolver_ctx_t              *ctx;
} ngx_http_upstream_resolved_t;


typedef void (*ngx_http_upstream_handler_pt)(ngx_http_request_t *r,
    ngx_http_upstream_t *u);


struct ngx_http_upstream_s {
    ngx_http_upstream_handler_pt     read_event_handler;
    ngx_http_upstream_handler_pt     write_event_handler;

    ngx_peer_connection_t            peer;

    ngx_event_pipe_t                *pipe;

    ngx_chain_t                     *request_bufs;

    ngx_output_chain_ctx_t           output;
    ngx_chain_writer_ctx_t           writer;

    ngx_http_upstream_conf_t        *conf;
#if (NGX_HTTP_CACHE)
    ngx_array_t                     *caches;
#endif

    ngx_http_upstream_headers_in_t   headers_in;

    ngx_http_upstream_resolved_t    *resolved;

    ngx_buf_t                        from_client;

    ngx_buf_t                        buffer;
    off_t                            length;

    ngx_chain_t                     *out_bufs;
    ngx_chain_t                     *busy_bufs;
    ngx_chain_t                     *free_bufs;

    ngx_int_t                       (*input_filter_init)(void *data);
    ngx_int_t                       (*input_filter)(void *data, ssize_t bytes);
    void                             *input_filter_ctx;

```

```
#if (NGX_HTTP_CACHE)
    ngx_int_t                      (*create_key)(ngx_http_request_t *r);
#endif
    ngx_int_t                      (*create_request)(ngx_http_request_t *r);
    ngx_int_t                      (*reinit_request)(ngx_http_request_t *r);
    ngx_int_t                      (*process_header)(ngx_http_request_t *r);
    void                           (*abort_request)(ngx_http_request_t *r);
    void                           (*finalize_request)(ngx_http_request_t *r,
                                         ngx_int_t rc);
    ngx_int_t                      (*rewrite_redirect)(ngx_http_request_t *r,
                                         ngx_table_elt_t *h, size_t prefix);
    ngx_int_t                      (*rewrite_cookie)(ngx_http_request_t *r,
                                         ngx_table_elt_t *h);

    ngx_msec_t                      timeout;

    ngx_http_upstream_state_t      *state;

    ngx_str_t                       method;
    ngx_str_t                       schema;
    ngx_str_t                       uri;

#if (NGX_HTTP_SSL)
    ngx_str_t                       ssl_name;
#endif

    ngx_http_cleanup_pt            *cleanup;

    unsigned                        store:1;
    unsigned                        cacheable:1;
    unsigned                        accel:1;
    unsigned                        ssl:1;
#if (NGX_HTTP_CACHE)
    unsigned                        cache_status:3;
#endif

    unsigned                        buffering:1;
    unsigned                        keepalive:1;
    unsigned                        upgrade:1;

    unsigned                        request_sent:1;
    unsigned                        header_sent:1;
};


typedef struct {
    ngx_uint_t                     status;
    ngx_uint_t                     mask;
} ngx_http_upstream_next_t;


typedef struct {
    ngx_str_t   key;
    ngx_str_t   value;
    ngx_uint_t  skip_empty;
} ngx_http_upstream_param_t;


ngx_int_t ngx_http_upstream_cookie_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
ngx_int_t ngx_http_upstream_header_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);

ngx_int_t ngx_http_upstream_create(ngx_http_request_t *r);
void ngx_http_upstream_init(ngx_http_request_t *r);
ngx_http_upstream_srv_conf_t *ngx_http_upstream_add(ngx_conf_t *cf,
    ngx_url_t *u, ngx_uint_t flags);
char *ngx_http_upstream_bind_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
char *ngx_http_upstream_param_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
ngx_int_t ngx_http_upstream_hide_headers_hash(ngx_conf_t *cf,
    ngx_http_upstream_conf_t *conf, ngx_http_upstream_conf_t *prev,
    ngx_str_t *default_hide_headers, ngx_hash_init_t *hash);
```

```
403 #define ngx_http_conf_upstream_srv_conf(uscf, module)                         \
404     uscf->srv_conf[module.ctx_index]
405
406
407 extern ngx_module_t        ngx_http_upstream_module;
408 extern ngx_conf_bitmask_t  ngx_http_upstream_cache_method_mask[];
409 extern ngx_conf_bitmask_t  ngx_http_upstream_ignore_headers_masks[];
410
411
412 #endif /* _NGX_HTTP_UPSTREAM_H_INCLUDED_ */
```

One Level Up Top Level

src/event/ngx_event_pipe.c - nginx-1.7.10

Functions defined

- ngx_event_pipe
- ngx_event_pipe_add_free_buf
- ngx_event_pipe_copy_input_filter
- ngx_event_pipe_drain_chains
- ngx_event_pipe_read_upstream
- ngx_event_pipe_remove_shadow_links
- ngx_event_pipe_write_chain_to_temp_file
- ngx_event_pipe_write_to_downstream

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11  #include <ngx_event_pipe.h>
12
13
14  static ngx_int_t ngx_event_pipe_read_upstream(ngx_event_pipe_t *p);
15  static ngx_int_t ngx_event_pipe_write_to_downstream(ngx_event_pipe_t *p);
16
17  static ngx_int_t ngx_event_pipe_write_chain_to_temp_file(ngx_event_pipe_t *p);
18  static ngx_inline void ngx_event_pipe_remove_shadow_links(ngx_buf_t *buf);
19  static ngx_int_t ngx_event_pipe_drain_chains(ngx_event_pipe_t *p);
20
21
22  ngx_int_t
23  ngx_event_pipe(ngx_event_pipe_t *p, ngx_int_t do_write)
24  {
25      u_int         flags;
26      ngx_int_t     rc;
27      ngx_event_t  *rev, *wev;
28
29      for ( ;; ) {
30          if (do_write) {
31              p->log->action = "sending to client";
32
33              rc = ngx_event_pipe_write_to_downstream(p);
34
35              if (rc == NGX_ABORT) {
36                  return NGX_ABORT;
37              }
38
39              if (rc == NGX_BUSY) {
40                  return NGX_OK;
41              }
42          }
43
44          p->read = 0;
45          p->upstream_blocked = 0;
46
47          p->log->action = "reading upstream";
```

```
48
49          if (ngx_event_pipe_read_upstream(p) == NGX_ABORT) {
50              return NGX_ABORT;
51          }
52
53          if (!p->read && !p->upstream_blocked) {
54              break;
55          }
56
57          do_write = 1;
58      }
59
60      if (p->upstream->fd != (ngx_socket_t) -1) {
61          rev = p->upstream->read;
62
63          flags = (rev->eof || rev->error) ? NGX_CLOSE_EVENT : 0;
64
65          if (ngx_handle_read_event(rev, flags) != NGX_OK) {
66              return NGX_ABORT;
67          }
68
69          if (!rev->delayed) {
70              if (rev->active && !rev->ready) {
71                  ngx_add_timer(rev, p->read_timeout);
72
73              } else if (rev->timer_set) {
74                  ngx_del_timer(rev);
75              }
76          }
77      }
78
79      if (p->downstream->fd != (ngx_socket_t) -1
80          && p->downstream->data == p->output_ctx)
81      {
82          wev = p->downstream->write;
83          if (ngx_handle_write_event(wev, p->send_lowat) != NGX_OK) {
84              return NGX_ABORT;
85          }
86
87          if (!wev->delayed) {
88              if (wev->active && !wev->ready) {
89                  ngx_add_timer(wev, p->send_timeout);
90
91              } else if (wev->timer_set) {
92                  ngx_del_timer(wev);
93              }
94          }
95      }
96
97      return NGX_OK;
98  }
99
100
101 static ngx_int_t
102 ngx_event_pipe_read_upstream(ngx_event_pipe_t *p)
103 {
104     off_t         limit;
105     ssize_t       n, size;
106     ngx_int_t     rc;
107     ngx_buf_t    *b;
108     ngx_msec_t    delay;
109     ngx_chain_t  *chain, *cl, *ln;
110
111     if (p->upstream_eof || p->upstream_error || p->upstream_done) {
112         return NGX_OK;
113     }
114
115     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
116                    "pipe read upstream: %d", p->upstream->read->ready);
117
118     for ( ;; ) {
119
120         if (p->upstream_eof || p->upstream_error || p->upstream_done) {
121             break;
122         }
123
```

```
        if (p->preread_bufs == NULL && !p->upstream->read->ready) {
            break;
        }

        if (p->preread_bufs) {

            /* use the pre-read bufs if they exist */

            chain = p->preread_bufs;
            p->preread_bufs = NULL;
            n = p->preread_size;

            ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "pipe preread: %z", n);

            if (n) {
                p->read = 1;
            }

        } else {

#if (NGX_HAVE_KQUEUE)

            /*
             * kqueue notifies about the end of file or a pending error.
             * This test allows not to allocate a buf on these conditions
             * and not to call c->recv_chain().
             */

            if (p->upstream->read->available == 0
                && p->upstream->read->pending_eof)
            {
                p->upstream->read->ready = 0;
                p->upstream->read->eof = 1;
                p->upstream_eof = 1;
                p->read = 1;

                if (p->upstream->read->kq_errno) {
                    p->upstream->read->error = 1;
                    p->upstream_error = 1;
                    p->upstream_eof = 0;

                    ngx_log_error(NGX_LOG_ERR, p->log,
                                  p->upstream->read->kq_errno,
                                  "kevent() reported that upstream "
                                  "closed connection");
                }

                break;
            }
#endif

            if (p->limit_rate) {
                if (p->upstream->read->delayed) {
                    break;
                }

                limit = (off_t) p->limit_rate * (ngx_time() - p->start_sec + 1)
                        - p->read_length;

                if (limit <= 0) {
                    p->upstream->read->delayed = 1;
                    delay = (ngx_msec_t) (- limit * 1000 / p->limit_rate + 1);
                    ngx_add_timer(p->upstream->read, delay);
                    break;
                }

            } else {
                limit = 0;
            }

            if (p->free_raw_bufs) {

                /* use the free bufs if they exist */

                chain = p->free_raw_bufs;
```

```
            if (p->single_buf) {
                p->free_raw_bufs = p->free_raw_bufs->next;
                chain->next = NULL;
            } else {
                p->free_raw_bufs = NULL;
            }

        } else if (p->allocated < p->bufs.num) {

            /* allocate a new buf if it's still allowed */

            b = ngx_create_temp_buf(p->pool, p->bufs.size);
            if (b == NULL) {
                return NGX_ABORT;
            }

            p->allocated++;

            chain = ngx_alloc_chain_link(p->pool);
            if (chain == NULL) {
                return NGX_ABORT;
            }

            chain->buf = b;
            chain->next = NULL;

        } else if (!p->cacheable
                   && p->downstream->data == p->output_ctx
                   && p->downstream->write->ready
                   && !p->downstream->write->delayed)
        {
            /*
             * if the bufs are not needed to be saved in a cache and
             * a downstream is ready then write the bufs to a downstream
             */

            p->upstream_blocked = 1;

            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "pipe downstream ready");

            break;

        } else if (p->cacheable
                   || p->temp_file->offset < p->max_temp_file_size)
        {

            /*
             * if it is allowed, then save some bufs from p->in
             * to a temporary file, and add them to a p->out chain
             */

            rc = ngx_event_pipe_write_chain_to_temp_file(p);

            ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "pipe temp offset: %O", p->temp_file->offset);

            if (rc == NGX_BUSY) {
                break;
            }

            if (rc == NGX_AGAIN) {
                if (ngx_event_flags & NGX_USE_LEVEL_EVENT
                    && p->upstream->read->active
                    && p->upstream->read->ready)
                {
                    if (ngx_del_event(p->upstream->read, NGX_READ_EVENT, 0)
                        == NGX_ERROR)
                    {
                        return NGX_ABORT;
                    }
                }
            }

            if (rc != NGX_OK) {
                return rc;
```

```
            }

            chain = p->free_raw_bufs;
            if (p->single_buf) {
                p->free_raw_bufs = p->free_raw_bufs->next;
                chain->next = NULL;
            } else {
                p->free_raw_bufs = NULL;
            }

        } else {

            /* there are no bufs to read in */

            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "no pipe bufs to read in");

            break;
        }

        n = p->upstream->recv_chain(p->upstream, chain, limit);

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
                       "pipe recv chain: %z", n);

        if (p->free_raw_bufs) {
            chain->next = p->free_raw_bufs;
        }
        p->free_raw_bufs = chain;

        if (n == NGX_ERROR) {
            p->upstream_error = 1;
            return NGX_ERROR;
        }

        if (n == NGX_AGAIN) {
            if (p->single_buf) {
                ngx_event_pipe_remove_shadow_links(chain->buf);
            }

            break;
        }

        p->read = 1;

        if (n == 0) {
            p->upstream_eof = 1;
            break;
        }
    }

    delay = p->limit_rate ? (ngx_msec_t) n * 1000 / p->limit_rate : 0;

    p->read_length += n;
    cl = chain;
    p->free_raw_bufs = NULL;

    while (cl && n > 0) {

        ngx_event_pipe_remove_shadow_links(cl->buf);

        size = cl->buf->end - cl->buf->last;

        if (n >= size) {
            cl->buf->last = cl->buf->end;

            /* STUB */ cl->buf->num = p->num++;

            if (p->input_filter(p, cl->buf) == NGX_ERROR) {
                return NGX_ABORT;
            }

            n -= size;
            ln = cl;
            cl = cl->next;
            ngx_free_chain(p->pool, ln);
```

```
352
353             } else {
354                 cl->buf->last += n;
355                 n = 0;
356             }
357         }
358
359         if (cl) {
360             for (ln = cl; ln->next; ln = ln->next) { /* void */ }
361
362             ln->next = p->free_raw_bufs;
363             p->free_raw_bufs = cl;
364         }
365
366         if (delay > 0) {
367             p->upstream->read->delayed = 1;
368             ngx_add_timer(p->upstream->read, delay);
369             break;
370         }
371     }
372
373 #if (NGX_DEBUG)
374
375     for (cl = p->busy; cl; cl = cl->next) {
376         ngx_log_debug8(NGX_LOG_DEBUG_EVENT, p->log, 0,
377                        "pipe buf busy s:%d t:%d f:%d "
378                        "%p, pos %p, size: %z "
379                        "file: %O, size: %z",
380                        (cl->buf->shadow ? 1 : 0),
381                        cl->buf->temporary, cl->buf->in_file,
382                        cl->buf->start, cl->buf->pos,
383                        cl->buf->last - cl->buf->pos,
384                        cl->buf->file_pos,
385                        cl->buf->file_last - cl->buf->file_pos);
386     }
387
388     for (cl = p->out; cl; cl = cl->next) {
389         ngx_log_debug8(NGX_LOG_DEBUG_EVENT, p->log, 0,
390                        "pipe buf out  s:%d t:%d f:%d "
391                        "%p, pos %p, size: %z "
392                        "file: %O, size: %z",
393                        (cl->buf->shadow ? 1 : 0),
394                        cl->buf->temporary, cl->buf->in_file,
395                        cl->buf->start, cl->buf->pos,
396                        cl->buf->last - cl->buf->pos,
397                        cl->buf->file_pos,
398                        cl->buf->file_last - cl->buf->file_pos);
399     }
400
401     for (cl = p->in; cl; cl = cl->next) {
402         ngx_log_debug8(NGX_LOG_DEBUG_EVENT, p->log, 0,
403                        "pipe buf in   s:%d t:%d f:%d "
404                        "%p, pos %p, size: %z "
405                        "file: %O, size: %z",
406                        (cl->buf->shadow ? 1 : 0),
407                        cl->buf->temporary, cl->buf->in_file,
408                        cl->buf->start, cl->buf->pos,
409                        cl->buf->last - cl->buf->pos,
410                        cl->buf->file_pos,
411                        cl->buf->file_last - cl->buf->file_pos);
412     }
413
414     for (cl = p->free_raw_bufs; cl; cl = cl->next) {
415         ngx_log_debug8(NGX_LOG_DEBUG_EVENT, p->log, 0,
416                        "pipe buf free s:%d t:%d f:%d "
417                        "%p, pos %p, size: %z "
418                        "file: %O, size: %z",
419                        (cl->buf->shadow ? 1 : 0),
420                        cl->buf->temporary, cl->buf->in_file,
421                        cl->buf->start, cl->buf->pos,
422                        cl->buf->last - cl->buf->pos,
423                        cl->buf->file_pos,
424                        cl->buf->file_last - cl->buf->file_pos);
425     }
426
427     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
```

```
                   "pipe length: %O", p->length);

#endif

    if (p->free_raw_bufs && p->length != -1) {
        cl = p->free_raw_bufs;

        if (cl->buf->last - cl->buf->pos >= p->length) {

            p->free_raw_bufs = cl->next;

            /* STUB */ cl->buf->num = p->num++;

            if (p->input_filter(p, cl->buf) == NGX_ERROR) {
                return NGX_ABORT;
            }

            ngx_free_chain(p->pool, cl);
        }
    }

    if (p->length == 0) {
        p->upstream_done = 1;
        p->read = 1;
    }

    if ((p->upstream_eof || p->upstream_error) && p->free_raw_bufs) {

        /* STUB */ p->free_raw_bufs->buf->num = p->num++;

        if (p->input_filter(p, p->free_raw_bufs->buf) == NGX_ERROR) {
            return NGX_ABORT;
        }

        p->free_raw_bufs = p->free_raw_bufs->next;

        if (p->free_bufs && p->buf_to_file == NULL) {
            for (cl = p->free_raw_bufs; cl; cl = cl->next) {
                if (cl->buf->shadow == NULL) {
                    ngx_pfree(p->pool, cl->buf->start);
                }
            }
        }
    }

    if (p->cacheable && (p->in || p->buf_to_file)) {

        ngx_log_debug0(NGX_LOG_DEBUG_EVENT, p->log, 0,
                       "pipe write chain");

        if (ngx_event_pipe_write_chain_to_temp_file(p) == NGX_ABORT) {
            return NGX_ABORT;
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_event_pipe_write_to_downstream(ngx_event_pipe_t *p)
{
    u_char            *prev;
    size_t             bsize;
    ngx_int_t          rc;
    ngx_uint_t         flush, flushed, prev_last_shadow;
    ngx_chain_t       *out, **ll, *cl;
    ngx_connection_t  *downstream;

    downstream = p->downstream;

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
                   "pipe write downstream: %d", downstream->write->ready);

    flushed = 0;

```

```
    for ( ;; ) {
        if (p->downstream_error) {
            return ngx_event_pipe_drain_chains(p);
        }

        if (p->upstream_eof || p->upstream_error || p->upstream_done) {

            /* pass the p->out and p->in chains to the output filter */

            for (cl = p->busy; cl; cl = cl->next) {
                cl->buf->recycled = 0;
            }

            if (p->out) {
                ngx_log_debug0(NGX_LOG_DEBUG_EVENT, p->log, 0,
                               "pipe write downstream flush out");

                for (cl = p->out; cl; cl = cl->next) {
                    cl->buf->recycled = 0;
                }

                rc = p->output_filter(p->output_ctx, p->out);

                if (rc == NGX_ERROR) {
                    p->downstream_error = 1;
                    return ngx_event_pipe_drain_chains(p);
                }

                p->out = NULL;
            }

            if (p->in) {
                ngx_log_debug0(NGX_LOG_DEBUG_EVENT, p->log, 0,
                               "pipe write downstream flush in");

                for (cl = p->in; cl; cl = cl->next) {
                    cl->buf->recycled = 0;
                }

                rc = p->output_filter(p->output_ctx, p->in);

                if (rc == NGX_ERROR) {
                    p->downstream_error = 1;
                    return ngx_event_pipe_drain_chains(p);
                }

                p->in = NULL;
            }

            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "pipe write downstream done");

            /* TODO: free unused bufs */

            p->downstream_done = 1;
            break;
        }

        if (downstream->data != p->output_ctx
            || !downstream->write->ready
            || downstream->write->delayed)
        {
            break;
        }

        /* bsize is the size of the busy recycled bufs */

        prev = NULL;
        bsize = 0;

        for (cl = p->busy; cl; cl = cl->next) {

            if (cl->buf->recycled) {
                if (prev == cl->buf->start) {
                    continue;
                }
```

```
580
581                 bsize += cl->buf->end - cl->buf->start;
582                 prev = cl->buf->start;
583             }
584         }
585
586         ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
587                        "pipe write busy: %uz", bsize);
588
589         out = NULL;
590
591         if (bsize >= (size_t) p->busy_size) {
592             flush = 1;
593             goto flush;
594         }
595
596         flush = 0;
597         ll = NULL;
598         prev_last_shadow = 1;
599
600         for ( ;; ) {
601             if (p->out) {
602                 cl = p->out;
603
604                 if (cl->buf->recycled) {
605                     ngx_log_error(NGX_LOG_ALERT, p->log, 0,
606                                   "recycled buffer in pipe out chain");
607                 }
608
609                 p->out = p->out->next;
610
611             } else if (!p->cacheable && p->in) {
612                 cl = p->in;
613
614                 ngx_log_debug3(NGX_LOG_DEBUG_EVENT, p->log, 0,
615                                "pipe write buf ls:%d %p %z",
616                                cl->buf->last_shadow,
617                                cl->buf->pos,
618                                cl->buf->last - cl->buf->pos);
619
620                 if (cl->buf->recycled && prev_last_shadow) {
621                     if (bsize + cl->buf->end - cl->buf->start > p->busy_size) {
622                         flush = 1;
623                         break;
624                     }
625
626                     bsize += cl->buf->end - cl->buf->start;
627                 }
628
629                 prev_last_shadow = cl->buf->last_shadow;
630
631                 p->in = p->in->next;
632
633             } else {
634                 break;
635             }
636
637             cl->next = NULL;
638
639             if (out) {
640                 *ll = cl;
641             } else {
642                 out = cl;
643             }
644             ll = &cl->next;
645         }
646
647     flush:
648
649         ngx_log_debug2(NGX_LOG_DEBUG_EVENT, p->log, 0,
650                        "pipe write: out:%p, f:%d", out, flush);
651
652         if (out == NULL) {
653
654             if (!flush) {
655                 break;
```

```
656             }
657
658             /* a workaround for AIO */
659             if (flushed++ > 10) {
660                 return NGX_BUSY;
661             }
662         }
663
664         rc = p->output_filter(p->output_ctx, out);
665
666         ngx_chain_update_chains(p->pool, &p->free, &p->busy, &out, p->tag);
667
668         if (rc == NGX_ERROR) {
669             p->downstream_error = 1;
670             return ngx_event_pipe_drain_chains(p);
671         }
672
673         for (cl = p->free; cl; cl = cl->next) {
674
675             if (cl->buf->temp_file) {
676                 if (p->cacheable || !p->cyclic_temp_file) {
677                     continue;
678                 }
679
680                 /* reset p->temp_offset if all bufs had been sent */
681
682                 if (cl->buf->file_last == p->temp_file->offset) {
683                     p->temp_file->offset = 0;
684                 }
685             }
686
687             /* TODO: free buf if p->free_bufs && upstream done */
688
689             /* add the free shadow raw buf to p->free_raw_bufs */
690
691             if (cl->buf->last_shadow) {
692                 if (ngx_event_pipe_add_free_buf(p, cl->buf->shadow) != NGX_OK) {
693                     return NGX_ABORT;
694                 }
695
696                 cl->buf->last_shadow = 0;
697             }
698
699             cl->buf->shadow = NULL;
700         }
701     }
702
703     return NGX_OK;
704 }
705
706
707 static ngx_int_t
708 ngx_event_pipe_write_chain_to_temp_file(ngx_event_pipe_t *p)
709 {
710     ssize_t       size, bsize, n;
711     ngx_buf_t    *b;
712     ngx_uint_t    prev_last_shadow;
713     ngx_chain_t  *cl, *tl, *next, *out, **ll, **last_out, **last_free, fl;
714
715     if (p->buf_to_file) {
716         fl.buf = p->buf_to_file;
717         fl.next = p->in;
718         out = &fl;
719
720     } else {
721         out = p->in;
722     }
723
724     if (!p->cacheable) {
725
726         size = 0;
727         cl = out;
728         ll = NULL;
729         prev_last_shadow = 1;
730
731         ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0,
```

```
                      "pipe offset: %O", p->temp_file->offset);

        do {
            bsize = cl->buf->last - cl->buf->pos;

            ngx_log_debug4(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "pipe buf ls:%d %p, pos %p, size: %z",
                           cl->buf->last_shadow, cl->buf->start,
                           cl->buf->pos, bsize);

            if (prev_last_shadow
                && ((size + bsize > p->temp_file_write_size)
                    || (p->temp_file->offset + size + bsize
                        > p->max_temp_file_size)))
            {
                break;
            }

            prev_last_shadow = cl->buf->last_shadow;

            size += bsize;
            ll = &cl->next;
            cl = cl->next;

        } while (cl);

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0, "size: %z", size);

        if (ll == NULL) {
            return NGX_BUSY;
        }

        if (cl) {
           p->in = cl;
           *ll = NULL;

        } else {
           p->in = NULL;
           p->last_in = &p->in;
        }

    } else {
        p->in = NULL;
        p->last_in = &p->in;
    }

    n = ngx_write_chain_to_temp_file(p->temp_file, out);

    if (n == NGX_ERROR) {
        return NGX_ABORT;
    }

    if (p->buf_to_file) {
        p->temp_file->offset = p->buf_to_file->last - p->buf_to_file->pos;
        n -= p->buf_to_file->last - p->buf_to_file->pos;
        p->buf_to_file = NULL;
        out = out->next;
    }

    if (n > 0) {
        /* update previous buffer or add new buffer */

        if (p->out) {
            for (cl = p->out; cl->next; cl = cl->next) { /* void */ }

            b = cl->buf;

            if (b->file_last == p->temp_file->offset) {
                p->temp_file->offset += n;
                b->file_last = p->temp_file->offset;
                goto free;
            }

            last_out = &cl->next;

        } else {
```

```
            last_out = &p->out;
        }

        cl = ngx_chain_get_free_buf(p->pool, &p->free);
        if (cl == NULL) {
            return NGX_ABORT;
        }

        b = cl->buf;

        ngx_memzero(b, sizeof(ngx_buf_t));

        b->tag = p->tag;

        b->file = &p->temp_file->file;
        b->file_pos = p->temp_file->offset;
        p->temp_file->offset += n;
        b->file_last = p->temp_file->offset;

        b->in_file = 1;
        b->temp_file = 1;

        *last_out = cl;
    }

free:

    for (last_free = &p->free_raw_bufs;
         *last_free != NULL;
         last_free = &(*last_free)->next)
    {
        /* void */
    }

    for (cl = out; cl; cl = next) {
        next = cl->next;

        cl->next = p->free;
        p->free = cl;

        b = cl->buf;

        if (b->last_shadow) {

            tl = ngx_alloc_chain_link(p->pool);
            if (tl == NULL) {
                return NGX_ABORT;
            }

            tl->buf = b->shadow;
            tl->next = NULL;

            *last_free = tl;
            last_free = &tl->next;

            b->shadow->pos = b->shadow->start;
            b->shadow->last = b->shadow->start;

            ngx_event_pipe_remove_shadow_links(b->shadow);
        }
    }

    return NGX_OK;
}


/* the copy input filter */

ngx_int_t
ngx_event_pipe_copy_input_filter(ngx_event_pipe_t *p, ngx_buf_t *buf)
{
    ngx_buf_t    *b;
    ngx_chain_t  *cl;

    if (buf->pos == buf->last) {
        return NGX_OK;
```

```
    }

    cl = ngx_chain_get_free_buf(p->pool, &p->free);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    b = cl->buf;

    ngx_memcpy(b, buf, sizeof(ngx_buf_t));
    b->shadow = buf;
    b->tag = p->tag;
    b->last_shadow = 1;
    b->recycled = 1;
    buf->shadow = b;

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0, "input buf #%d", b->num);

    if (p->in) {
        *p->last_in = cl;
    } else {
        p->in = cl;
    }
    p->last_in = &cl->next;

    if (p->length == -1) {
        return NGX_OK;
    }

    p->length -= b->last - b->pos;

    return NGX_OK;
}


static ngx_inline void
ngx_event_pipe_remove_shadow_links(ngx_buf_t *buf)
{
    ngx_buf_t  *b, *next;

    b = buf->shadow;

    if (b == NULL) {
        return;
    }

    while (!b->last_shadow) {
        next = b->shadow;

        b->temporary = 0;
        b->recycled = 0;

        b->shadow = NULL;
        b = next;
    }

    b->temporary = 0;
    b->recycled = 0;
    b->last_shadow = 0;

    b->shadow = NULL;

    buf->shadow = NULL;
}


ngx_int_t
ngx_event_pipe_add_free_buf(ngx_event_pipe_t *p, ngx_buf_t *b)
{
    ngx_chain_t  *cl;

    cl = ngx_alloc_chain_link(p->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

```

```
    if (p->buf_to_file && b->start == p->buf_to_file->start) {
        b->pos = p->buf_to_file->last;
        b->last = p->buf_to_file->last;

    } else {
        b->pos = b->start;
        b->last = b->start;
    }

    b->shadow = NULL;

    cl->buf = b;

    if (p->free_raw_bufs == NULL) {
        p->free_raw_bufs = cl;
        cl->next = NULL;

        return NGX_OK;
    }

    if (p->free_raw_bufs->buf->pos == p->free_raw_bufs->buf->last) {

        /* add the free buf to the list start */

        cl->next = p->free_raw_bufs;
        p->free_raw_bufs = cl;

        return NGX_OK;
    }

    /* the first free buf is partially filled, thus add the free buf after it */

    cl->next = p->free_raw_bufs->next;
    p->free_raw_bufs->next = cl;

    return NGX_OK;
}


static ngx_int_t
 ngx_event_pipe_drain_chains(ngx_event_pipe_t *p)
 {
     ngx_chain_t  *cl, *tl;

     for ( ;; ) {
         if (p->busy) {
             cl = p->busy;
             p->busy = NULL;

         } else if (p->out) {
             cl = p->out;
             p->out = NULL;

         } else if (p->in) {
             cl = p->in;
             p->in = NULL;

         } else {
             return NGX_OK;
         }

         while (cl) {
             if (cl->buf->last_shadow) {
                 if (ngx_event_pipe_add_free_buf(p, cl->buf->shadow) != NGX_OK) {
                     return NGX_ABORT;
                 }

                 cl->buf->last_shadow = 0;
             }

             cl->buf->shadow = NULL;
             tl = cl->next;
             cl->next = p->free;
             p->free = cl;
             cl = tl;
         }
```

```
1036      }
1037  }
```

One Level Up Top Level

src/event/ngx_event_pipe.h - nginx-1.7.10

Data types defined

- ngx_event_pipe_input_filter_pt
- ngx_event_pipe_output_filter_pt
- ngx_event_pipe_s
- ngx_event_pipe_t

Macros defined

- _NGX_EVENT_PIPE_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_EVENT_PIPE_H_INCLUDED_
9  #define _NGX_EVENT_PIPE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15
16
17  typedef struct ngx_event_pipe_s  ngx_event_pipe_t;
18
19  typedef ngx_int_t (*ngx_event_pipe_input_filter_pt)(ngx_event_pipe_t *p,
20                                                      ngx_buf_t *buf);
21  typedef ngx_int_t (*ngx_event_pipe_output_filter_pt)(void *data,
22                                                       ngx_chain_t *chain);
23
24
25  struct ngx_event_pipe_s {
26      ngx_connection_t  *upstream;
27      ngx_connection_t  *downstream;
28
29      ngx_chain_t       *free_raw_bufs;
30      ngx_chain_t       *in;
31      ngx_chain_t      **last_in;
32
33      ngx_chain_t       *out;
34      ngx_chain_t       *free;
35      ngx_chain_t       *busy;
36
37      /*
38       * the input filter i.e. that moves HTTP/1.1 chunks
39       * from the raw bufs to an incoming chain
40       */
41
42      ngx_event_pipe_input_filter_pt    input_filter;
43      void                             *input_ctx;
44
45      ngx_event_pipe_output_filter_pt   output_filter;
46      void                             *output_ctx;
47
48      unsigned           read:1;
49      unsigned           cacheable:1;
50      unsigned           single_buf:1;
```

```
    unsigned           free_bufs:1;
    unsigned           upstream_done:1;
    unsigned           upstream_error:1;
    unsigned           upstream_eof:1;
    unsigned           upstream_blocked:1;
    unsigned           downstream_done:1;
    unsigned           downstream_error:1;
    unsigned           cyclic_temp_file:1;

    ngx_int_t          allocated;
    ngx_bufs_t         bufs;
    ngx_buf_tag_t      tag;

    ssize_t            busy_size;

    off_t              read_length;
    off_t              length;

    off_t              max_temp_file_size;
    ssize_t            temp_file_write_size;

    ngx_msec_t         read_timeout;
    ngx_msec_t         send_timeout;
    ssize_t            send_lowat;

    ngx_pool_t        *pool;
    ngx_log_t         *log;

    ngx_chain_t       *preread_bufs;
    size_t             preread_size;
    ngx_buf_t         *buf_to_file;

    size_t             limit_rate;
    time_t             start_sec;

    ngx_temp_file_t   *temp_file;

    /* STUB */ int     num;
};


ngx_int_t ngx_event_pipe(ngx_event_pipe_t *p, ngx_int_t do_write);
ngx_int_t ngx_event_pipe_copy_input_filter(ngx_event_pipe_t *p, ngx_buf_t *buf);
ngx_int_t ngx_event_pipe_add_free_buf(ngx_event_pipe_t *p, ngx_buf_t *b);


#endif /* _NGX_EVENT_PIPE_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_upstream_round_robin.c - nginx-1.7.10

Functions defined

- ngx_http_upstream_create_round_robin_peer
- ngx_http_upstream_empty_save_session
- ngx_http_upstream_empty_set_session
- ngx_http_upstream_free_round_robin_peer
- ngx_http_upstream_get_peer
- ngx_http_upstream_get_round_robin_peer
- ngx_http_upstream_init_round_robin
- ngx_http_upstream_init_round_robin_peer
- ngx_http_upstream_save_round_robin_peer_session
- ngx_http_upstream_set_round_robin_peer_session

Macros defined

- ngx_http_upstream_tries

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define ngx_http_upstream_tries(p) ((p)->number                                  \
14                                      + ((p)->next ? (p)->next->number : 0))
15
16
17  static ngx_http_upstream_rr_peer_t *ngx_http_upstream_get_peer(
18      ngx_http_upstream_rr_peer_data_t *rrp);
19
20  #if (NGX_HTTP_SSL)
21
22  static ngx_int_t ngx_http_upstream_empty_set_session(ngx_peer_connection_t *pc,
23      void *data);
24  static void ngx_http_upstream_empty_save_session(ngx_peer_connection_t *pc,
25      void *data);
26
27  #endif
28
29
30  ngx_int_t
31  ngx_http_upstream_init_round_robin(ngx_conf_t *cf,
32      ngx_http_upstream_srv_conf_t *us)
33  {
34      ngx_url_t                      u;
35      ngx_uint_t                     i, j, n, w;
36      ngx_http_upstream_server_t    *server;
```

```
    ngx_http_upstream_rr_peer_t   *peer;
    ngx_http_upstream_rr_peers_t  *peers, *backup;

    us->peer.init = ngx_http_upstream_init_round_robin_peer;

    if (us->servers) {
        server = us->servers->elts;

        n = 0;
        w = 0;

        for (i = 0; i < us->servers->nelts; i++) {
            if (server[i].backup) {
                continue;
            }

            n += server[i].naddrs;
            w += server[i].naddrs * server[i].weight;
        }

        if (n == 0) {
            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                          "no servers in upstream \"%V\" in %s:%ui",
                          &us->host, us->file_name, us->line);
            return NGX_ERROR;
        }

        peers = ngx_pcalloc(cf->pool, sizeof(ngx_http_upstream_rr_peers_t)
                            + sizeof(ngx_http_upstream_rr_peer_t) * (n - 1));
        if (peers == NULL) {
            return NGX_ERROR;
        }

        peers->single = (n == 1);
        peers->number = n;
        peers->weighted = (w != n);
        peers->total_weight = w;
        peers->name = &us->host;

        n = 0;
        peer = peers->peer;

        for (i = 0; i < us->servers->nelts; i++) {
            if (server[i].backup) {
                continue;
            }

            for (j = 0; j < server[i].naddrs; j++) {
                peer[n].sockaddr = server[i].addrs[j].sockaddr;
                peer[n].socklen = server[i].addrs[j].socklen;
                peer[n].name = server[i].addrs[j].name;
                peer[n].weight = server[i].weight;
                peer[n].effective_weight = server[i].weight;
                peer[n].current_weight = 0;
                peer[n].max_fails = server[i].max_fails;
                peer[n].fail_timeout = server[i].fail_timeout;
                peer[n].down = server[i].down;
                peer[n].server = server[i].name;
                n++;
            }
        }

        us->peer.data = peers;

         /* backup servers */

         n = 0;
         w = 0;

         for (i = 0; i < us->servers->nelts; i++) {
             if (!server[i].backup) {
                 continue;
             }

             n += server[i].naddrs;
             w += server[i].naddrs * server[i].weight;
```

```
        }

        if (n == 0) {
            return NGX_OK;
        }

        backup = ngx_pcalloc(cf->pool, sizeof(ngx_http_upstream_rr_peers_t)
                             + sizeof(ngx_http_upstream_rr_peer_t) * (n - 1));
        if (backup == NULL) {
            return NGX_ERROR;
        }

        peers->single = 0;
        backup->single = 0;
        backup->number = n;
        backup->weighted = (w != n);
        backup->total_weight = w;
        backup->name = &us->host;

        n = 0;
        peer = backup->peer;

        for (i = 0; i < us->servers->nelts; i++) {
            if (!server[i].backup) {
                continue;
            }

            for (j = 0; j < server[i].naddrs; j++) {
                peer[n].sockaddr = server[i].addrs[j].sockaddr;
                peer[n].socklen = server[i].addrs[j].socklen;
                peer[n].name = server[i].addrs[j].name;
                peer[n].weight = server[i].weight;
                peer[n].effective_weight = server[i].weight;
                peer[n].current_weight = 0;
                peer[n].max_fails = server[i].max_fails;
                peer[n].fail_timeout = server[i].fail_timeout;
                peer[n].down = server[i].down;
                peer[n].server = server[i].name;
                n++;
            }
        }

        peers->next = backup;

        return NGX_OK;
    }


    /* an upstream implicitly defined by proxy_pass, etc. */

    if (us->port == 0) {
        ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                      "no port in upstream \"%V\" in %s:%ui",
                      &us->host, us->file_name, us->line);
        return NGX_ERROR;
    }

    ngx_memzero(&u, sizeof(ngx_url_t));

    u.host = us->host;
    u.port = us->port;

    if (ngx_inet_resolve_host(cf->pool, &u) != NGX_OK) {
        if (u.err) {
            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                          "%s in upstream \"%V\" in %s:%ui",
                          u.err, &us->host, us->file_name, us->line);
        }

        return NGX_ERROR;
    }

    n = u.naddrs;

    peers = ngx_pcalloc(cf->pool, sizeof(ngx_http_upstream_rr_peers_t)
                                + sizeof(ngx_http_upstream_rr_peer_t) * (n - 1));
```

```
    if (peers == NULL) {
        return NGX_ERROR;
    }

    peers->single = (n == 1);
    peers->number = n;
    peers->weighted = 0;
    peers->total_weight = n;
    peers->name = &us->host;

    peer = peers->peer;

    for (i = 0; i < u.naddrs; i++) {
        peer[i].sockaddr = u.addrs[i].sockaddr;
        peer[i].socklen = u.addrs[i].socklen;
        peer[i].name = u.addrs[i].name;
        peer[i].weight = 1;
        peer[i].effective_weight = 1;
        peer[i].current_weight = 0;
        peer[i].max_fails = 1;
        peer[i].fail_timeout = 10;
    }

    us->peer.data = peers;

    /* implicitly defined upstream has no backup servers */

    return NGX_OK;
}


ngx_int_t
ngx_http_upstream_init_round_robin_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us)
{
    ngx_uint_t                         n;
    ngx_http_upstream_rr_peer_data_t  *rrp;

    rrp = r->upstream->peer.data;

    if (rrp == NULL) {
        rrp = ngx_palloc(r->pool, sizeof(ngx_http_upstream_rr_peer_data_t));
        if (rrp == NULL) {
            return NGX_ERROR;
        }

        r->upstream->peer.data = rrp;
    }

    rrp->peers = us->peer.data;
    rrp->current = 0;

    n = rrp->peers->number;

    if (rrp->peers->next && rrp->peers->next->number > n) {
        n = rrp->peers->next->number;
    }

    if (n <= 8 * sizeof(uintptr_t)) {
        rrp->tried = &rrp->data;
        rrp->data = 0;

    } else {
        n = (n + (8 * sizeof(uintptr_t) - 1)) / (8 * sizeof(uintptr_t));

        rrp->tried = ngx_pcalloc(r->pool, n * sizeof(uintptr_t));
        if (rrp->tried == NULL) {
            return NGX_ERROR;
        }
    }

    r->upstream->peer.get = ngx_http_upstream_get_round_robin_peer;
    r->upstream->peer.free = ngx_http_upstream_free_round_robin_peer;
    r->upstream->peer.tries = ngx_http_upstream_tries(rrp->peers);
#if (NGX_HTTP_SSL)
    r->upstream->peer.set_session =
```

```
                                ngx_http_upstream_set_round_robin_peer_session;
    r->upstream->peer.save_session =
                                ngx_http_upstream_save_round_robin_peer_session;
#endif

    return NGX_OK;
}


ngx_int_t
ngx_http_upstream_create_round_robin_peer(ngx_http_request_t *r,
    ngx_http_upstream_resolved_t *ur)
{
    u_char                            *p;
    size_t                             len;
    socklen_t                          socklen;
    ngx_uint_t                         i, n;
    struct sockaddr                   *sockaddr;
    ngx_http_upstream_rr_peer_t       *peer;
    ngx_http_upstream_rr_peers_t      *peers;
    ngx_http_upstream_rr_peer_data_t  *rrp;

    rrp = r->upstream->peer.data;

    if (rrp == NULL) {
        rrp = ngx_palloc(r->pool, sizeof(ngx_http_upstream_rr_peer_data_t));
        if (rrp == NULL) {
            return NGX_ERROR;
        }

        r->upstream->peer.data = rrp;
    }

    peers = ngx_pcalloc(r->pool, sizeof(ngx_http_upstream_rr_peers_t)
                    + sizeof(ngx_http_upstream_rr_peer_t) * (ur->naddrs - 1));
    if (peers == NULL) {
        return NGX_ERROR;
    }

    peers->single = (ur->naddrs == 1);
    peers->number = ur->naddrs;
    peers->name = &ur->host;

    peer = peers->peer;

    if (ur->sockaddr) {
        peer[0].sockaddr = ur->sockaddr;
        peer[0].socklen = ur->socklen;
        peer[0].name = ur->host;
        peer[0].weight = 1;
        peer[0].effective_weight = 1;
        peer[0].current_weight = 0;
        peer[0].max_fails = 1;
        peer[0].fail_timeout = 10;

    } else {

        for (i = 0; i < ur->naddrs; i++) {

            socklen = ur->addrs[i].socklen;

            sockaddr = ngx_palloc(r->pool, socklen);
            if (sockaddr == NULL) {
                return NGX_ERROR;
            }

            ngx_memcpy(sockaddr, ur->addrs[i].sockaddr, socklen);

            switch (sockaddr->sa_family) {
#if (NGX_HAVE_INET6)
            case AF_INET6:
                ((struct sockaddr_in6 *) sockaddr)->sin6_port = htons(ur->port);
                break;
#endif
            default: /* AF_INET */
                ((struct sockaddr_in *) sockaddr)->sin_port = htons(ur->port);
```

```
            }

            p = ngx_pnalloc(r->pool, NGX_SOCKADDR_STRLEN);
            if (p == NULL) {
                return NGX_ERROR;
            }

            len = ngx_sock_ntop(sockaddr, socklen, p, NGX_SOCKADDR_STRLEN, 1);

            peer[i].sockaddr = sockaddr;
            peer[i].socklen = socklen;
            peer[i].name.len = len;
            peer[i].name.data = p;
            peer[i].weight = 1;
            peer[i].effective_weight = 1;
            peer[i].current_weight = 0;
            peer[i].max_fails = 1;
            peer[i].fail_timeout = 10;
        }
    }

    rrp->peers = peers;
    rrp->current = 0;

    if (rrp->peers->number <= 8 * sizeof(uintptr_t)) {
        rrp->tried = &rrp->data;
        rrp->data = 0;

    } else {
        n = (rrp->peers->number + (8 * sizeof(uintptr_t) - 1))
                / (8 * sizeof(uintptr_t));

        rrp->tried = ngx_pcalloc(r->pool, n * sizeof(uintptr_t));
        if (rrp->tried == NULL) {
            return NGX_ERROR;
        }
    }

    r->upstream->peer.get = ngx_http_upstream_get_round_robin_peer;
    r->upstream->peer.free = ngx_http_upstream_free_round_robin_peer;
    r->upstream->peer.tries = ngx_http_upstream_tries(rrp->peers);
#if (NGX_HTTP_SSL)
    r->upstream->peer.set_session = ngx_http_upstream_empty_set_session;
    r->upstream->peer.save_session = ngx_http_upstream_empty_save_session;
#endif

    return NGX_OK;
}


ngx_int_t
ngx_http_upstream_get_round_robin_peer(ngx_peer_connection_t *pc, void *data)
{
    ngx_http_upstream_rr_peer_data_t  *rrp = data;

    ngx_int_t                      rc;
    ngx_uint_t                     i, n;
    ngx_http_upstream_rr_peer_t   *peer;
    ngx_http_upstream_rr_peers_t  *peers;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "get rr peer, try: %ui", pc->tries);

    pc->cached = 0;
    pc->connection = NULL;

    peers = rrp->peers;

    /* ngx_lock_mutex(peers->mutex); */

    if (peers->single) {
        peer = &peers->peer[0];

        if (peer->down) {
            goto failed;
        }
```

```
417
418     } else {
419
420         /* there are several peers */
421
422         peer = ngx_http_upstream_get_peer(rrp);
423
424         if (peer == NULL) {
425             goto failed;
426         }
427
428         ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
429                        "get rr peer, current: %ui %i",
430                        rrp->current, peer->current_weight);
431     }
432
433     pc->sockaddr = peer->sockaddr;
434     pc->socklen = peer->socklen;
435     pc->name = &peer->name;
436
437     /* ngx_unlock_mutex(peers->mutex); */
438
439     return NGX_OK;
440
441 failed:
442
443     if (peers->next) {
444
445         /* ngx_unlock_mutex(peers->mutex); */
446
447         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pc->log, 0, "backup servers");
448
449         rrp->peers = peers->next;
450
451         n = (rrp->peers->number + (8 * sizeof(uintptr_t) - 1))
452                 / (8 * sizeof(uintptr_t));
453
454         for (i = 0; i < n; i++) {
455              rrp->tried[i] = 0;
456         }
457
458         rc = ngx_http_upstream_get_round_robin_peer(pc, rrp);
459
460         if (rc != NGX_BUSY) {
461             return rc;
462         }
463
464         /* ngx_lock_mutex(peers->mutex); */
465     }
466
467     /* all peers failed, mark them as live for quick recovery */
468
469     for (i = 0; i < peers->number; i++) {
470         peers->peer[i].fails = 0;
471     }
472
473     /* ngx_unlock_mutex(peers->mutex); */
474
475     pc->name = peers->name;
476
477     return NGX_BUSY;
478 }
479
480
481 static ngx_http_upstream_rr_peer_t *
482 ngx_http_upstream_get_peer(ngx_http_upstream_rr_peer_data_t *rrp)
483 {
484     time_t                        now;
485     uintptr_t                     m;
486     ngx_int_t                     total;
487     ngx_uint_t                    i, n;
488     ngx_http_upstream_rr_peer_t  *peer, *best;
489
490     now = ngx_time();
491
492     best = NULL;
```

```
493     total = 0;
494
495     for (i = 0; i < rrp->peers->number; i++) {
496
497         n = i / (8 * sizeof(uintptr_t));
498         m = (uintptr_t) 1 << i % (8 * sizeof(uintptr_t));
499
500         if (rrp->tried[n] & m) {
501             continue;
502         }
503
504         peer = &rrp->peers->peer[i];
505
506         if (peer->down) {
507             continue;
508         }
509
510         if (peer->max_fails
511             && peer->fails >= peer->max_fails
512             && now - peer->checked <= peer->fail_timeout)
513         {
514             continue;
515         }
516
517         peer->current_weight += peer->effective_weight;
518         total += peer->effective_weight;
519
520         if (peer->effective_weight < peer->weight) {
521             peer->effective_weight++;
522         }
523
524         if (best == NULL || peer->current_weight > best->current_weight) {
525             best = peer;
526         }
527     }
528
529     if (best == NULL) {
530         return NULL;
531     }
532
533     i = best - &rrp->peers->peer[0];
534
535     rrp->current = i;
536
537     n = i / (8 * sizeof(uintptr_t));
538     m = (uintptr_t) 1 << i % (8 * sizeof(uintptr_t));
539
540     rrp->tried[n] |= m;
541
542     best->current_weight -= total;
543
544     if (now - best->checked > best->fail_timeout) {
545         best->checked = now;
546     }
547
548     return best;
549 }
550
551
552 void
553 ngx_http_upstream_free_round_robin_peer(ngx_peer_connection_t *pc, void *data,
554     ngx_uint_t state)
555 {
556     ngx_http_upstream_rr_peer_data_t  *rrp = data;
557
558     time_t                       now;
559     ngx_http_upstream_rr_peer_t  *peer;
560
561     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
562                    "free rr peer %ui %ui", pc->tries, state);
563
564     /* TODO: NGX_PEER_KEEPALIVE */
565
566     if (rrp->peers->single) {
567         pc->tries = 0;
568         return;
```

```
    }

    peer = &rrp->peers->peer[rrp->current];

    if (state & NGX_PEER_FAILED) {
        now = ngx_time();

        /* ngx_lock_mutex(rrp->peers->mutex); */

        peer->fails++;
        peer->accessed = now;
        peer->checked = now;

        if (peer->max_fails) {
            peer->effective_weight -= peer->weight / peer->max_fails;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                       "free rr peer failed: %ui %i",
                       rrp->current, peer->effective_weight);

        if (peer->effective_weight < 0) {
            peer->effective_weight = 0;
        }

        /* ngx_unlock_mutex(rrp->peers->mutex); */

    } else {

        /* mark peer live if check passed */

        if (peer->accessed < peer->checked) {
            peer->fails = 0;
        }
    }

    if (pc->tries) {
        pc->tries--;
    }

    /* ngx_unlock_mutex(rrp->peers->mutex); */
}


#if (NGX_HTTP_SSL)

ngx_int_t
ngx_http_upstream_set_round_robin_peer_session(ngx_peer_connection_t *pc,
    void *data)
{
    ngx_http_upstream_rr_peer_data_t  *rrp = data;

    ngx_int_t                      rc;
    ngx_ssl_session_t             *ssl_session;
    ngx_http_upstream_rr_peer_t   *peer;

    peer = &rrp->peers->peer[rrp->current];

    /* TODO: threads only mutex */
    /* ngx_lock_mutex(rrp->peers->mutex); */

    ssl_session = peer->ssl_session;

    rc = ngx_ssl_set_session(pc->connection, ssl_session);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "set session: %p", ssl_session);

    /* ngx_unlock_mutex(rrp->peers->mutex); */

    return rc;
}


void
ngx_http_upstream_save_round_robin_peer_session(ngx_peer_connection_t *pc,
```

```
    void *data)
{
    ngx_http_upstream_rr_peer_data_t  *rrp = data;

    ngx_ssl_session_t            *old_ssl_session, *ssl_session;
    ngx_http_upstream_rr_peer_t  *peer;

    ssl_session = ngx_ssl_get_session(pc->connection);

    if (ssl_session == NULL) {
        return;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "save session: %p", ssl_session);

    peer = &rrp->peers->peer[rrp->current];

    /* TODO: threads only mutex */
    /* ngx_lock_mutex(rrp->peers->mutex); */

    old_ssl_session = peer->ssl_session;
    peer->ssl_session = ssl_session;

    /* ngx_unlock_mutex(rrp->peers->mutex); */

    if (old_ssl_session) {

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                       "old session: %p", old_ssl_session);

        /* TODO: may block */

        ngx_ssl_free_session(old_ssl_session);
    }
}


static ngx_int_t
ngx_http_upstream_empty_set_session(ngx_peer_connection_t *pc, void *data)
{
    return NGX_OK;
}


static void
ngx_http_upstream_empty_save_session(ngx_peer_connection_t *pc, void *data)
{
    return;
}

#endif
```

One Level Up Top Level

src/http/ngx_http_upstream_round_robin.h - nginx-1.7.10

Data types defined

- ngx_http_upstream_rr_peer_data_t
- ngx_http_upstream_rr_peer_t
- ngx_http_upstream_rr_peers_s
- ngx_http_upstream_rr_peers_t

Macros defined

- _NGX_HTTP_UPSTREAM_ROUND_ROBIN_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_UPSTREAM_ROUND_ROBIN_H_INCLUDED_
9  #define _NGX_HTTP_UPSTREAM_ROUND_ROBIN_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
17  typedef struct {
18      struct sockaddr                *sockaddr;
19      socklen_t                       socklen;
20      ngx_str_t                       name;
21      ngx_str_t                       server;
22
23      ngx_int_t                       current_weight;
24      ngx_int_t                       effective_weight;
25      ngx_int_t                       weight;
26
27      ngx_uint_t                      fails;
28      time_t                          accessed;
29      time_t                          checked;
30
31      ngx_uint_t                      max_fails;
32      time_t                          fail_timeout;
33
34      ngx_uint_t                      down;          /* unsigned  down:1; */
35
36  #if (NGX_HTTP_SSL)
37      ngx_ssl_session_t              *ssl_session;   /* local to a process */
38  #endif
39  } ngx_http_upstream_rr_peer_t;
40
41
42  typedef struct ngx_http_upstream_rr_peers_s  ngx_http_upstream_rr_peers_t;
43
44  struct ngx_http_upstream_rr_peers_s {
45      ngx_uint_t                      number;
46
47  /* ngx_mutex_t                    *mutex; */
48
49      ngx_uint_t                      total_weight;
50
```

```
    unsigned                         single:1;
    unsigned                         weighted:1;

    ngx_str_t                       *name;

    ngx_http_upstream_rr_peers_t    *next;

    ngx_http_upstream_rr_peer_t      peer[1];
};


typedef struct {
    ngx_http_upstream_rr_peers_t    *peers;
    ngx_uint_t                       current;
    uintptr_t                       *tried;
    uintptr_t                        data;
} ngx_http_upstream_rr_peer_data_t;


ngx_int_t ngx_http_upstream_init_round_robin(ngx_conf_t *cf,
    ngx_http_upstream_srv_conf_t *us);
ngx_int_t ngx_http_upstream_init_round_robin_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us);
ngx_int_t ngx_http_upstream_create_round_robin_peer(ngx_http_request_t *r,
    ngx_http_upstream_resolved_t *ur);
ngx_int_t ngx_http_upstream_get_round_robin_peer(ngx_peer_connection_t *pc,
    void *data);
void ngx_http_upstream_free_round_robin_peer(ngx_peer_connection_t *pc,
    void *data, ngx_uint_t state);

#if (NGX_HTTP_SSL)
ngx_int_t
    ngx_http_upstream_set_round_robin_peer_session(ngx_peer_connection_t *pc,
    void *data);
void ngx_http_upstream_save_round_robin_peer_session(ngx_peer_connection_t *pc,
    void *data);
#endif


#endif /* _NGX_HTTP_UPSTREAM_ROUND_ROBIN_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_file_cache.c - nginx-1.7.10

Global variables defined

- ngx_http_cache_status
- ngx_http_file_cache_key

Functions defined

- ngx_http_cache_aio_event_handler
- ngx_http_cache_send
- ngx_http_file_cache_add
- ngx_http_file_cache_add_file
- ngx_http_file_cache_aio_read
- ngx_http_file_cache_cleanup
- ngx_http_file_cache_create
- ngx_http_file_cache_create_key
- ngx_http_file_cache_delete
- ngx_http_file_cache_delete_file
- ngx_http_file_cache_exists
- ngx_http_file_cache_expire
- ngx_http_file_cache_forced_expire
- ngx_http_file_cache_free
- ngx_http_file_cache_init
- ngx_http_file_cache_loader
- ngx_http_file_cache_loader_sleep
- ngx_http_file_cache_lock
- ngx_http_file_cache_lock_wait
- ngx_http_file_cache_lock_wait_handler
- ngx_http_file_cache_lookup
- ngx_http_file_cache_manage_directory
- ngx_http_file_cache_manage_file
- ngx_http_file_cache_manager
- ngx_http_file_cache_name
- ngx_http_file_cache_new

- ngx_http_file_cache_noop
- ngx_http_file_cache_open
- ngx_http_file_cache_rbtree_insert_value
- ngx_http_file_cache_read
- ngx_http_file_cache_reopen
- ngx_http_file_cache_set_header
- ngx_http_file_cache_set_slot
- ngx_http_file_cache_update
- ngx_http_file_cache_update_header
- ngx_http_file_cache_update_variant
- ngx_http_file_cache_valid
- ngx_http_file_cache_valid_set_slot
- ngx_http_file_cache_vary
- ngx_http_file_cache_vary_header

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <ngx_md5.h>
12
13
14  static ngx_int_t ngx_http_file_cache_lock(ngx_http_request_t *r,
15      ngx_http_cache_t *c);
16  static void ngx_http_file_cache_lock_wait_handler(ngx_event_t *ev);
17  static void ngx_http_file_cache_lock_wait(ngx_http_request_t *r,
18      ngx_http_cache_t *c);
19  static ngx_int_t ngx_http_file_cache_read(ngx_http_request_t *r,
20      ngx_http_cache_t *c);
21  static ssize_t ngx_http_file_cache_aio_read(ngx_http_request_t *r,
22      ngx_http_cache_t *c);
23  #if (NGX_HAVE_FILE_AIO)
24  static void ngx_http_cache_aio_event_handler(ngx_event_t *ev);
25  #endif
26  static ngx_int_t ngx_http_file_cache_exists(ngx_http_file_cache_t *cache,
27      ngx_http_cache_t *c);
28  static ngx_int_t ngx_http_file_cache_name(ngx_http_request_t *r,
29      ngx_path_t *path);
30  static ngx_http_file_cache_node_t *
31      ngx_http_file_cache_lookup(ngx_http_file_cache_t *cache, u_char *key);
32  static void ngx_http_file_cache_rbtree_insert_value(ngx_rbtree_node_t *temp,
33      ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);
34  static void ngx_http_file_cache_vary(ngx_http_request_t *r, u_char *vary,
35      size_t len, u_char *hash);
36  static void ngx_http_file_cache_vary_header(ngx_http_request_t *r,
37      ngx_md5_t *md5, ngx_str_t *name);
38  static ngx_int_t ngx_http_file_cache_reopen(ngx_http_request_t *r,
39      ngx_http_cache_t *c);
40  static ngx_int_t ngx_http_file_cache_update_variant(ngx_http_request_t *r,
41      ngx_http_cache_t *c);
```

```
42  static void ngx_http_file_cache_cleanup(void *data);
43  static time_t ngx_http_file_cache_forced_expire(ngx_http_file_cache_t *cache);
44  static time_t ngx_http_file_cache_expire(ngx_http_file_cache_t *cache);
45  static void ngx_http_file_cache_delete(ngx_http_file_cache_t *cache,
46      ngx_queue_t *q, u_char *name);
47  static void ngx_http_file_cache_loader_sleep(ngx_http_file_cache_t *cache);
48  static ngx_int_t ngx_http_file_cache_noop(ngx_tree_ctx_t *ctx,
49      ngx_str_t *path);
50  static ngx_int_t ngx_http_file_cache_manage_file(ngx_tree_ctx_t *ctx,
51      ngx_str_t *path);
52  static ngx_int_t ngx_http_file_cache_manage_directory(ngx_tree_ctx_t *ctx,
53      ngx_str_t *path);
54  static ngx_int_t ngx_http_file_cache_add_file(ngx_tree_ctx_t *ctx,
55      ngx_str_t *path);
56  static ngx_int_t ngx_http_file_cache_add(ngx_http_file_cache_t *cache,
57      ngx_http_cache_t *c);
58  static ngx_int_t ngx_http_file_cache_delete_file(ngx_tree_ctx_t *ctx,
59      ngx_str_t *path);
60
61
62  ngx_str_t  ngx_http_cache_status[] = {
63      ngx_string("MISS"),
64      ngx_string("BYPASS"),
65      ngx_string("EXPIRED"),
66      ngx_string("STALE"),
67      ngx_string("UPDATING"),
68      ngx_string("REVALIDATED"),
69      ngx_string("HIT")
70  };
71
72
73  static u_char  ngx_http_file_cache_key[] = { LF, 'K', 'E', 'Y', ':', ' ' };
74
75
76  static ngx_int_t
77  ngx_http_file_cache_init(ngx_shm_zone_t *shm_zone, void *data)
78  {
79      ngx_http_file_cache_t  *ocache = data;
80
81      size_t                  len;
82      ngx_uint_t              n;
83      ngx_http_file_cache_t  *cache;
84
85      cache = shm_zone->data;
86
87      if (ocache) {
88          if (ngx_strcmp(cache->path->name.data, ocache->path->name.data) != 0) {
89              ngx_log_error(NGX_LOG_EMERG, shm_zone->shm.log, 0,
90                            "cache \"%V\" uses the \"%V\" cache path "
91                            "while previously it used the \"%V\" cache path",
92                            &shm_zone->shm.name, &cache->path->name,
93                            &ocache->path->name);
94
95              return NGX_ERROR;
96          }
97
98          for (n = 0; n < 3; n++) {
99              if (cache->path->level[n] != ocache->path->level[n]) {
100                 ngx_log_error(NGX_LOG_EMERG, shm_zone->shm.log, 0,
101                               "cache \"%V\" had previously different levels",
102                               &shm_zone->shm.name);
103                 return NGX_ERROR;
104             }
105         }
106
107         cache->sh = ocache->sh;
108
109         cache->shpool = ocache->shpool;
110         cache->bsize = ocache->bsize;
111
112         cache->max_size /= cache->bsize;
113
114         if (!cache->sh->cold || cache->sh->loading) {
115             cache->path->loader = NULL;
116         }
117
```

```
        return NGX_OK;
    }

    cache->shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;

    if (shm_zone->shm.exists) {
        cache->sh = cache->shpool->data;
        cache->bsize = ngx_fs_bsize(cache->path->name.data);

        return NGX_OK;
    }

    cache->sh = ngx_slab_alloc(cache->shpool, sizeof(ngx_http_file_cache_sh_t));
    if (cache->sh == NULL) {
        return NGX_ERROR;
    }

    cache->shpool->data = cache->sh;

    ngx_rbtree_init(&cache->sh->rbtree, &cache->sh->sentinel,
                    ngx_http_file_cache_rbtree_insert_value);

    ngx_queue_init(&cache->sh->queue);

    cache->sh->cold = 1;
    cache->sh->loading = 0;
    cache->sh->size = 0;

    cache->bsize = ngx_fs_bsize(cache->path->name.data);

    cache->max_size /= cache->bsize;

    len = sizeof(" in cache keys zone \"\"") + shm_zone->shm.name.len;

    cache->shpool->log_ctx = ngx_slab_alloc(cache->shpool, len);
    if (cache->shpool->log_ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_sprintf(cache->shpool->log_ctx, " in cache keys zone \"%V\"%Z",
                &shm_zone->shm.name);

    cache->shpool->log_nomem = 0;

    return NGX_OK;
}


ngx_int_t
ngx_http_file_cache_new(ngx_http_request_t *r)
{
    ngx_http_cache_t  *c;

    c = ngx_pcalloc(r->pool, sizeof(ngx_http_cache_t));
    if (c == NULL) {
        return NGX_ERROR;
    }

    if (ngx_array_init(&c->keys, r->pool, 4, sizeof(ngx_str_t)) != NGX_OK) {
        return NGX_ERROR;
    }

    r->cache = c;
    c->file.log = r->connection->log;
    c->file.fd = NGX_INVALID_FILE;

    c->last_modified = -1;

    return NGX_OK;
}


ngx_int_t
ngx_http_file_cache_create(ngx_http_request_t *r)
{
    ngx_http_cache_t       *c;
```

```
    ngx_pool_cleanup_t     *cln;
    ngx_http_file_cache_t  *cache;

    c = r->cache;
    cache = c->file_cache;

    cln = ngx_pool_cleanup_add(r->pool, 0);
    if (cln == NULL) {
        return NGX_ERROR;
    }

    cln->handler = ngx_http_file_cache_cleanup;
    cln->data = c;

    if (ngx_http_file_cache_exists(cache, c) == NGX_ERROR) {
        return NGX_ERROR;
    }

    if (ngx_http_file_cache_name(r, cache->path) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}


void
ngx_http_file_cache_create_key(ngx_http_request_t *r)
{
    size_t             len;
    ngx_str_t         *key;
    ngx_uint_t         i;
    ngx_md5_t          md5;
    ngx_http_cache_t  *c;

    c = r->cache;

    len = 0;

    ngx_crc32_init(c->crc32);
    ngx_md5_init(&md5);

    key = c->keys.elts;
    for (i = 0; i < c->keys.nelts; i++) {
        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http cache key: \"%V\"", &key[i]);

        len += key[i].len;

        ngx_crc32_update(&c->crc32, key[i].data, key[i].len);
        ngx_md5_update(&md5, key[i].data, key[i].len);
    }

    c->header_start = sizeof(ngx_http_file_cache_header_t)
                      + sizeof(ngx_http_file_cache_key) + len + 1;

    ngx_crc32_final(c->crc32);
    ngx_md5_final(c->key, &md5);

    ngx_memcpy(c->main, c->key, NGX_HTTP_CACHE_KEY_LEN);
}


ngx_int_t
ngx_http_file_cache_open(ngx_http_request_t *r)
{
    ngx_int_t                  rc, rv;
    ngx_uint_t                 cold, test;
    ngx_http_cache_t          *c;
    ngx_pool_cleanup_t        *cln;
    ngx_open_file_info_t       of;
    ngx_http_file_cache_t     *cache;
    ngx_http_core_loc_conf_t  *clcf;

    c = r->cache;

```

```
    if (c->waiting) {
        return NGX_AGAIN;
    }

    if (c->reading) {
        return ngx_http_file_cache_read(r, c);
    }

    cache = c->file_cache;

    if (c->node == NULL) {
        cln = ngx_pool_cleanup_add(r->pool, 0);
        if (cln == NULL) {
            return NGX_ERROR;
        }

        cln->handler = ngx_http_file_cache_cleanup;
        cln->data = c;
    }

    rc = ngx_http_file_cache_exists(cache, c);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache exists: %i e:%d", rc, c->exists);

    if (rc == NGX_ERROR) {
        return rc;
    }

    if (rc == NGX_AGAIN) {
        return NGX_HTTP_CACHE_SCARCE;
    }

    cold = cache->sh->cold;

    if (rc == NGX_OK) {

        if (c->error) {
            return c->error;
        }

        c->temp_file = 1;
        test = c->exists ? 1 : 0;
        rv = NGX_DECLINED;

    } else { /* rc == NGX_DECLINED */

        if (c->min_uses > 1) {

            if (!cold) {
                return NGX_HTTP_CACHE_SCARCE;
            }

            test = 1;
            rv = NGX_HTTP_CACHE_SCARCE;

        } else {
            c->temp_file = 1;
            test = cold ? 1 : 0;
            rv = NGX_DECLINED;
        }
    }

    if (ngx_http_file_cache_name(r, cache->path) != NGX_OK) {
        return NGX_ERROR;
    }

    if (!test) {
        goto done;
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.uniq = c->uniq;
```

```
    of.valid = clcf->open_file_cache_valid;
    of.min_uses = clcf->open_file_cache_min_uses;
    of.events = clcf->open_file_cache_events;
    of.directio = NGX_OPEN_FILE_DIRECTIO_OFF;
    of.read_ahead = clcf->read_ahead;

    if (ngx_open_cached_file(clcf->open_file_cache, &c->file.name, &of, r->pool)
        != NGX_OK)
    {
        switch (of.err) {

        case 0:
            return NGX_ERROR;

        case NGX_ENOENT:
        case NGX_ENOTDIR:
            goto done;

        default:
            ngx_log_error(NGX_LOG_CRIT, r->connection->log, of.err,
                          ngx_open_file_n " \"%s\" failed", c->file.name.data);
            return NGX_ERROR;
        }
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache fd: %d", of.fd);

    c->file.fd = of.fd;
    c->file.log = r->connection->log;
    c->uniq = of.uniq;
    c->length = of.size;
    c->fs_size = (of.fs_size + cache->bsize - 1) / cache->bsize;

    c->buf = ngx_create_temp_buf(r->pool, c->body_start);
    if (c->buf == NULL) {
        return NGX_ERROR;
    }

    return ngx_http_file_cache_read(r, c);

done:

    if (rv == NGX_DECLINED) {
        return ngx_http_file_cache_lock(r, c);
    }

    return rv;
}


static ngx_int_t
ngx_http_file_cache_lock(ngx_http_request_t *r, ngx_http_cache_t *c)
{
    ngx_msec_t                now, timer;
    ngx_http_file_cache_t     *cache;

    if (!c->lock) {
        return NGX_DECLINED;
    }

    now = ngx_current_msec;

    cache = c->file_cache;

    ngx_shmtx_lock(&cache->shpool->mutex);

    timer = c->node->lock_time - now;

    if (!c->node->updating || (ngx_msec_int_t) timer <= 0) {
        c->node->updating = 1;
        c->node->lock_time = now + c->lock_age;
        c->updating = 1;
        c->lock_time = c->node->lock_time;
    }

```

```
    ngx_shmtx_unlock(&cache->shpool->mutex);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache lock u:%d wt:%M",
                   c->updating, c->wait_time);

    if (c->updating) {
        return NGX_DECLINED;
    }

    if (c->lock_timeout == 0) {
        return NGX_HTTP_CACHE_SCARCE;
    }

    c->waiting = 1;

    if (c->wait_time == 0) {
        c->wait_time = now + c->lock_timeout;

        c->wait_event.handler = ngx_http_file_cache_lock_wait_handler;
        c->wait_event.data = r;
        c->wait_event.log = r->connection->log;
    }

    timer = c->wait_time - now;

    ngx_add_timer(&c->wait_event, (timer > 500) ? 500 : timer);

    r->main->blocked++;

    return NGX_AGAIN;
}


static void
ngx_http_file_cache_lock_wait_handler(ngx_event_t *ev)
{
    ngx_connection_t    *c;
    ngx_http_request_t  *r;

    r = ev->data;
    c = r->connection;

    ngx_http_set_log_request(c->log, r);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
                   "http file cache wait: \"%V?%V\"", &r->uri, &r->args);

    ngx_http_file_cache_lock_wait(r, r->cache);

    ngx_http_run_posted_requests(c);
}


static void
ngx_http_file_cache_lock_wait(ngx_http_request_t *r, ngx_http_cache_t *c)
{
    ngx_uint_t              wait;
    ngx_msec_t              now, timer;
    ngx_http_file_cache_t  *cache;

    now = ngx_current_msec;

    timer = c->wait_time - now;

    if ((ngx_msec_int_t) timer <= 0) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "cache lock timeout");
        c->lock_timeout = 0;
        goto wakeup;
    }

    cache = c->file_cache;
    wait = 0;

    ngx_shmtx_lock(&cache->shpool->mutex);
```

```
498
499     timer = c->node->lock_time - now;
500
501     if (c->node->updating && (ngx_msec_int_t) timer > 0) {
502         wait = 1;
503     }
504
505     ngx_shmtx_unlock(&cache->shpool->mutex);
506
507     if (wait) {
508         ngx_add_timer(&c->wait_event, (timer > 500) ? 500 : timer);
509         return;
510     }
511
512 wakeup:
513
514     c->waiting = 0;
515     r->main->blocked--;
516     r->write_event_handler(r);
517 }
518
519
520 static ngx_int_t
521 ngx_http_file_cache_read(ngx_http_request_t *r, ngx_http_cache_t *c)
522 {
523     time_t                        now;
524     ssize_t                       n;
525     ngx_int_t                     rc;
526     ngx_http_file_cache_t        *cache;
527     ngx_http_file_cache_header_t  *h;
528
529     n = ngx_http_file_cache_aio_read(r, c);
530
531     if (n < 0) {
532         return n;
533     }
534
535     if ((size_t) n < c->header_start) {
536         ngx_log_error(NGX_LOG_CRIT, r->connection->log, 0,
537                       "cache file \"%s\" is too small", c->file.name.data);
538         return NGX_DECLINED;
539     }
540
541     h = (ngx_http_file_cache_header_t *) c->buf->pos;
542
543     if (h->version != NGX_HTTP_CACHE_VERSION) {
544         ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
545                       "cache file \"%s\" version mismatch", c->file.name.data);
546         return NGX_DECLINED;
547     }
548
549     if (h->crc32 != c->crc32) {
550         ngx_log_error(NGX_LOG_CRIT, r->connection->log, 0,
551                       "cache file \"%s\" has md5 collision", c->file.name.data);
552         return NGX_DECLINED;
553     }
554
555     if ((size_t) h->body_start > c->body_start) {
556         ngx_log_error(NGX_LOG_CRIT, r->connection->log, 0,
557                       "cache file \"%s\" has too long header",
558                       c->file.name.data);
559         return NGX_DECLINED;
560     }
561
562     if (h->vary_len > NGX_HTTP_CACHE_VARY_LEN) {
563         ngx_log_error(NGX_LOG_CRIT, r->connection->log, 0,
564                       "cache file \"%s\" has incorrect vary length",
565                       c->file.name.data);
566         return NGX_DECLINED;
567     }
568
569     if (h->vary_len) {
570         ngx_http_file_cache_vary(r, h->vary, h->vary_len, c->variant);
571
572         if (ngx_memcmp(c->variant, h->variant, NGX_HTTP_CACHE_KEY_LEN) != 0) {
573             ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
```

```
                          "http file cache vary mismatch");
            return ngx_http_file_cache_reopen(r, c);
        }
    }

    c->buf->last += n;

    c->valid_sec = h->valid_sec;
    c->last_modified = h->last_modified;
    c->date = h->date;
    c->valid_msec = h->valid_msec;
    c->header_start = h->header_start;
    c->body_start = h->body_start;
    c->etag.len = h->etag_len;
    c->etag.data = h->etag;

    r->cached = 1;

    cache = c->file_cache;

    if (cache->sh->cold) {

        ngx_shmtx_lock(&cache->shpool->mutex);

        if (!c->node->exists) {
            c->node->uses = 1;
            c->node->body_start = c->body_start;
            c->node->exists = 1;
            c->node->uniq = c->uniq;
            c->node->fs_size = c->fs_size;

            cache->sh->size += c->fs_size;
        }

        ngx_shmtx_unlock(&cache->shpool->mutex);
    }

    now = ngx_time();

    if (c->valid_sec < now) {

        ngx_shmtx_lock(&cache->shpool->mutex);

        if (c->node->updating) {
            rc = NGX_HTTP_CACHE_UPDATING;

        } else {
            c->node->updating = 1;
            c->updating = 1;
            c->lock_time = c->node->lock_time;
            rc = NGX_HTTP_CACHE_STALE;
        }

        ngx_shmtx_unlock(&cache->shpool->mutex);

        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http file cache expired: %i %T %T",
                       rc, c->valid_sec, now);

        return rc;
    }

    return NGX_OK;
}


static ssize_t
ngx_http_file_cache_aio_read(ngx_http_request_t *r, ngx_http_cache_t *c)
{
#if (NGX_HAVE_FILE_AIO)
    ssize_t                    n;
    ngx_http_core_loc_conf_t  *clcf;

    if (!ngx_file_aio) {
        goto noaio;
    }
```

```
650
651     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
652
653     if (!clcf->aio) {
654         goto noaio;
655     }
656
657     n = ngx_file_aio_read(&c->file, c->buf->pos, c->body_start, 0, r->pool);
658
659     if (n != NGX_AGAIN) {
660         c->reading = 0;
661         return n;
662     }
663
664     c->reading = 1;
665
666     c->file.aio->data = r;
667     c->file.aio->handler = ngx_http_cache_aio_event_handler;
668
669     r->main->blocked++;
670     r->aio = 1;
671
672     return NGX_AGAIN;
673
674 noaio:
675
676 #endif
677
678     return ngx_read_file(&c->file, c->buf->pos, c->body_start, 0);
679 }
680
681
682 #if (NGX_HAVE_FILE_AIO)
683
684 static void
685 ngx_http_cache_aio_event_handler(ngx_event_t *ev)
686 {
687     ngx_event_aio_t     *aio;
688     ngx_connection_t    *c;
689     ngx_http_request_t  *r;
690
691     aio = ev->data;
692     r = aio->data;
693     c = r->connection;
694
695     ngx_http_set_log_request(c->log, r);
696
697     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
698                    "http file cache aio: \"%V?%V\"", &r->uri, &r->args);
699
700     r->main->blocked--;
701     r->aio = 0;
702
703     r->write_event_handler(r);
704
705     ngx_http_run_posted_requests(c);
706 }
707
708 #endif
709
710
711 static ngx_int_t
712 ngx_http_file_cache_exists(ngx_http_file_cache_t *cache, ngx_http_cache_t *c)
713 {
714     ngx_int_t                    rc;
715     ngx_http_file_cache_node_t  *fcn;
716
717     ngx_shmtx_lock(&cache->shpool->mutex);
718
719     fcn = c->node;
720
721     if (fcn == NULL) {
722         fcn = ngx_http_file_cache_lookup(cache, c->key);
723     }
724
725     if (fcn) {
```

```
        ngx_queue_remove(&fcn->queue);

        if (c->node == NULL) {
            fcn->uses++;
            fcn->count++;
        }

        if (fcn->error) {

            if (fcn->valid_sec < ngx_time()) {
                goto renew;
            }

            rc = NGX_OK;

            goto done;
        }

        if (fcn->exists || fcn->uses >= c->min_uses) {

            c->exists = fcn->exists;
            if (fcn->body_start) {
                c->body_start = fcn->body_start;
            }

            rc = NGX_OK;

            goto done;
        }

        rc = NGX_AGAIN;

        goto done;
    }

    fcn = ngx_slab_calloc_locked(cache->shpool,
                                 sizeof(ngx_http_file_cache_node_t));
    if (fcn == NULL) {
        ngx_shmtx_unlock(&cache->shpool->mutex);

        (void) ngx_http_file_cache_forced_expire(cache);

        ngx_shmtx_lock(&cache->shpool->mutex);

        fcn = ngx_slab_calloc_locked(cache->shpool,
                                     sizeof(ngx_http_file_cache_node_t));
        if (fcn == NULL) {
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                          "could not allocate node%s", cache->shpool->log_ctx);
            rc = NGX_ERROR;
            goto failed;
        }
    }

    ngx_memcpy((u_char *) &fcn->node.key, c->key, sizeof(ngx_rbtree_key_t));

    ngx_memcpy(fcn->key, &c->key[sizeof(ngx_rbtree_key_t)],
               NGX_HTTP_CACHE_KEY_LEN - sizeof(ngx_rbtree_key_t));

    ngx_rbtree_insert(&cache->sh->rbtree, &fcn->node);

    fcn->uses = 1;
    fcn->count = 1;

renew:

    rc = NGX_DECLINED;

    fcn->valid_msec = 0;
    fcn->error = 0;
    fcn->exists = 0;
    fcn->valid_sec = 0;
    fcn->uniq = 0;
    fcn->body_start = 0;
    fcn->fs_size = 0;

```

```
done:

    fcn->expire = ngx_time() + cache->inactive;

    ngx_queue_insert_head(&cache->sh->queue, &fcn->queue);

    c->uniq = fcn->uniq;
    c->error = fcn->error;
    c->node = fcn;

failed:

    ngx_shmtx_unlock(&cache->shpool->mutex);

    return rc;
}


static ngx_int_t
ngx_http_file_cache_name(ngx_http_request_t *r, ngx_path_t *path)
{
    u_char            *p;
    ngx_http_cache_t  *c;

    c = r->cache;

    if (c->file.name.len) {
        return NGX_OK;
    }

    c->file.name.len = path->name.len + 1 + path->len
                       + 2 * NGX_HTTP_CACHE_KEY_LEN;

    c->file.name.data = ngx_pnalloc(r->pool, c->file.name.len + 1);
    if (c->file.name.data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(c->file.name.data, path->name.data, path->name.len);

    p = c->file.name.data + path->name.len + 1 + path->len;
    p = ngx_hex_dump(p, c->key, NGX_HTTP_CACHE_KEY_LEN);
    *p = '\0';

    ngx_create_hashed_filename(path, c->file.name.data, c->file.name.len);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "cache file: \"%s\"", c->file.name.data);

    return NGX_OK;
}


static ngx_http_file_cache_node_t *
ngx_http_file_cache_lookup(ngx_http_file_cache_t *cache, u_char *key)
{
    ngx_int_t                   rc;
    ngx_rbtree_key_t            node_key;
    ngx_rbtree_node_t           *node, *sentinel;
    ngx_http_file_cache_node_t  *fcn;

    ngx_memcpy((u_char *) &node_key, key, sizeof(ngx_rbtree_key_t));

    node = cache->sh->rbtree.root;
    sentinel = cache->sh->rbtree.sentinel;

    while (node != sentinel) {

        if (node_key < node->key) {
            node = node->left;
            continue;
        }

        if (node_key > node->key) {
            node = node->right;
            continue;
```

```
        }

        /* node_key == node->key */

        fcn = (ngx_http_file_cache_node_t *) node;

        rc = ngx_memcmp(&key[sizeof(ngx_rbtree_key_t)], fcn->key,
                        NGX_HTTP_CACHE_KEY_LEN - sizeof(ngx_rbtree_key_t));

        if (rc == 0) {
            return fcn;
        }

        node = (rc < 0) ? node->left : node->right;
    }

    /* not found */

    return NULL;
}


static void
ngx_http_file_cache_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
    ngx_rbtree_node_t           **p;
    ngx_http_file_cache_node_t   *cn, *cnt;

    for ( ;; ) {

        if (node->key < temp->key) {

            p = &temp->left;

        } else if (node->key > temp->key) {

            p = &temp->right;

        } else { /* node->key == temp->key */

            cn = (ngx_http_file_cache_node_t *) node;
            cnt = (ngx_http_file_cache_node_t *) temp;

            p = (ngx_memcmp(cn->key, cnt->key,
                            NGX_HTTP_CACHE_KEY_LEN - sizeof(ngx_rbtree_key_t))
                 < 0)
                ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
    ngx_rbt_red(node);
}


static void
ngx_http_file_cache_vary(ngx_http_request_t *r, u_char *vary, size_t len,
    u_char *hash)
{
    u_char     *p, *last;
    ngx_str_t   name;
    ngx_md5_t   md5;
    u_char      buf[NGX_HTTP_CACHE_VARY_LEN];

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache vary: \"%*s\"", len, vary);
```

```
954
955     ngx_md5_init(&md5);
956     ngx_md5_update(&md5, r->cache->main, NGX_HTTP_CACHE_KEY_LEN);
957
958     ngx_strlow(buf, vary, len);
959
960     p = buf;
961     last = buf + len;
962
963     while (p < last) {
964
965         while (p < last && (*p == ' ' || *p == ',')) { p++; }
966
967         name.data = p;
968
969         while (p < last && *p != ',' && *p != ' ') { p++; }
970
971         name.len = p - name.data;
972
973         if (name.len == 0) {
974             break;
975         }
976
977         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
978                        "http file cache vary: %V", &name);
979
980         ngx_md5_update(&md5, name.data, name.len);
981         ngx_md5_update(&md5, (u_char *) ":", sizeof(":") - 1);
982
983         ngx_http_file_cache_vary_header(r, &md5, &name);
984
985         ngx_md5_update(&md5, (u_char *) CRLF, sizeof(CRLF) - 1);
986     }
987
988     ngx_md5_final(hash, &md5);
989 }
990
991
992 static void
993 ngx_http_file_cache_vary_header(ngx_http_request_t *r, ngx_md5_t *md5,
994     ngx_str_t *name)
995 {
996     size_t            len;
997     u_char           *p, *start, *last;
998     ngx_uint_t        i, multiple, normalize;
999     ngx_list_part_t  *part;
1000     ngx_table_elt_t  *header;
1001
1002     multiple = 0;
1003     normalize = 0;
1004
1005     if (name->len == sizeof("Accept-Charset") - 1
1006         && ngx_strncasecmp(name->data, (u_char *) "Accept-Charset",
1007                            sizeof("Accept-Charset") - 1) == 0)
1008     {
1009         normalize = 1;
1010
1011     } else if (name->len == sizeof("Accept-Encoding") - 1
1012         && ngx_strncasecmp(name->data, (u_char *) "Accept-Encoding",
1013                            sizeof("Accept-Encoding") - 1) == 0)
1014     {
1015         normalize = 1;
1016
1017     } else if (name->len == sizeof("Accept-Language") - 1
1018         && ngx_strncasecmp(name->data, (u_char *) "Accept-Language",
1019                            sizeof("Accept-Language") - 1) == 0)
1020     {
1021         normalize = 1;
1022     }
1023
1024     part = &r->headers_in.headers.part;
1025     header = part->elts;
1026
1027     for (i = 0; /* void */ ; i++) {
1028
1029         if (i >= part->nelts) {
```

```
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            header = part->elts;
            i = 0;
        }

        if (header[i].hash == 0) {
            continue;
        }

        if (header[i].key.len != name->len) {
            continue;
        }

        if (ngx_strncasecmp(header[i].key.data, name->data, name->len) != 0) {
            continue;
        }

        if (!normalize) {

            if (multiple) {
                ngx_md5_update(md5, (u_char *) ",", sizeof(",") - 1);
            }

            ngx_md5_update(md5, header[i].value.data, header[i].value.len);

            multiple = 1;

            continue;
        }

        /* normalize spaces */

        p = header[i].value.data;
        last = p + header[i].value.len;

        while (p < last) {

            while (p < last && (*p == ' ' || *p == ',')) { p++; }

            start = p;

            while (p < last && *p != ',' && *p != ' ') { p++; }

            len = p - start;

            if (len == 0) {
                break;
            }

            if (multiple) {
                ngx_md5_update(md5, (u_char *) ",", sizeof(",") - 1);
            }

            ngx_md5_update(md5, start, len);

            multiple = 1;
        }
    }
}


static ngx_int_t
ngx_http_file_cache_reopen(ngx_http_request_t *r, ngx_http_cache_t *c)
{
    ngx_http_file_cache_t  *cache;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->file.log, 0,
                   "http file cache reopen");

    if (c->secondary) {
        ngx_log_error(NGX_LOG_CRIT, r->connection->log, 0,
                      "cache file \"%s\" has incorrect vary hash",
```

```
                      c->file.name.data);
        return NGX_DECLINED;
    }

    cache = c->file_cache;

    ngx_shmtx_lock(&cache->shpool->mutex);

    c->node->count--;
    c->node = NULL;

    ngx_shmtx_unlock(&cache->shpool->mutex);

    c->secondary = 1;
    c->file.name.len = 0;
    c->body_start = c->buf->end - c->buf->start;

    ngx_memcpy(c->key, c->variant, NGX_HTTP_CACHE_KEY_LEN);

    return ngx_http_file_cache_open(r);
}


ngx_int_t
ngx_http_file_cache_set_header(ngx_http_request_t *r, u_char *buf)
{
    ngx_http_file_cache_header_t  *h = (ngx_http_file_cache_header_t *) buf;

    u_char            *p;
    ngx_str_t         *key;
    ngx_uint_t         i;
    ngx_http_cache_t  *c;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache set header");

    c = r->cache;

    ngx_memzero(h, sizeof(ngx_http_file_cache_header_t));

    h->version = NGX_HTTP_CACHE_VERSION;
    h->valid_sec = c->valid_sec;
    h->last_modified = c->last_modified;
    h->date = c->date;
    h->crc32 = c->crc32;
    h->valid_msec = (u_short) c->valid_msec;
    h->header_start = (u_short) c->header_start;
    h->body_start = (u_short) c->body_start;

    if (c->etag.len <= NGX_HTTP_CACHE_ETAG_LEN) {
        h->etag_len = (u_char) c->etag.len;
        ngx_memcpy(h->etag, c->etag.data, c->etag.len);
    }

    if (c->vary.len) {
        if (c->vary.len > NGX_HTTP_CACHE_VARY_LEN) {
            /* should not happen */
            c->vary.len = NGX_HTTP_CACHE_VARY_LEN;
        }

        h->vary_len = (u_char) c->vary.len;
        ngx_memcpy(h->vary, c->vary.data, c->vary.len);

        ngx_http_file_cache_vary(r, c->vary.data, c->vary.len, c->variant);
        ngx_memcpy(h->variant, c->variant, NGX_HTTP_CACHE_KEY_LEN);
    }

    if (ngx_http_file_cache_update_variant(r, c) != NGX_OK) {
        return NGX_ERROR;
    }

    p = buf + sizeof(ngx_http_file_cache_header_t);

    p = ngx_cpymem(p, ngx_http_file_cache_key, sizeof(ngx_http_file_cache_key));

    key = c->keys.elts;
```

```
    for (i = 0; i < c->keys.nelts; i++) {
        p = ngx_copy(p, key[i].data, key[i].len);
    }

    *p = LF;

    return NGX_OK;
}


static ngx_int_t
ngx_http_file_cache_update_variant(ngx_http_request_t *r, ngx_http_cache_t *c)
{
    ngx_http_file_cache_t  *cache;

    if (!c->secondary) {
        return NGX_OK;
    }

    if (c->vary.len
        && ngx_memcmp(c->variant, c->key, NGX_HTTP_CACHE_KEY_LEN) == 0)
    {
        return NGX_OK;
    }

    /*
     * if the variant hash doesn't match one we used as a secondary
     * cache key, switch back to the original key
     */

    cache = c->file_cache;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache main key");

    ngx_shmtx_lock(&cache->shpool->mutex);

    c->node->count--;
    c->node->updating = 0;
    c->node = NULL;

    ngx_shmtx_unlock(&cache->shpool->mutex);

    c->file.name.len = 0;

    ngx_memcpy(c->key, c->main, NGX_HTTP_CACHE_KEY_LEN);

    if (ngx_http_file_cache_exists(cache, c) == NGX_ERROR) {
        return NGX_ERROR;
    }

    if (ngx_http_file_cache_name(r, cache->path) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}


void
ngx_http_file_cache_update(ngx_http_request_t *r, ngx_temp_file_t *tf)
{
    off_t                  fs_size;
    ngx_int_t              rc;
    ngx_file_uniq_t        uniq;
    ngx_file_info_t        fi;
    ngx_http_cache_t       *c;
    ngx_ext_rename_file_t  ext;
    ngx_http_file_cache_t  *cache;

    c = r->cache;

    if (c->updated) {
        return;
    }
```

```
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache update");

    cache = c->file_cache;

    c->updated = 1;
    c->updating = 0;

    uniq = 0;
    fs_size = 0;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache rename: \"%s\" to \"%s\"",
                   tf->file.name.data, c->file.name.data);

    ext.access = NGX_FILE_OWNER_ACCESS;
    ext.path_access = NGX_FILE_OWNER_ACCESS;
    ext.time = -1;
    ext.create_path = 1;
    ext.delete_file = 1;
    ext.log = r->connection->log;

    rc = ngx_ext_rename_file(&tf->file.name, &c->file.name, &ext);

    if (rc == NGX_OK) {

        if (ngx_fd_info(tf->file.fd, &fi) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
                          ngx_fd_info_n " \"%s\" failed", tf->file.name.data);

            rc = NGX_ERROR;

        } else {
            uniq = ngx_file_uniq(&fi);
            fs_size = (ngx_file_fs_size(&fi) + cache->bsize - 1) / cache->bsize;
        }
    }

    ngx_shmtx_lock(&cache->shpool->mutex);

    c->node->count--;
    c->node->uniq = uniq;
    c->node->body_start = c->body_start;

    cache->sh->size += fs_size - c->node->fs_size;
    c->node->fs_size = fs_size;

    if (rc == NGX_OK) {
        c->node->exists = 1;
    }

    c->node->updating = 0;

    ngx_shmtx_unlock(&cache->shpool->mutex);
}


void
ngx_http_file_cache_update_header(ngx_http_request_t *r)
{
    ssize_t                     n;
    ngx_err_t                   err;
    ngx_file_t                  file;
    ngx_file_info_t             fi;
    ngx_http_cache_t           *c;
    ngx_http_file_cache_header_t  h;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache update header");

    c = r->cache;

    ngx_memzero(&file, sizeof(ngx_file_t));

    file.name = c->file.name;
    file.log = r->connection->log;
```

```
    file.fd = ngx_open_file(file.name.data, NGX_FILE_RDWR, NGX_FILE_OPEN, 0);

    if (file.fd == NGX_INVALID_FILE) {
        err = ngx_errno;

        /* cache file may have been deleted */

        if (err == NGX_ENOENT) {
            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http file cache \"%s\" not found",
                           file.name.data);
            return;
        }

        ngx_log_error(NGX_LOG_CRIT, r->connection->log, err,
                      ngx_open_file_n " \"%s\" failed", file.name.data);
        return;
    }

    /*
     * make sure cache file wasn't replaced;
     * if it was, do nothing
     */

    if (ngx_fd_info(file.fd, &fi) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
                      ngx_fd_info_n " \"%s\" failed", file.name.data);
        goto done;
    }

    if (c->uniq != ngx_file_uniq(&fi)
        || c->length != ngx_file_size(&fi))
    {
        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http file cache \"%s\" changed",
                       file.name.data);
        goto done;
    }

    n = ngx_read_file(&file, (u_char *) &h,
                      sizeof(ngx_http_file_cache_header_t), 0);

    if (n == NGX_ERROR) {
        goto done;
    }

    if ((size_t) n != sizeof(ngx_http_file_cache_header_t)) {
        ngx_log_error(NGX_LOG_CRIT, r->connection->log, 0,
                      ngx_read_file_n " read only %z of %z from \"%s\"",
                      n, sizeof(ngx_http_file_cache_header_t), file.name.data);
        goto done;
    }

    if (h.version != NGX_HTTP_CACHE_VERSION
        || h.last_modified != c->last_modified
        || h.crc32 != c->crc32
        || h.header_start != c->header_start
        || h.body_start != c->body_start)
    {
        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http file cache \"%s\" content changed",
                       file.name.data);
        goto done;
    }

    /*
     * update cache file header with new data,
     * notably h.valid_sec and h.date
     */

    ngx_memzero(&h, sizeof(ngx_http_file_cache_header_t));

    h.version = NGX_HTTP_CACHE_VERSION;
    h.valid_sec = c->valid_sec;
    h.last_modified = c->last_modified;
    h.date = c->date;
```

```
    h.crc32 = c->crc32;
    h.valid_msec = (u_short) c->valid_msec;
    h.header_start = (u_short) c->header_start;
    h.body_start = (u_short) c->body_start;

    if (c->etag.len <= NGX_HTTP_CACHE_ETAG_LEN) {
        h.etag_len = (u_char) c->etag.len;
        ngx_memcpy(h.etag, c->etag.data, c->etag.len);
    }

    if (c->vary.len) {
        if (c->vary.len > NGX_HTTP_CACHE_VARY_LEN) {
            /* should not happen */
            c->vary.len = NGX_HTTP_CACHE_VARY_LEN;
        }

        h.vary_len = (u_char) c->vary.len;
        ngx_memcpy(h.vary, c->vary.data, c->vary.len);

        ngx_http_file_cache_vary(r, c->vary.data, c->vary.len, c->variant);
        ngx_memcpy(h.variant, c->variant, NGX_HTTP_CACHE_KEY_LEN);
    }

    (void) ngx_write_file(&file, (u_char *) &h,
                          sizeof(ngx_http_file_cache_header_t), 0);

done:

    if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", file.name.data);
    }
}


ngx_int_t
ngx_http_cache_send(ngx_http_request_t *r)
{
    ngx_int_t          rc;
    ngx_buf_t         *b;
    ngx_chain_t        out;
    ngx_http_cache_t  *c;

    c = r->cache;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http file cache send: %s", c->file.name.data);

    if (r != r->main && c->length - c->body_start == 0) {
        return ngx_http_send_header(r);
    }

    /* we need to allocate all before the header would be sent */

    b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
    if (b == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    b->file = ngx_pcalloc(r->pool, sizeof(ngx_file_t));
    if (b->file == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
        return rc;
    }

    b->file_pos = c->body_start;
    b->file_last = c->length;

    b->in_file = (c->length - c->body_start) ? 1: 0;
    b->last_buf = (r == r->main) ? 1: 0;
    b->last_in_chain = 1;
```

```
1486
1487      b->file->fd = c->file.fd;
1488      b->file->name = c->file.name;
1489      b->file->log = r->connection->log;
1490
1491      out.buf = b;
1492      out.next = NULL;
1493
1494      return ngx_http_output_filter(r, &out);
1495  }
1496
1497
1498  void
1499  ngx_http_file_cache_free(ngx_http_cache_t *c, ngx_temp_file_t *tf)
1500  {
1501      ngx_http_file_cache_t       *cache;
1502      ngx_http_file_cache_node_t  *fcn;
1503
1504      if (c->updated || c->node == NULL) {
1505          return;
1506      }
1507
1508      cache = c->file_cache;
1509
1510      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->file.log, 0,
1511                     "http file cache free, fd: %d", c->file.fd);
1512
1513      ngx_shmtx_lock(&cache->shpool->mutex);
1514
1515      fcn = c->node;
1516      fcn->count--;
1517
1518      if (c->updating && fcn->lock_time == c->lock_time) {
1519          fcn->updating = 0;
1520      }
1521
1522      if (c->error) {
1523          fcn->error = c->error;
1524
1525          if (c->valid_sec) {
1526              fcn->valid_sec = c->valid_sec;
1527              fcn->valid_msec = c->valid_msec;
1528          }
1529
1530      } else if (!fcn->exists && fcn->count == 0 && c->min_uses == 1) {
1531          ngx_queue_remove(&fcn->queue);
1532          ngx_rbtree_delete(&cache->sh->rbtree, &fcn->node);
1533          ngx_slab_free_locked(cache->shpool, fcn);
1534          c->node = NULL;
1535      }
1536
1537      ngx_shmtx_unlock(&cache->shpool->mutex);
1538
1539      c->updated = 1;
1540      c->updating = 0;
1541
1542      if (c->temp_file) {
1543          if (tf && tf->file.fd != NGX_INVALID_FILE) {
1544              ngx_log_debug1(NGX_LOG_DEBUG_HTTP, c->file.log, 0,
1545                             "http file cache incomplete: \"%s\"",
1546                             tf->file.name.data);
1547
1548              if (ngx_delete_file(tf->file.name.data) == NGX_FILE_ERROR) {
1549                  ngx_log_error(NGX_LOG_CRIT, c->file.log, ngx_errno,
1550                                ngx_delete_file_n " \"%s\" failed",
1551                                tf->file.name.data);
1552              }
1553          }
1554      }
1555
1556      if (c->wait_event.timer_set) {
1557          ngx_del_timer(&c->wait_event);
1558      }
1559  }
1560
1561
```

```
1562 static void
1563 ngx_http_file_cache_cleanup(void *data)
1564 {
1565     ngx_http_cache_t  *c = data;
1566
1567     if (c->updated) {
1568         return;
1569     }
1570
1571     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, c->file.log, 0,
1572                    "http file cache cleanup");
1573
1574     if (c->updating) {
1575         ngx_log_error(NGX_LOG_ALERT, c->file.log, 0,
1576                       "stalled cache updating, error:%ui", c->error);
1577     }
1578
1579     ngx_http_file_cache_free(c, NULL);
1580 }
1581
1582
1583 static time_t
1584 ngx_http_file_cache_forced_expire(ngx_http_file_cache_t *cache)
1585 {
1586     u_char                     *name;
1587     size_t                      len;
1588     time_t                      wait;
1589     ngx_uint_t                  tries;
1590     ngx_path_t                 *path;
1591     ngx_queue_t                *q;
1592     ngx_http_file_cache_node_t  *fcn;
1593
1594     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
1595                    "http file cache forced expire");
1596
1597     path = cache->path;
1598     len = path->name.len + 1 + path->len + 2 * NGX_HTTP_CACHE_KEY_LEN;
1599
1600     name = ngx_alloc(len + 1, ngx_cycle->log);
1601     if (name == NULL) {
1602         return 10;
1603     }
1604
1605     ngx_memcpy(name, path->name.data, path->name.len);
1606
1607     wait = 10;
1608     tries = 20;
1609
1610     ngx_shmtx_lock(&cache->shpool->mutex);
1611
1612     for (q = ngx_queue_last(&cache->sh->queue);
1613          q != ngx_queue_sentinel(&cache->sh->queue);
1614          q = ngx_queue_prev(q))
1615     {
1616         fcn = ngx_queue_data(q, ngx_http_file_cache_node_t, queue);
1617
1618         ngx_log_debug6(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
1619                        "http file cache forced expire: #%d %d %02xd%02xd%02xd%02xd",
1620                        fcn->count, fcn->exists,
1621                        fcn->key[0], fcn->key[1], fcn->key[2], fcn->key[3]);
1622
1623         if (fcn->count == 0) {
1624             ngx_http_file_cache_delete(cache, q, name);
1625             wait = 0;
1626
1627         } else {
1628             if (--tries) {
1629                 continue;
1630             }
1631
1632             wait = 1;
1633         }
1634
1635         break;
1636     }
1637
```

```
    ngx_shmtx_unlock(&cache->shpool->mutex);

    ngx_free(name);

    return wait;
}


static time_t
ngx_http_file_cache_expire(ngx_http_file_cache_t *cache)
{
    u_char                      *name, *p;
    size_t                       len;
    time_t                       now, wait;
    ngx_path_t                  *path;
    ngx_queue_t                 *q;
    ngx_http_file_cache_node_t  *fcn;
    u_char                       key[2 * NGX_HTTP_CACHE_KEY_LEN];

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
                   "http file cache expire");

    path = cache->path;
    len = path->name.len + 1 + path->len + 2 * NGX_HTTP_CACHE_KEY_LEN;

    name = ngx_alloc(len + 1, ngx_cycle->log);
    if (name == NULL) {
        return 10;
    }

    ngx_memcpy(name, path->name.data, path->name.len);

    now = ngx_time();

    ngx_shmtx_lock(&cache->shpool->mutex);

    for ( ;; ) {

        if (ngx_quit || ngx_terminate) {
            wait = 1;
            break;
        }

        if (ngx_queue_empty(&cache->sh->queue)) {
            wait = 10;
            break;
        }

        q = ngx_queue_last(&cache->sh->queue);

        fcn = ngx_queue_data(q, ngx_http_file_cache_node_t, queue);

        wait = fcn->expire - now;

        if (wait > 0) {
            wait = wait > 10 ? 10 : wait;
            break;
        }

        ngx_log_debug6(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
                       "http file cache expire: #%d %d %02xd%02xd%02xd%02xd",
                       fcn->count, fcn->exists,
                       fcn->key[0], fcn->key[1], fcn->key[2], fcn->key[3]);

        if (fcn->count == 0) {
            ngx_http_file_cache_delete(cache, q, name);
            continue;
        }

        if (fcn->deleting) {
            wait = 1;
            break;
        }

        p = ngx_hex_dump(key, (u_char *) &fcn->node.key,
                         sizeof(ngx_rbtree_key_t));
```

```
        len = NGX_HTTP_CACHE_KEY_LEN - sizeof(ngx_rbtree_key_t);
        (void) ngx_hex_dump(p, fcn->key, len);

        /*
         * abnormally exited workers may leave locked cache entries,
         * and although it may be safe to remove them completely,
         * we prefer to just move them to the top of the inactive queue
         */

        ngx_queue_remove(q);
        fcn->expire = ngx_time() + cache->inactive;
        ngx_queue_insert_head(&cache->sh->queue, &fcn->queue);

        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                      "ignore long locked inactive cache entry %*s, count:%d",
                      2 * NGX_HTTP_CACHE_KEY_LEN, key, fcn->count);
    }

    ngx_shmtx_unlock(&cache->shpool->mutex);

    ngx_free(name);

    return wait;
}


static void
ngx_http_file_cache_delete(ngx_http_file_cache_t *cache, ngx_queue_t *q,
    u_char *name)
{
    u_char                     *p;
    size_t                      len;
    ngx_path_t                 *path;
    ngx_http_file_cache_node_t  *fcn;

    fcn = ngx_queue_data(q, ngx_http_file_cache_node_t, queue);

    if (fcn->exists) {
        cache->sh->size -= fcn->fs_size;

        path = cache->path;
        p = name + path->name.len + 1 + path->len;
        p = ngx_hex_dump(p, (u_char *) &fcn->node.key,
                         sizeof(ngx_rbtree_key_t));
        len = NGX_HTTP_CACHE_KEY_LEN - sizeof(ngx_rbtree_key_t);
        p = ngx_hex_dump(p, fcn->key, len);
        *p = '\0';

        fcn->count++;
        fcn->deleting = 1;
        ngx_shmtx_unlock(&cache->shpool->mutex);

        len = path->name.len + 1 + path->len + 2 * NGX_HTTP_CACHE_KEY_LEN;
        ngx_create_hashed_filename(path, name, len);

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
                       "http file cache expire: \"%s\"", name);

        if (ngx_delete_file(name) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_CRIT, ngx_cycle->log, ngx_errno,
                          ngx_delete_file_n " \"%s\" failed", name);
        }

        ngx_shmtx_lock(&cache->shpool->mutex);
        fcn->count--;
        fcn->deleting = 0;
    }

    if (fcn->count == 0) {
        ngx_queue_remove(q);
        ngx_rbtree_delete(&cache->sh->rbtree, &fcn->node);
        ngx_slab_free_locked(cache->shpool, fcn);
    }
}
```

```c
static time_t
ngx_http_file_cache_manager(void *data)
{
    ngx_http_file_cache_t  *cache = data;

    off_t   size;
    time_t  next, wait;

    next = ngx_http_file_cache_expire(cache);

    cache->last = ngx_current_msec;
    cache->files = 0;

    for ( ;; ) {
        ngx_shmtx_lock(&cache->shpool->mutex);

        size = cache->sh->size;

        ngx_shmtx_unlock(&cache->shpool->mutex);

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
                       "http file cache size: %O", size);

        if (size < cache->max_size) {
            return next;
        }

        wait = ngx_http_file_cache_forced_expire(cache);

        if (wait > 0) {
            return wait;
        }

        if (ngx_quit || ngx_terminate) {
            return next;
        }
    }
}


static void
ngx_http_file_cache_loader(void *data)
{
    ngx_http_file_cache_t  *cache = data;

    ngx_tree_ctx_t  tree;

    if (!cache->sh->cold || cache->sh->loading) {
        return;
    }

    if (!ngx_atomic_cmp_set(&cache->sh->loading, 0, ngx_pid)) {
        return;
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
                   "http file cache loader");

    tree.init_handler = NULL;
    tree.file_handler = ngx_http_file_cache_manage_file;
    tree.pre_tree_handler = ngx_http_file_cache_manage_directory;
    tree.post_tree_handler = ngx_http_file_cache_noop;
    tree.spec_handler = ngx_http_file_cache_delete_file;
    tree.data = cache;
    tree.alloc = 0;
    tree.log = ngx_cycle->log;

    cache->last = ngx_current_msec;
    cache->files = 0;

    if (ngx_walk_tree(&tree, &cache->path->name) == NGX_ABORT) {
        cache->sh->loading = 0;
        return;
    }

    cache->sh->cold = 0;
```

```
    cache->sh->loading = 0;

    ngx_log_error(NGX_LOG_NOTICE, ngx_cycle->log, 0,
                  "http file cache: %V %.3fM, bsize: %uz",
                  &cache->path->name,
                  ((double) cache->sh->size * cache->bsize) / (1024 * 1024),
                  cache->bsize);
}


static ngx_int_t
ngx_http_file_cache_noop(ngx_tree_ctx_t *ctx, ngx_str_t *path)
{
    return NGX_OK;
}


static ngx_int_t
ngx_http_file_cache_manage_file(ngx_tree_ctx_t *ctx, ngx_str_t *path)
{
    ngx_msec_t              elapsed;
    ngx_http_file_cache_t  *cache;

    cache = ctx->data;

    if (ngx_http_file_cache_add_file(ctx, path) != NGX_OK) {
        (void) ngx_http_file_cache_delete_file(ctx, path);
    }

    if (++cache->files >= cache->loader_files) {
        ngx_http_file_cache_loader_sleep(cache);

    } else {
        ngx_time_update();

        elapsed = ngx_abs((ngx_msec_int_t) (ngx_current_msec - cache->last));

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ngx_cycle->log, 0,
                       "http file cache loader time elapsed: %M", elapsed);

        if (elapsed >= cache->loader_threshold) {
            ngx_http_file_cache_loader_sleep(cache);
        }
    }

    return (ngx_quit || ngx_terminate) ? NGX_ABORT : NGX_OK;
}


static ngx_int_t
ngx_http_file_cache_manage_directory(ngx_tree_ctx_t *ctx, ngx_str_t *path)
{
    if (path->len >= 5
        && ngx_strncmp(path->data + path->len - 5, "/temp", 5) == 0)
    {
        return NGX_DECLINED;
    }

    return NGX_OK;
}


static void
ngx_http_file_cache_loader_sleep(ngx_http_file_cache_t *cache)
{
    ngx_msleep(cache->loader_sleep);

    ngx_time_update();

    cache->last = ngx_current_msec;
    cache->files = 0;
}


static ngx_int_t
ngx_http_file_cache_add_file(ngx_tree_ctx_t *ctx, ngx_str_t *name)
```

```
{
    u_char                *p;
    ngx_int_t              n;
    ngx_uint_t             i;
    ngx_http_cache_t       c;
    ngx_http_file_cache_t  *cache;

    if (name->len < 2 * NGX_HTTP_CACHE_KEY_LEN) {
        return NGX_ERROR;
    }

    if (ctx->size < (off_t) sizeof(ngx_http_file_cache_header_t)) {
        ngx_log_error(NGX_LOG_CRIT, ctx->log, 0,
                      "cache file \"%s\" is too small", name->data);
        return NGX_ERROR;
    }

    ngx_memzero(&c, sizeof(ngx_http_cache_t));
    cache = ctx->data;

    c.length = ctx->size;
    c.fs_size = (ctx->fs_size + cache->bsize - 1) / cache->bsize;

    p = &name->data[name->len - 2 * NGX_HTTP_CACHE_KEY_LEN];

    for (i = 0; i < NGX_HTTP_CACHE_KEY_LEN; i++) {
        n = ngx_hextoi(p, 2);

        if (n == NGX_ERROR) {
            return NGX_ERROR;
        }

        p += 2;

        c.key[i] = (u_char) n;
    }

    return ngx_http_file_cache_add(cache, &c);
}


static ngx_int_t
ngx_http_file_cache_add(ngx_http_file_cache_t *cache, ngx_http_cache_t *c)
{
    ngx_http_file_cache_node_t  *fcn;

    ngx_shmtx_lock(&cache->shpool->mutex);

    fcn = ngx_http_file_cache_lookup(cache, c->key);

    if (fcn == NULL) {

        fcn = ngx_slab_calloc_locked(cache->shpool,
                                     sizeof(ngx_http_file_cache_node_t));
        if (fcn == NULL) {
            ngx_shmtx_unlock(&cache->shpool->mutex);
            return NGX_ERROR;
        }

        ngx_memcpy((u_char *) &fcn->node.key, c->key, sizeof(ngx_rbtree_key_t));

        ngx_memcpy(fcn->key, &c->key[sizeof(ngx_rbtree_key_t)],
                   NGX_HTTP_CACHE_KEY_LEN - sizeof(ngx_rbtree_key_t));

        ngx_rbtree_insert(&cache->sh->rbtree, &fcn->node);

        fcn->uses = 1;
        fcn->exists = 1;
        fcn->fs_size = c->fs_size;

        cache->sh->size += c->fs_size;

    } else {
        ngx_queue_remove(&fcn->queue);
    }

```

```
2018      fcn->expire = ngx_time() + cache->inactive;
2019
2020      ngx_queue_insert_head(&cache->sh->queue, &fcn->queue);
2021
2022      ngx_shmtx_unlock(&cache->shpool->mutex);
2023
2024      return NGX_OK;
2025  }
2026
2027
2028  static ngx_int_t
2029  ngx_http_file_cache_delete_file(ngx_tree_ctx_t *ctx, ngx_str_t *path)
2030  {
2031      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
2032                     "http file cache delete: \"%s\"", path->data);
2033
2034      if (ngx_delete_file(path->data) == NGX_FILE_ERROR) {
2035          ngx_log_error(NGX_LOG_CRIT, ctx->log, ngx_errno,
2036                        ngx_delete_file_n " \"%s\" failed", path->data);
2037      }
2038
2039      return NGX_OK;
2040  }
2041
2042
2043  time_t
2044  ngx_http_file_cache_valid(ngx_array_t *cache_valid, ngx_uint_t status)
2045  {
2046      ngx_uint_t               i;
2047      ngx_http_cache_valid_t  *valid;
2048
2049      if (cache_valid == NULL) {
2050          return 0;
2051      }
2052
2053      valid = cache_valid->elts;
2054      for (i = 0; i < cache_valid->nelts; i++) {
2055
2056          if (valid[i].status == 0) {
2057              return valid[i].valid;
2058          }
2059
2060          if (valid[i].status == status) {
2061              return valid[i].valid;
2062          }
2063      }
2064
2065      return 0;
2066  }
2067
2068
2069  char *
2070  ngx_http_file_cache_set_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
2071  {
2072      char  *confp = conf;
2073
2074      off_t                   max_size;
2075      u_char                 *last, *p;
2076      time_t                  inactive;
2077      size_t                  len;
2078      ssize_t                 size;
2079      ngx_str_t               s, name, *value;
2080      ngx_int_t               loader_files;
2081      ngx_msec_t              loader_sleep, loader_threshold;
2082      ngx_uint_t              i, n, use_temp_path;
2083      ngx_array_t            *caches;
2084      ngx_http_file_cache_t  *cache, **ce;
2085
2086      cache = ngx_pcalloc(cf->pool, sizeof(ngx_http_file_cache_t));
2087      if (cache == NULL) {
2088          return NGX_CONF_ERROR;
2089      }
2090
2091      cache->path = ngx_pcalloc(cf->pool, sizeof(ngx_path_t));
2092      if (cache->path == NULL) {
2093          return NGX_CONF_ERROR;
```

```
    }

    use_temp_path = 1;

    inactive = 600;
    loader_files = 100;
    loader_sleep = 50;
    loader_threshold = 200;

    name.len = 0;
    size = 0;
    max_size = NGX_MAX_OFF_T_VALUE;

    value = cf->args->elts;

    cache->path->name = value[1];

    if (cache->path->name.data[cache->path->name.len - 1] == '/') {
        cache->path->name.len--;
    }

    if (ngx_conf_full_name(cf->cycle, &cache->path->name, 0) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    for (i = 2; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "levels=", 7) == 0) {

            p = value[i].data + 7;
            last = value[i].data + value[i].len;

            for (n = 0; n < 3 && p < last; n++) {

                if (*p > '0' && *p < '3') {

                    cache->path->level[n] = *p++ - '0';
                    cache->path->len += cache->path->level[n] + 1;

                    if (p == last) {
                        break;
                    }

                    if (*p++ == ':' && n < 2 && p != last) {
                        continue;
                    }

                    goto invalid_levels;
                }

                goto invalid_levels;
            }

            if (cache->path->len < 10 + 3) {
                continue;
            }

        invalid_levels:

            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid \"levels\" \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        if (ngx_strncmp(value[i].data, "use_temp_path=", 14) == 0) {

            if (ngx_strcmp(&value[i].data[14], "on") == 0) {
                use_temp_path = 1;

            } else if (ngx_strcmp(&value[i].data[14], "off") == 0) {
                use_temp_path = 0;

            } else {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid use_temp_path value \"%V\", "
                                   "it must be \"on\" or \"off\"",
```

```
                                &value[i]);
            return NGX_CONF_ERROR;
        }

        continue;
    }

    if (ngx_strncmp(value[i].data, "keys_zone=", 10) == 0) {

        name.data = value[i].data + 10;

        p = (u_char *) ngx_strchr(name.data, ':');

        if (p) {
            name.len = p - name.data;

            p++;

            s.len = value[i].data + value[i].len - p;
            s.data = p;

            size = ngx_parse_size(&s);
            if (size > 8191) {
                continue;
            }
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid keys zone size \"%V\"", &value[i]);
        return NGX_CONF_ERROR;
    }

    if (ngx_strncmp(value[i].data, "inactive=", 9) == 0) {

        s.len = value[i].len - 9;
        s.data = value[i].data + 9;

        inactive = ngx_parse_time(&s, 1);
        if (inactive == (time_t) NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid inactive value \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        continue;
    }

    if (ngx_strncmp(value[i].data, "max_size=", 9) == 0) {

        s.len = value[i].len - 9;
        s.data = value[i].data + 9;

        max_size = ngx_parse_offset(&s);
        if (max_size < 0) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid max_size value \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        continue;
    }

    if (ngx_strncmp(value[i].data, "loader_files=", 13) == 0) {

        loader_files = ngx_atoi(value[i].data + 13, value[i].len - 13);
        if (loader_files == NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "invalid loader_files value \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        continue;
    }

    if (ngx_strncmp(value[i].data, "loader_sleep=", 13) == 0) {

```

```
            s.len = value[i].len - 13;
            s.data = value[i].data + 13;

            loader_sleep = ngx_parse_time(&s, 0);
            if (loader_sleep == (ngx_msec_t) NGX_ERROR) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid loader_sleep value \"%V\"", &value[i]);
                return NGX_CONF_ERROR;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "loader_threshold=", 17) == 0) {

            s.len = value[i].len - 17;
            s.data = value[i].data + 17;

            loader_threshold = ngx_parse_time(&s, 0);
            if (loader_threshold == (ngx_msec_t) NGX_ERROR) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid loader_threshold value \"%V\"", &value[i]);
                return NGX_CONF_ERROR;
            }

            continue;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid parameter \"%V\"", &value[i]);
        return NGX_CONF_ERROR;
    }

    if (name.len == 0 || size == 0) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"%V\" must have \"keys_zone\" parameter",
                           &cmd->name);
        return NGX_CONF_ERROR;
    }

    cache->path->manager = ngx_http_file_cache_manager;
    cache->path->loader = ngx_http_file_cache_loader;
    cache->path->data = cache;
    cache->path->conf_file = cf->conf_file->file.name.data;
    cache->path->line = cf->conf_file->line;
    cache->loader_files = loader_files;
    cache->loader_sleep = loader_sleep;
    cache->loader_threshold = loader_threshold;

    if (ngx_add_path(cf, &cache->path) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (!use_temp_path) {
        cache->temp_path = ngx_pcalloc(cf->pool, sizeof(ngx_path_t));
        if (cache->temp_path == NULL) {
            return NGX_CONF_ERROR;
        }

        len = cache->path->name.len + sizeof("/temp") - 1;

        p = ngx_pnalloc(cf->pool, len + 1);
        if (p == NULL) {
            return NGX_CONF_ERROR;
        }

        cache->temp_path->name.len = len;
        cache->temp_path->name.data = p;

        p = ngx_cpymem(p, cache->path->name.data, cache->path->name.len);
        ngx_memcpy(p, "/temp", sizeof("/temp"));

        ngx_memcpy(&cache->temp_path->level, &cache->path->level,
                   3 * sizeof(size_t));

        cache->temp_path->len = cache->path->len;
```

```
        cache->temp_path->conf_file = cf->conf_file->file.name.data;
        cache->temp_path->line = cf->conf_file->line;

        if (ngx_add_path(cf, &cache->temp_path) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

    cache->shm_zone = ngx_shared_memory_add(cf, &name, size, cmd->post);
    if (cache->shm_zone == NULL) {
        return NGX_CONF_ERROR;
    }

    if (cache->shm_zone->data) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "duplicate zone \"%V\"", &name);
        return NGX_CONF_ERROR;
    }


    cache->shm_zone->init = ngx_http_file_cache_init;
    cache->shm_zone->data = cache;

    cache->inactive = inactive;
    cache->max_size = max_size;

    caches = (ngx_array_t *) (confp + cmd->offset);

    ce = ngx_array_push(caches);
    if (ce == NULL) {
        return NGX_CONF_ERROR;
    }

    *ce = cache;

    return NGX_CONF_OK;
}


char *
ngx_http_file_cache_valid_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf)
{
    char  *p = conf;

    time_t                   valid;
    ngx_str_t               *value;
    ngx_uint_t               i, n, status;
    ngx_array_t            **a;
    ngx_http_cache_valid_t   *v;
    static ngx_uint_t         statuses[] = { 200, 301, 302 };

    a = (ngx_array_t **) (p + cmd->offset);

    if (*a == NGX_CONF_UNSET_PTR) {
        *a = ngx_array_create(cf->pool, 1, sizeof(ngx_http_cache_valid_t));
        if (*a == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    value = cf->args->elts;
    n = cf->args->nelts - 1;

    valid = ngx_parse_time(&value[n], 1);
    if (valid == (time_t) NGX_ERROR) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid time value \"%V\"", &value[n]);
        return NGX_CONF_ERROR;
    }

    if (n == 1) {

        for (i = 0; i < 3; i++) {
            v = ngx_array_push(*a);
            if (v == NULL) {
```

```
2398                return NGX_CONF_ERROR;
2399            }
2400
2401            v->status = statuses[i];
2402            v->valid = valid;
2403        }
2404
2405        return NGX_CONF_OK;
2406    }
2407
2408    for (i = 1; i < n; i++) {
2409
2410        if (ngx_strcmp(value[i].data, "any") == 0) {
2411
2412            status = 0;
2413
2414        } else {
2415
2416            status = ngx_atoi(value[i].data, value[i].len);
2417            if (status < 100) {
2418                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
2419                                   "invalid status \"%V\"", &value[i]);
2420                return NGX_CONF_ERROR;
2421            }
2422        }
2423
2424        v = ngx_array_push(*a);
2425        if (v == NULL) {
2426            return NGX_CONF_ERROR;
2427        }
2428
2429        v->status = status;
2430        v->valid = valid;
2431    }
2432
2433    return NGX_CONF_OK;
2434 }
```

One Level Up Top Level

src/http/ngx_http_cache.h - nginx-1.7.10

Data types defined

- ngx_http_cache_s
- ngx_http_cache_valid_t
- ngx_http_file_cache_header_t
- ngx_http_file_cache_node_t
- ngx_http_file_cache_s
- ngx_http_file_cache_sh_t

Macros defined

- NGX_HTTP_CACHE_BYPASS
- NGX_HTTP_CACHE_ETAG_LEN
- NGX_HTTP_CACHE_EXPIRED
- NGX_HTTP_CACHE_HIT
- NGX_HTTP_CACHE_KEY_LEN
- NGX_HTTP_CACHE_MISS
- NGX_HTTP_CACHE_REVALIDATED
- NGX_HTTP_CACHE_SCARCE
- NGX_HTTP_CACHE_STALE
- NGX_HTTP_CACHE_UPDATING
- NGX_HTTP_CACHE_VARY_LEN
- NGX_HTTP_CACHE_VERSION
- _NGX_HTTP_CACHE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_CACHE_H_INCLUDED_
9  #define _NGX_HTTP_CACHE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
17  #define NGX_HTTP_CACHE_MISS          1
18  #define NGX_HTTP_CACHE_BYPASS        2
```

```
#define NGX_HTTP_CACHE_EXPIRED       3
#define NGX_HTTP_CACHE_STALE         4
#define NGX_HTTP_CACHE_UPDATING      5
#define NGX_HTTP_CACHE_REVALIDATED   6
#define NGX_HTTP_CACHE_HIT           7
#define NGX_HTTP_CACHE_SCARCE        8

#define NGX_HTTP_CACHE_KEY_LEN       16
#define NGX_HTTP_CACHE_ETAG_LEN      42
#define NGX_HTTP_CACHE_VARY_LEN      42

#define NGX_HTTP_CACHE_VERSION       3


typedef struct {
    ngx_uint_t                       status;
    time_t                           valid;
} ngx_http_cache_valid_t;


typedef struct {
    ngx_rbtree_node_t                node;
    ngx_queue_t                      queue;

    u_char                           key[NGX_HTTP_CACHE_KEY_LEN
                                         - sizeof(ngx_rbtree_key_t)];

    unsigned                         count:20;
    unsigned                         uses:10;
    unsigned                         valid_msec:10;
    unsigned                         error:10;
    unsigned                         exists:1;
    unsigned                         updating:1;
    unsigned                         deleting:1;
                                     /* 11 unused bits */

    ngx_file_uniq_t                  uniq;
    time_t                           expire;
    time_t                           valid_sec;
    size_t                           body_start;
    off_t                            fs_size;
    ngx_msec_t                       lock_time;
} ngx_http_file_cache_node_t;


struct ngx_http_cache_s {
    ngx_file_t                       file;
    ngx_array_t                      keys;
    uint32_t                         crc32;
    u_char                           key[NGX_HTTP_CACHE_KEY_LEN];
    u_char                           main[NGX_HTTP_CACHE_KEY_LEN];

    ngx_file_uniq_t                  uniq;
    time_t                           valid_sec;
    time_t                           last_modified;
    time_t                           date;

    ngx_str_t                        etag;
    ngx_str_t                        vary;
    u_char                           variant[NGX_HTTP_CACHE_KEY_LEN];

    size_t                           header_start;
    size_t                           body_start;
    off_t                            length;
    off_t                            fs_size;

    ngx_uint_t                       min_uses;
    ngx_uint_t                       error;
    ngx_uint_t                       valid_msec;

    ngx_buf_t                       *buf;

    ngx_http_file_cache_t           *file_cache;
    ngx_http_file_cache_node_t      *node;

    ngx_msec_t                       lock_timeout;
```

```
    ngx_msec_t                       lock_age;
    ngx_msec_t                       lock_time;
    ngx_msec_t                       wait_time;

    ngx_event_t                      wait_event;

     unsigned                         lock:1;
     unsigned                         waiting:1;

     unsigned                         updated:1;
     unsigned                         updating:1;
     unsigned                         exists:1;
     unsigned                         temp_file:1;
     unsigned                         reading:1;
     unsigned                         secondary:1;
};


typedef struct {
    ngx_uint_t                       version;
    time_t                           valid_sec;
    time_t                           last_modified;
    time_t                           date;
    uint32_t                         crc32;
    u_short                          valid_msec;
    u_short                          header_start;
    u_short                          body_start;
    u_char                           etag_len;
    u_char                           etag[NGX_HTTP_CACHE_ETAG_LEN];
    u_char                           vary_len;
    u_char                           vary[NGX_HTTP_CACHE_VARY_LEN];
    u_char                           variant[NGX_HTTP_CACHE_KEY_LEN];
} ngx_http_file_cache_header_t;


typedef struct {
    ngx_rbtree_t                     rbtree;
    ngx_rbtree_node_t                sentinel;
    ngx_queue_t                      queue;
    ngx_atomic_t                     cold;
    ngx_atomic_t                     loading;
    off_t                            size;
} ngx_http_file_cache_sh_t;


struct ngx_http_file_cache_s {
    ngx_http_file_cache_sh_t        *sh;
    ngx_slab_pool_t                 *shpool;

    ngx_path_t                      *path;
    ngx_path_t                      *temp_path;

    off_t                            max_size;
    size_t                           bsize;

    time_t                           inactive;

    ngx_uint_t                       files;
    ngx_uint_t                       loader_files;
    ngx_msec_t                       last;
    ngx_msec_t                       loader_sleep;
    ngx_msec_t                       loader_threshold;

    ngx_shm_zone_t                  *shm_zone;
};


ngx_int_t ngx_http_file_cache_new(ngx_http_request_t *r);
ngx_int_t ngx_http_file_cache_create(ngx_http_request_t *r);
void ngx_http_file_cache_create_key(ngx_http_request_t *r);
ngx_int_t ngx_http_file_cache_open(ngx_http_request_t *r);
ngx_int_t ngx_http_file_cache_set_header(ngx_http_request_t *r, u_char *buf);
void ngx_http_file_cache_update(ngx_http_request_t *r, ngx_temp_file_t *tf);
void ngx_http_file_cache_update_header(ngx_http_request_t *r);
ngx_int_t ngx_http_cache_send(ngx_http_request_t *);
void ngx_http_file_cache_free(ngx_http_cache_t *c, ngx_temp_file_t *tf);
```

```
time_t ngx_http_file_cache_valid(ngx_array_t *cache_valid, ngx_uint_t status);

char *ngx_http_file_cache_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
char *ngx_http_file_cache_valid_set_slot(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


extern ngx_str_t  ngx_http_cache_status[];


#endif /* _NGX_HTTP_CACHE_H_INCLUDED_ */
```

One Level Up Top Level

src/event/modules/ngx_epoll_module.c - nginx-1.7.10

Global variables defined

- ep
- epoll_name
- event_list
- nevents
- ngx_aio_ctx
- ngx_epoll_commands
- ngx_epoll_module
- ngx_epoll_module_ctx
- ngx_eventfd
- ngx_eventfd_conn
- ngx_eventfd_event

Data types defined

- aio_context_t
- epoll_data
- epoll_data_t
- epoll_event
- io_event
- ngx_epoll_conf_t

Functions defined

- epoll_create
- epoll_ctl
- epoll_wait
- io_destroy
- io_getevents
- io_setup
- ngx_epoll_add_connection
- ngx_epoll_add_event
- ngx_epoll_aio_init
- ngx_epoll_create_conf

- ngx_epoll_del_connection
- ngx_epoll_del_event
- ngx_epoll_done
- ngx_epoll_eventfd_handler
- ngx_epoll_init
- ngx_epoll_init_conf
- ngx_epoll_process_events

Macros defined

- EPOLLERR
- EPOLLET
- EPOLLHUP
- EPOLLIN
- EPOLLMSG
- EPOLLONESHOT
- EPOLLOUT
- EPOLLPRI
- EPOLLRDBAND
- EPOLLRDHUP
- EPOLLRDNORM
- EPOLLWRBAND
- EPOLLWRNORM
- EPOLL_CTL_ADD
- EPOLL_CTL_DEL
- EPOLL_CTL_MOD
- SYS_eventfd
- SYS_io_destroy
- SYS_io_getevents
- SYS_io_setup

Source code

```

/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


```

```c
#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_event.h>


#if (NGX_TEST_BUILD_EPOLL)

/* epoll declarations */

#define EPOLLIN         0x001
#define EPOLLPRI        0x002
#define EPOLLOUT        0x004
#define EPOLLRDNORM     0x040
#define EPOLLRDBAND     0x080
#define EPOLLWRNORM     0x100
#define EPOLLWRBAND     0x200
#define EPOLLMSG        0x400
#define EPOLLERR        0x008
#define EPOLLHUP        0x010

#define EPOLLRDHUP      0x2000

#define EPOLLET         0x80000000
#define EPOLLONESHOT    0x40000000

#define EPOLL_CTL_ADD   1
#define EPOLL_CTL_DEL   2
#define EPOLL_CTL_MOD   3

typedef union epoll_data {
    void         *ptr;
    int           fd;
    uint32_t      u32;
    uint64_t      u64;
} epoll_data_t;

struct epoll_event {
    uint32_t      events;
    epoll_data_t  data;
};


int epoll_create(int size);

int epoll_create(int size)
{
    return -1;
}


int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);

int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)
{
    return -1;
}


int epoll_wait(int epfd, struct epoll_event *events, int nevents, int timeout);

int epoll_wait(int epfd, struct epoll_event *events, int nevents, int timeout)
{
    return -1;
}

#if (NGX_HAVE_FILE_AIO)

#define SYS_io_setup      245
#define SYS_io_destroy    246
#define SYS_io_getevents  247
#define SYS_eventfd       323

typedef u_int  aio_context_t;

struct io_event {
    uint64_t  data;  /* the data field from the iocb */
```

```
    uint64_t  obj;   /* what iocb this event came from */
    int64_t   res;   /* result code for this event */
    int64_t   res2;  /* secondary result */
};


#endif
#endif


typedef struct {
    ngx_uint_t  events;
    ngx_uint_t  aio_requests;
} ngx_epoll_conf_t;


static ngx_int_t ngx_epoll_init(ngx_cycle_t *cycle, ngx_msec_t timer);
static void ngx_epoll_done(ngx_cycle_t *cycle);
static ngx_int_t ngx_epoll_add_event(ngx_event_t *ev, ngx_int_t event,
    ngx_uint_t flags);
static ngx_int_t ngx_epoll_del_event(ngx_event_t *ev, ngx_int_t event,
    ngx_uint_t flags);
static ngx_int_t ngx_epoll_add_connection(ngx_connection_t *c);
static ngx_int_t ngx_epoll_del_connection(ngx_connection_t *c,
    ngx_uint_t flags);
static ngx_int_t ngx_epoll_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
    ngx_uint_t flags);

#if (NGX_HAVE_FILE_AIO)
static void ngx_epoll_eventfd_handler(ngx_event_t *ev);
#endif

static void *ngx_epoll_create_conf(ngx_cycle_t *cycle);
static char *ngx_epoll_init_conf(ngx_cycle_t *cycle, void *conf);

static int                  ep = -1;
static struct epoll_event  *event_list;
static ngx_uint_t           nevents;

#if (NGX_HAVE_FILE_AIO)

int                         ngx_eventfd = -1;
aio_context_t               ngx_aio_ctx = 0;

static ngx_event_t          ngx_eventfd_event;
static ngx_connection_t     ngx_eventfd_conn;

#endif

static ngx_str_t      epoll_name = ngx_string("epoll");

static ngx_command_t  ngx_epoll_commands[] = {

    { ngx_string("epoll_events"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      0,
      offsetof(ngx_epoll_conf_t, events),
      NULL },

    { ngx_string("worker_aio_requests"),
      NGX_EVENT_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      0,
      offsetof(ngx_epoll_conf_t, aio_requests),
      NULL },

      ngx_null_command
};


ngx_event_module_t  ngx_epoll_module_ctx = {
    &epoll_name,
    ngx_epoll_create_conf,               /* create configuration */
    ngx_epoll_init_conf,                 /* init configuration */

```

```
    {
        ngx_epoll_add_event,             /* add an event */
        ngx_epoll_del_event,             /* delete an event */
        ngx_epoll_add_event,             /* enable an event */
        ngx_epoll_del_event,             /* disable an event */
        ngx_epoll_add_connection,        /* add an connection */
        ngx_epoll_del_connection,        /* delete an connection */
        NULL,                            /* process the changes */
        ngx_epoll_process_events,        /* process the events */
        ngx_epoll_init,                  /* init the events */
        ngx_epoll_done,                  /* done the events */
    }
};

ngx_module_t  ngx_epoll_module = {
    NGX_MODULE_V1,
    &ngx_epoll_module_ctx,               /* module context */
    ngx_epoll_commands,                  /* module directives */
    NGX_EVENT_MODULE,                    /* module type */
    NULL,                                /* init master */
    NULL,                                /* init module */
    NULL,                                /* init process */
    NULL,                                /* init thread */
    NULL,                                /* exit thread */
    NULL,                                /* exit process */
    NULL,                                /* exit master */
    NGX_MODULE_V1_PADDING
};


#if (NGX_HAVE_FILE_AIO)

/*
* We call io_setup(), io_destroy() io_submit(), and io_getevents() directly
* as syscalls instead of libaio usage, because the library header file
* supports eventfd() since 0.3.107 version only.
*/

static int
io_setup(u_int nr_reqs, aio_context_t *ctx)
{
    return syscall(SYS_io_setup, nr_reqs, ctx);
}


static int
io_destroy(aio_context_t ctx)
{
    return syscall(SYS_io_destroy, ctx);
}


static int
io_getevents(aio_context_t ctx, long min_nr, long nr, struct io_event *events,
    struct timespec *tmo)
{
    return syscall(SYS_io_getevents, ctx, min_nr, nr, events, tmo);
}


static void
ngx_epoll_aio_init(ngx_cycle_t *cycle, ngx_epoll_conf_t *epcf)
{
    int                 n;
    struct epoll_event  ee;

#if (NGX_HAVE_SYS_EVENTFD_H)
    ngx_eventfd = eventfd(0, 0);
#else
    ngx_eventfd = syscall(SYS_eventfd, 0);
#endif

    if (ngx_eventfd == -1) {
        ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                      "eventfd() failed");
        ngx_file_aio = 0;
```

```
        return;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "eventfd: %d", ngx_eventfd);

    n = 1;

    if (ioctl(ngx_eventfd, FIONBIO, &n) == -1) {
        ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                      "ioctl(eventfd, FIONBIO) failed");
        goto failed;
    }

    if (io_setup(epcf->aio_requests, &ngx_aio_ctx) == -1) {
        ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                      "io_setup() failed");
        goto failed;
    }

    ngx_eventfd_event.data = &ngx_eventfd_conn;
    ngx_eventfd_event.handler = ngx_epoll_eventfd_handler;
    ngx_eventfd_event.log = cycle->log;
    ngx_eventfd_event.active = 1;
    ngx_eventfd_conn.fd = ngx_eventfd;
    ngx_eventfd_conn.read = &ngx_eventfd_event;
    ngx_eventfd_conn.log = cycle->log;

    ee.events = EPOLLIN|EPOLLET;
    ee.data.ptr = &ngx_eventfd_conn;

    if (epoll_ctl(ep, EPOLL_CTL_ADD, ngx_eventfd, &ee) != -1) {
        return;
    }

    ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                  "epoll_ctl(EPOLL_CTL_ADD, eventfd) failed");

    if (io_destroy(ngx_aio_ctx) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "io_destroy() failed");
    }

failed:

    if (close(ngx_eventfd) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "eventfd close() failed");
    }

    ngx_eventfd = -1;
    ngx_aio_ctx = 0;
    ngx_file_aio = 0;
}

#endif


static ngx_int_t
ngx_epoll_init(ngx_cycle_t *cycle, ngx_msec_t timer)
{
    ngx_epoll_conf_t  *epcf;

    epcf = ngx_event_get_conf(cycle->conf_ctx, ngx_epoll_module);

    if (ep == -1) {
        ep = epoll_create(cycle->connection_n / 2);

        if (ep == -1) {
            ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
                          "epoll_create() failed");
            return NGX_ERROR;
        }

#if (NGX_HAVE_FILE_AIO)

```

```
        ngx_epoll_aio_init(cycle, epcf);

#endif
    }

    if (nevents < epcf->events) {
        if (event_list) {
            ngx_free(event_list);
        }

        event_list = ngx_alloc(sizeof(struct epoll_event) * epcf->events,
                               cycle->log);
        if (event_list == NULL) {
            return NGX_ERROR;
        }
    }

    nevents = epcf->events;

    ngx_io = ngx_os_io;

    ngx_event_actions = ngx_epoll_module_ctx.actions;

#if (NGX_HAVE_CLEAR_EVENT)
    ngx_event_flags = NGX_USE_CLEAR_EVENT
#else
    ngx_event_flags = NGX_USE_LEVEL_EVENT
#endif
                      |NGX_USE_GREEDY_EVENT
                      |NGX_USE_EPOLL_EVENT;

    return NGX_OK;
}


static void
ngx_epoll_done(ngx_cycle_t *cycle)
{
    if (close(ep) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "epoll close() failed");
    }

    ep = -1;

#if (NGX_HAVE_FILE_AIO)

    if (ngx_eventfd != -1) {

        if (io_destroy(ngx_aio_ctx) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "io_destroy() failed");
        }

        if (close(ngx_eventfd) == -1) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "eventfd close() failed");
        }

        ngx_eventfd = -1;
    }

    ngx_aio_ctx = 0;

#endif

    ngx_free(event_list);

    event_list = NULL;
    nevents = 0;
}


static ngx_int_t
ngx_epoll_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
```

```
388     int                 op;
389     uint32_t            events, prev;
390     ngx_event_t         *e;
391     ngx_connection_t    *c;
392     struct epoll_event   ee;
393
394     c = ev->data;
395
396     events = (uint32_t) event;
397
398     if (event == NGX_READ_EVENT) {
399         e = c->write;
400         prev = EPOLLOUT;
401 #if (NGX_READ_EVENT != EPOLLIN|EPOLLRDHUP)
402         events = EPOLLIN|EPOLLRDHUP;
403 #endif
404
405     } else {
406         e = c->read;
407         prev = EPOLLIN|EPOLLRDHUP;
408 #if (NGX_WRITE_EVENT != EPOLLOUT)
409         events = EPOLLOUT;
410 #endif
411     }
412
413     if (e->active) {
414         op = EPOLL_CTL_MOD;
415         events |= prev;
416
417     } else {
418         op = EPOLL_CTL_ADD;
419     }
420
421     ee.events = events | (uint32_t) flags;
422     ee.data.ptr = (void *) ((uintptr_t) c | ev->instance);
423
424     ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ev->log, 0,
425                    "epoll add event: fd:%d op:%d ev:%08XD",
426                    c->fd, op, ee.events);
427
428     if (epoll_ctl(ep, op, c->fd, &ee) == -1) {
429         ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
430                       "epoll_ctl(%d, %d) failed", op, c->fd);
431         return NGX_ERROR;
432     }
433
434     ev->active = 1;
435 #if 0
436     ev->oneshot = (flags & NGX_ONESHOT_EVENT) ? 1 : 0;
437 #endif
438
439     return NGX_OK;
440 }
441
442
443 static ngx_int_t
444 ngx_epoll_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
445 {
446     int                 op;
447     uint32_t            prev;
448     ngx_event_t         *e;
449     ngx_connection_t    *c;
450     struct epoll_event   ee;
451
452     /*
453      * when the file descriptor is closed, the epoll automatically deletes
454      * it from its queue, so we do not need to delete explicitly the event
455      * before the closing the file descriptor
456      */
457
458     if (flags & NGX_CLOSE_EVENT) {
459         ev->active = 0;
460         return NGX_OK;
461     }
462
463     c = ev->data;
```

```
464
465     if (event == NGX_READ_EVENT) {
466         e = c->write;
467         prev = EPOLLOUT;
468
469     } else {
470         e = c->read;
471         prev = EPOLLIN|EPOLLRDHUP;
472     }
473
474     if (e->active) {
475         op = EPOLL_CTL_MOD;
476         ee.events = prev | (uint32_t) flags;
477         ee.data.ptr = (void *) ((uintptr_t) c | ev->instance);
478
479     } else {
480         op = EPOLL_CTL_DEL;
481         ee.events = 0;
482         ee.data.ptr = NULL;
483     }
484
485     ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ev->log, 0,
486                    "epoll del event: fd:%d op:%d ev:%08XD",
487                    c->fd, op, ee.events);
488
489     if (epoll_ctl(ep, op, c->fd, &ee) == -1) {
490         ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
491                       "epoll_ctl(%d, %d) failed", op, c->fd);
492         return NGX_ERROR;
493     }
494
495     ev->active = 0;
496
497     return NGX_OK;
498 }
499
500
501 static ngx_int_t
502 ngx_epoll_add_connection(ngx_connection_t *c)
503 {
504     struct epoll_event  ee;
505
506     ee.events = EPOLLIN|EPOLLOUT|EPOLLET|EPOLLRDHUP;
507     ee.data.ptr = (void *) ((uintptr_t) c | c->read->instance);
508
509     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
510                    "epoll add connection: fd:%d ev:%08XD", c->fd, ee.events);
511
512     if (epoll_ctl(ep, EPOLL_CTL_ADD, c->fd, &ee) == -1) {
513         ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
514                       "epoll_ctl(EPOLL_CTL_ADD, %d) failed", c->fd);
515         return NGX_ERROR;
516     }
517
518     c->read->active = 1;
519     c->write->active = 1;
520
521     return NGX_OK;
522 }
523
524
525 static ngx_int_t
526 ngx_epoll_del_connection(ngx_connection_t *c, ngx_uint_t flags)
527 {
528     int                 op;
529     struct epoll_event  ee;
530
531     /*
532      * when the file descriptor is closed the epoll automatically deletes
533      * it from its queue so we do not need to delete explicitly the event
534      * before the closing the file descriptor
535      */
536
537     if (flags & NGX_CLOSE_EVENT) {
538         c->read->active = 0;
539         c->write->active = 0;
```

```
        return NGX_OK;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "epoll del connection: fd:%d", c->fd);

    op = EPOLL_CTL_DEL;
    ee.events = 0;
    ee.data.ptr = NULL;

    if (epoll_ctl(ep, op, c->fd, &ee) == -1) {
        ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
                      "epoll_ctl(%d, %d) failed", op, c->fd);
        return NGX_ERROR;
    }

    c->read->active = 0;
    c->write->active = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_epoll_process_events(ngx_cycle_t *cycle, ngx_msec_t timer, ngx_uint_t flags)
{
    int                events;
    uint32_t           revents;
    ngx_int_t          instance, i;
    ngx_uint_t         level;
    ngx_err_t          err;
    ngx_event_t       *rev, *wev;
    ngx_queue_t       *queue;
    ngx_connection_t  *c;

    /* NGX_TIMER_INFINITE == INFTIM */

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "epoll timer: %M", timer);

    events = epoll_wait(ep, event_list, (int) nevents, timer);

    err = (events == -1) ? ngx_errno : 0;

    if (flags & NGX_UPDATE_TIME || ngx_event_timer_alarm) {
        ngx_time_update();
    }

    if (err) {
        if (err == NGX_EINTR) {

            if (ngx_event_timer_alarm) {
                ngx_event_timer_alarm = 0;
                return NGX_OK;
            }

            level = NGX_LOG_INFO;

        } else {
            level = NGX_LOG_ALERT;
        }

        ngx_log_error(level, cycle->log, err, "epoll_wait() failed");
        return NGX_ERROR;
    }

    if (events == 0) {
        if (timer != NGX_TIMER_INFINITE) {
            return NGX_OK;
        }

        ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                      "epoll_wait() returned no events without timeout");
        return NGX_ERROR;
    }

```

```
616     for (i = 0; i < events; i++) {
617         c = event_list[i].data.ptr;
618
619         instance = (uintptr_t) c & 1;
620         c = (ngx_connection_t *) ((uintptr_t) c & (uintptr_t) ~1);
621
622         rev = c->read;
623
624         if (c->fd == -1 || rev->instance != instance) {
625
626             /*
627              * the stale event from a file descriptor
628              * that was just closed in this iteration
629              */
630
631             ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
632                            "epoll: stale event %p", c);
633             continue;
634         }
635
636         revents = event_list[i].events;
637
638         ngx_log_debug3(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
639                        "epoll: fd:%d ev:%04XD d:%p",
640                        c->fd, revents, event_list[i].data.ptr);
641
642         if (revents & (EPOLLERR|EPOLLHUP)) {
643             ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
644                            "epoll_wait() error on fd:%d ev:%04XD",
645                            c->fd, revents);
646         }
647
648 #if 0
649         if (revents & ~(EPOLLIN|EPOLLOUT|EPOLLERR|EPOLLHUP)) {
650             ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
651                           "strange epoll_wait() events fd:%d ev:%04XD",
652                           c->fd, revents);
653         }
654 #endif
655
656         if ((revents & (EPOLLERR|EPOLLHUP))
657              && (revents & (EPOLLIN|EPOLLOUT)) == 0)
658         {
659             /*
660              * if the error events were returned without EPOLLIN or EPOLLOUT,
661              * then add these flags to handle the events at least in one
662              * active handler
663              */
664
665             revents |= EPOLLIN|EPOLLOUT;
666         }
667
668         if ((revents & EPOLLIN) && rev->active) {
669
670 #if (NGX_HAVE_EPOLLRDHUP)
671             if (revents & EPOLLRDHUP) {
672                 rev->pending_eof = 1;
673             }
674 #endif
675
676             rev->ready = 1;
677
678             if (flags & NGX_POST_EVENTS) {
679                 queue = rev->accept ? &ngx_posted_accept_events
680                                     : &ngx_posted_events;
681
682                 ngx_post_event(rev, queue);
683
684             } else {
685                 rev->handler(rev);
686             }
687         }
688
689         wev = c->write;
690
691         if ((revents & EPOLLOUT) && wev->active) {
```

```
692
693             if (c->fd == -1 || wev->instance != instance) {
694
695                 /*
696                  * the stale event from a file descriptor
697                  * that was just closed in this iteration
698                  */
699
700                 ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
701                                "epoll: stale event %p", c);
702                 continue;
703             }
704
705             wev->ready = 1;
706
707             if (flags & NGX_POST_EVENTS) {
708                 ngx_post_event(wev, &ngx_posted_events);
709
710             } else {
711                 wev->handler(wev);
712             }
713         }
714     }
715
716     return NGX_OK;
717 }
718
719
720 #if (NGX_HAVE_FILE_AIO)
721
722 static void
723 ngx_epoll_eventfd_handler(ngx_event_t *ev)
724 {
725     int               n, events;
726     long              i;
727     uint64_t          ready;
728     ngx_err_t         err;
729     ngx_event_t      *e;
730     ngx_event_aio_t  *aio;
731     struct io_event   event[64];
732     struct timespec   ts;
733
734     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ev->log, 0, "eventfd handler");
735
736     n = read(ngx_eventfd, &ready, 8);
737
738     err = ngx_errno;
739
740     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ev->log, 0, "eventfd: %d", n);
741
742     if (n != 8) {
743         if (n == -1) {
744             if (err == NGX_EAGAIN) {
745                 return;
746             }
747
748             ngx_log_error(NGX_LOG_ALERT, ev->log, err, "read(eventfd) failed");
749             return;
750         }
751
752         ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
753                       "read(eventfd) returned only %d bytes", n);
754         return;
755     }
756
757     ts.tv_sec = 0;
758     ts.tv_nsec = 0;
759
760     while (ready) {
761
762         events = io_getevents(ngx_aio_ctx, 1, 64, event, &ts);
763
764         ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ev->log, 0,
765                        "io_getevents: %l", events);
766
767         if (events > 0) {
```

```
768             ready -= events;
769
770             for (i = 0; i < events; i++) {
771
772                 ngx_log_debug4(NGX_LOG_DEBUG_EVENT, ev->log, 0,
773                                "io_event: %uXL %uXL %L %L",
774                                 event[i].data, event[i].obj,
775                                 event[i].res, event[i].res2);
776
777                 e = (ngx_event_t *) (uintptr_t) event[i].data;
778
779                 e->complete = 1;
780                 e->active = 0;
781                 e->ready = 1;
782
783                 aio = e->data;
784                 aio->res = event[i].res;
785
786                 ngx_post_event(e, &ngx_posted_events);
787             }
788
789             continue;
790         }
791
792         if (events == 0) {
793             return;
794         }
795
796         /* events == -1 */
797         ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
798                       "io_getevents() failed");
799         return;
800     }
801 }
802
803 #endif
804
805
806 static void *
807 ngx_epoll_create_conf(ngx_cycle_t *cycle)
808 {
809     ngx_epoll_conf_t  *epcf;
810
811     epcf = ngx_palloc(cycle->pool, sizeof(ngx_epoll_conf_t));
812     if (epcf == NULL) {
813         return NULL;
814     }
815
816     epcf->events = NGX_CONF_UNSET;
817     epcf->aio_requests = NGX_CONF_UNSET;
818
819     return epcf;
820 }
821
822
823 static char *
824 ngx_epoll_init_conf(ngx_cycle_t *cycle, void *conf)
825 {
826     ngx_epoll_conf_t *epcf = conf;
827
828     ngx_conf_init_uint_value(epcf->events, 512);
829     ngx_conf_init_uint_value(epcf->aio_requests, 32);
830
831     return NGX_CONF_OK;
832 }
```

One Level Up Top Level

src/os/unix/ngx_linux_aio_read.c - nginx-1.7.10

Functions defined

- io_submit
- ngx_file_aio_event_handler
- ngx_file_aio_read

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  extern int             ngx_eventfd;
14  extern aio_context_t   ngx_aio_ctx;
15
16
17  static void ngx_file_aio_event_handler(ngx_event_t *ev);
18
19
20  static int
21  io_submit(aio_context_t ctx, long n, struct iocb **paiocb)
22  {
23      return syscall(SYS_io_submit, ctx, n, paiocb);
24  }
25
26
27  ssize_t
28  ngx_file_aio_read(ngx_file_t *file, u_char *buf, size_t size, off_t offset,
29      ngx_pool_t *pool)
30  {
31      ngx_err_t         err;
32      struct iocb      *piocb[1];
33      ngx_event_t      *ev;
34      ngx_event_aio_t  *aio;
35
36      if (!ngx_file_aio) {
37          return ngx_read_file(file, buf, size, offset);
38      }
39
40      aio = file->aio;
41
42      if (aio == NULL) {
43          aio = ngx_pcalloc(pool, sizeof(ngx_event_aio_t));
44          if (aio == NULL) {
45              return NGX_ERROR;
46          }
47
48          aio->file = file;
49          aio->fd = file->fd;
50          aio->event.data = aio;
51          aio->event.ready = 1;
52          aio->event.log = file->log;
53          file->aio = aio;
54      }
55
56      ev = &aio->event;
57
58      if (!ev->ready) {
```

```
        ngx_log_error(NGX_LOG_ALERT, file->log, 0,
                      "second aio post for \"%V\"", &file->name);
        return NGX_AGAIN;
    }

    ngx_log_debug4(NGX_LOG_DEBUG_CORE, file->log, 0,
                   "aio complete:%d @%O:%z %V",
                   ev->complete, offset, size, &file->name);

    if (ev->complete) {
        ev->active = 0;
        ev->complete = 0;

        if (aio->res >= 0) {
            ngx_set_errno(0);
            return aio->res;
        }

        ngx_set_errno(-aio->res);

        ngx_log_error(NGX_LOG_CRIT, file->log, ngx_errno,
                      "aio read \"%s\" failed", file->name.data);

        return NGX_ERROR;
    }

    ngx_memzero(&aio->aiocb, sizeof(struct iocb));

    aio->aiocb.aio_data = (uint64_t) (uintptr_t) ev;
    aio->aiocb.aio_lio_opcode = IOCB_CMD_PREAD;
    aio->aiocb.aio_fildes = file->fd;
    aio->aiocb.aio_buf = (uint64_t) (uintptr_t) buf;
    aio->aiocb.aio_nbytes = size;
    aio->aiocb.aio_offset = offset;
    aio->aiocb.aio_flags = IOCB_FLAG_RESFD;
    aio->aiocb.aio_resfd = ngx_eventfd;

    ev->handler = ngx_file_aio_event_handler;

    piocb[0] = &aio->aiocb;

     if (io_submit(ngx_aio_ctx, 1, piocb) == 1) {
         ev->active = 1;
         ev->ready = 0;
         ev->complete = 0;

         return NGX_AGAIN;
     }

     err = ngx_errno;

     if (err == NGX_EAGAIN) {
         return ngx_read_file(file, buf, size, offset);
     }

     ngx_log_error(NGX_LOG_CRIT, file->log, err,
                   "io_submit(\"%V\") failed", &file->name);

     if (err == NGX_ENOSYS) {
         ngx_file_aio = 0;
         return ngx_read_file(file, buf, size, offset);
     }

     return NGX_ERROR;
}


static void
ngx_file_aio_event_handler(ngx_event_t *ev)
{
    ngx_event_aio_t  *aio;

    aio = ev->data;

    ngx_log_debug2(NGX_LOG_DEBUG_CORE, ev->log, 0,
                   "aio event handler fd:%d %V", aio->fd, &aio->file->name);
```

```
135
136     aio->handler(ev);
137 }
```

One Level Up Top Level

src/os/unix/ngx_freebsd_config.h - nginx-1.7.10

Data types defined

- ngx_aiocb_t

Macros defined

- CMSG_DATA
- CMSG_DATA
- CMSG_LEN
- CMSG_LEN
- CMSG_SPACE
- CMSG_SPACE
- IOV_MAX
- NGX_HAVE_DEBUG_MALLOC
- NGX_HAVE_INHERITED_NONBLOCK
- NGX_HAVE_OS_SPECIFIC_INIT
- NGX_KEEPALIVE_FACTOR
- NGX_LISTEN_BACKLOG
- _NGX_FREEBSD_CONFIG_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_FREEBSD_CONFIG_H_INCLUDED_
9  #define _NGX_FREEBSD_CONFIG_H_INCLUDED_
10
11
12  #include <sys/types.h>
13  #include <sys/time.h>
14  #include <unistd.h>
15  #include <stdarg.h>
16  #include <stddef.h>             /* offsetof() */
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <ctype.h>
20  #include <errno.h>
21  #include <string.h>
22  #include <signal.h>
23  #include <pwd.h>
24  #include <grp.h>
25  #include <dirent.h>
26  #include <glob.h>
27  #include <time.h>
28  #include <sys/param.h>          /* ALIGN() */
29  #include <sys/mount.h>          /* statfs() */
```

```
30
31  #include <sys/filio.h>          /* FIONBIO */
32  #include <sys/uio.h>
33  #include <sys/stat.h>
34  #include <fcntl.h>
35
36  #include <sys/wait.h>
37  #include <sys/mman.h>
38  #include <sys/resource.h>
39  #include <sched.h>
40
41  #include <sys/socket.h>
42  #include <netinet/in.h>
43  #include <netinet/tcp.h>        /* TCP_NODELAY, TCP_NOPUSH */
44  #include <arpa/inet.h>
45  #include <netdb.h>
46  #include <sys/un.h>
47
48  #include <libutil.h>            /* setproctitle() before 4.1 */
49  #include <osreldate.h>
50  #include <sys/sysctl.h>
51
52
53  #if __FreeBSD_version < 400017
54
55  /*
56  * FreeBSD 3.x has no CMSG_SPACE() and CMSG_LEN() and has the broken CMSG_DATA()
57  */
58
59  #undef  CMSG_SPACE
60  #define CMSG_SPACE(l)       (ALIGN(sizeof(struct cmsghdr)) + ALIGN(l))
61
62  #undef  CMSG_LEN
63  #define CMSG_LEN(l)         (ALIGN(sizeof(struct cmsghdr)) + (l))
64
65  #undef  CMSG_DATA
66  #define CMSG_DATA(cmsg)     ((u_char *)(cmsg) + ALIGN(sizeof(struct cmsghdr)))
67
68  #endif
69
70
71  #include <ngx_auto_config.h>
72
73
74  #if (NGX_HAVE_POSIX_SEM)
75  #include <semaphore.h>
76  #endif
77
78
79  #if (NGX_HAVE_POLL)
80  #include <poll.h>
81  #endif
82
83
84  #if (NGX_HAVE_KQUEUE)
85  #include <sys/event.h>
86  #endif
87
88
89  #if (NGX_HAVE_FILE_AIO || NGX_HAVE_AIO)
90  #include <aio.h>
91  typedef struct aiocb  ngx_aiocb_t;
92  #endif
93
94
95  #define NGX_LISTEN_BACKLOG        -1
96
97
98  #ifdef __DragonFly__
99  #define NGX_KEEPALIVE_FACTOR      1000
100  #endif
101
102
103  #if (__FreeBSD_version < 430000 || __FreeBSD_version < 500012)
104
105  pid_t rfork_thread(int flags, void *stack, int (*func)(void *arg), void *arg);
```

```
106
107 #endif
108
109 #ifndef IOV_MAX
110 #define IOV_MAX   1024
111 #endif
112
113
114 #ifndef NGX_HAVE_INHERITED_NONBLOCK
115 #define NGX_HAVE_INHERITED_NONBLOCK  1
116 #endif
117
118
119 #define NGX_HAVE_OS_SPECIFIC_INIT    1
120 #define NGX_HAVE_DEBUG_MALLOC        1
121
122
123 extern char **environ;
124 extern char  *malloc_options;
125
126
127 #endif /* _NGX_FREEBSD_CONFIG_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_readv_chain.c - nginx-1.7.10

Functions defined

- ngx_readv_chain

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ssize_t
14  ngx_readv_chain(ngx_connection_t *c, ngx_chain_t *chain, off_t limit)
15  {
16      u_char        *prev;
17      ssize_t        n, size;
18      ngx_err_t      err;
19      ngx_array_t    vec;
20      ngx_event_t   *rev;
21      struct iovec  *iov, iovs[NGX_IOVS_PREALLOCATE];
22
23      rev = c->read;
24
25  #if (NGX_HAVE_KQUEUE)
26
27      if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
28          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
29                         "readv: eof:%d, avail:%d, err:%d",
30                         rev->pending_eof, rev->available, rev->kq_errno);
31
32          if (rev->available == 0) {
33              if (rev->pending_eof) {
34                  rev->ready = 0;
35                  rev->eof = 1;
36
37                  ngx_log_error(NGX_LOG_INFO, c->log, rev->kq_errno,
38                                "kevent() reported about an closed connection");
39
40                  if (rev->kq_errno) {
41                      rev->error = 1;
42                      ngx_set_socket_errno(rev->kq_errno);
43                      return NGX_ERROR;
44                  }
45
46                  return 0;
47
48              } else {
49                  return NGX_AGAIN;
50              }
51          }
52      }
53
54  #endif
55
56      prev = NULL;
57      iov = NULL;
58      size = 0;
59
60      vec.elts = iovs;
61      vec.nelts = 0;
62      vec.size = sizeof(struct iovec);
```

```
    vec.nalloc = NGX_IOVS_PREALLOCATE;
    vec.pool = c->pool;

    /* coalesce the neighbouring bufs */

    while (chain) {
        n = chain->buf->end - chain->buf->last;

        if (limit) {
            if (size >= limit) {
                break;
            }

            if (size + n > limit) {
                n = (ssize_t) (limit - size);
            }
        }

        if (prev == chain->buf->last) {
            iov->iov_len += n;

        } else {
            if (vec.nelts >= IOV_MAX) {
                break;
            }

            iov = ngx_array_push(&vec);
            if (iov == NULL) {
                return NGX_ERROR;
            }

            iov->iov_base = (void *) chain->buf->last;
            iov->iov_len = n;
        }

        size += n;
        prev = chain->buf->end;
         chain = chain->next;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
                   "readv: %d, last:%d", vec.nelts, iov->iov_len);

    do {
        n = readv(c->fd, (struct iovec *) vec.elts, vec.nelts);

        if (n >= 0) {

#if (NGX_HAVE_KQUEUE)

            if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
                rev->available -= n;

                /*
                 * rev->available may be negative here because some additional
                 * bytes may be received between kevent() and recv()
                 */

                if (rev->available <= 0) {
                    if (!rev->pending_eof) {
                        rev->ready = 0;
                    }

                    if (rev->available < 0) {
                        rev->available = 0;
                    }
                }

                if (n == 0) {

                    /*
                     * on FreeBSD recv() may return 0 on closed socket
                     * even if kqueue reported about available data
                     */

#if 0
```

```
                    ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                                  "readv() returned 0 while kevent() reported "
                                  "%d available bytes", rev->available);
#endif

                    rev->ready = 0;
                    rev->eof = 1;
                    rev->available = 0;
                }

                return n;
            }

#endif /* NGX_HAVE_KQUEUE */

            if (n < size && !(ngx_event_flags & NGX_USE_GREEDY_EVENT)) {
                rev->ready = 0;
            }

            if (n == 0) {
                rev->eof = 1;
            }

            return n;
        }

        err = ngx_socket_errno;

        if (err == NGX_EAGAIN || err == NGX_EINTR) {
            ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
                           "readv() not ready");
            n = NGX_AGAIN;

        } else {
            n = ngx_connection_error(c, err, "readv() failed");
            break;
        }

    } while (err == NGX_EINTR);

    rev->ready = 0;

    if (n == NGX_ERROR) {
        c->read->error = 1;
    }

    return n;
}
```

One Level Up Top Level

src/os/unix/ngx_udp_recv.c - nginx-1.7.10

Functions defined

- ngx_udp_unix_recv
- ngx_udp_unix_recv

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  #if (NGX_HAVE_KQUEUE)
14
15  ssize_t
16  ngx_udp_unix_recv(ngx_connection_t *c, u_char *buf, size_t size)
17  {
18      ssize_t       n;
19      ngx_err_t     err;
20      ngx_event_t  *rev;
21
22      rev = c->read;
23
24      do {
25          n = recv(c->fd, buf, size, 0);
26
27          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
28                         "recv: fd:%d %d of %d", c->fd, n, size);
29
30          if (n >= 0) {
31              if (ngx_event_flags & NGX_USE_KQUEUE_EVENT) {
32                  rev->available -= n;
33
34                  /*
35                   * rev->available may be negative here because some additional
36                   * bytes may be received between kevent() and recv()
37                   */
38
39                  if (rev->available <= 0) {
40                      rev->ready = 0;
41                      rev->available = 0;
42                  }
43              }
44
45              return n;
46          }
47
48          err = ngx_socket_errno;
49
50          if (err == NGX_EAGAIN || err == NGX_EINTR) {
51              ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
52                             "recv() not ready");
53              n = NGX_AGAIN;
54
55          } else {
56              n = ngx_connection_error(c, err, "recv() failed");
57              break;
58          }
59
60      } while (err == NGX_EINTR);
```

```
61
62      rev->ready = 0;
63
64      if (n == NGX_ERROR) {
65          rev->error = 1;
66      }
67
68      return n;
69  }
70
71  #else /* ! NGX_HAVE_KQUEUE */
72
73  ssize_t
74  ngx_udp_unix_recv(ngx_connection_t *c, u_char *buf, size_t size)
75  {
76      ssize_t       n;
77      ngx_err_t     err;
78      ngx_event_t  *rev;
79
80      rev = c->read;
81
82      do {
83          n = recv(c->fd, buf, size, 0);
84
85          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
86                         "recv: fd:%d %d of %d", c->fd, n, size);
87
88          if (n >= 0) {
89              return n;
90          }
91
92          err = ngx_socket_errno;
93
94          if (err == NGX_EAGAIN || err == NGX_EINTR) {
95              ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
96                             "recv() not ready");
97              n = NGX_AGAIN;
98
99          } else {
100             n = ngx_connection_error(c, err, "recv() failed");
101             break;
102         }
103
104     } while (err == NGX_EINTR);
105
106     rev->ready = 0;
107
108     if (n == NGX_ERROR) {
109         rev->error = 1;
110     }
111
112     return n;
113 }
114
115 #endif /* NGX_HAVE_KQUEUE */
```

One Level Up Top Level

src/os/unix/ngx_send.c - nginx-1.7.10

Functions defined

- ngx_unix_send

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ssize_t
14  ngx_unix_send(ngx_connection_t *c, u_char *buf, size_t size)
15  {
16      ssize_t       n;
17      ngx_err_t     err;
18      ngx_event_t  *wev;
19
20      wev = c->write;
21
22  #if (NGX_HAVE_KQUEUE)
23
24      if ((ngx_event_flags & NGX_USE_KQUEUE_EVENT) && wev->pending_eof) {
25          (void) ngx_connection_error(c, wev->kq_errno,
26                                 "kevent() reported about an closed connection");
27          wev->error = 1;
28          return NGX_ERROR;
29      }
30
31  #endif
32
33      for ( ;; ) {
34          n = send(c->fd, buf, size, 0);
35
36          ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
37                         "send: fd:%d %d of %d", c->fd, n, size);
38
39          if (n > 0) {
40              if (n < (ssize_t) size) {
41                  wev->ready = 0;
42              }
43
44              c->sent += n;
45
46              return n;
47          }
48
49          err = ngx_socket_errno;
50
51          if (n == 0) {
52              ngx_log_error(NGX_LOG_ALERT, c->log, err, "send() returned zero");
53              wev->ready = 0;
54              return n;
55          }
56
57          if (err == NGX_EAGAIN || err == NGX_EINTR) {
58              wev->ready = 0;
59
60              ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
61                             "send() not ready");
62
```

```
63              if (err == NGX_EAGAIN) {
64                  return NGX_AGAIN;
65              }
66
67          } else {
68              wev->error = 1;
69              (void) ngx_connection_error(c, err, "send() failed");
70              return NGX_ERROR;
71          }
72      }
73  }
```

One Level Up Top Level

src/os/unix/ngx_freebsd_sendfile_chain.c - nginx-1.7.10

Functions defined

- ngx_freebsd_sendfile_chain

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  /*
14  * Although FreeBSD sendfile() allows to pass a header and a trailer,
15  * it cannot send a header with a part of the file in one packet until
16  * FreeBSD 5.3.  Besides, over the fast ethernet connection sendfile()
17  * may send the partially filled packets, i.e. the 8 file pages may be sent
18  * as the 11 full 1460-bytes packets, then one incomplete 324-bytes packet,
19  * and then again the 11 full 1460-bytes packets.
20  *
21  * Therefore we use the TCP_NOPUSH option (similar to Linux's TCP_CORK)
22  * to postpone the sending - it not only sends a header and the first part of
23  * the file in one packet, but also sends the file pages in the full packets.
24  *
25  * But until FreeBSD 4.5 turning TCP_NOPUSH off does not flush a pending
26  * data that less than MSS, so that data may be sent with 5 second delay.
27  * So we do not use TCP_NOPUSH on FreeBSD prior to 4.5, although it can be used
28  * for non-keepalive HTTP connections.
29  */
30
31
32  ngx_chain_t *
33  ngx_freebsd_sendfile_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
34  {
35      int              rc, flags;
36      off_t            send, prev_send, sent;
37      size_t           file_size;
38      ssize_t          n;
39      ngx_uint_t       eintr, eagain;
40      ngx_err_t        err;
41      ngx_buf_t       *file;
42      ngx_event_t     *wev;
43      ngx_chain_t     *cl;
44      ngx_iovec_t      header, trailer;
45      struct sf_hdtr   hdtr;
46      struct iovec     headers[NGX_IOVS_PREALLOCATE];
47      struct iovec     trailers[NGX_IOVS_PREALLOCATE];
48
49      wev = c->write;
50
51      if (!wev->ready) {
52          return in;
53      }
54
55  #if (NGX_HAVE_KQUEUE)
56
57      if ((ngx_event_flags & NGX_USE_KQUEUE_EVENT) && wev->pending_eof) {
58          (void) ngx_connection_error(c, wev->kq_errno,
59                                      "kevent() reported about an closed connection");
60          wev->error = 1;
61          return NGX_CHAIN_ERROR;
62      }
```

```
63
64  #endif
65
66      /* the maximum limit size is the maximum size_t value - the page size */
67
68      if (limit == 0 || limit > (off_t) (NGX_MAX_SIZE_T_VALUE - ngx_pagesize)) {
69          limit = NGX_MAX_SIZE_T_VALUE - ngx_pagesize;
70      }
71
72      send = 0;
73      eagain = 0;
74      flags = 0;
75
76      header.iovs = headers;
77      header.nalloc = NGX_IOVS_PREALLOCATE;
78
79      trailer.iovs = trailers;
80      trailer.nalloc = NGX_IOVS_PREALLOCATE;
81
82      for ( ;; ) {
83          eintr = 0;
84          prev_send = send;
85
86          /* create the header iovec and coalesce the neighbouring bufs */
87
88          cl = ngx_output_chain_to_iovec(&header, in, limit - send, c->log);
89
90          if (cl == NGX_CHAIN_ERROR) {
91              return NGX_CHAIN_ERROR;
92          }
93
94          send += header.size;
95
96          if (cl && cl->buf->in_file && send < limit) {
97              file = cl->buf;
98
99              /* coalesce the neighbouring file bufs */
100
101             file_size = (size_t) ngx_chain_coalesce_file(&cl, limit - send);
102
103             send += file_size;
104
105             /* create the trailer iovec and coalesce the neighbouring bufs */
106
107             cl = ngx_output_chain_to_iovec(&trailer, cl, limit - send, c->log);
108
109             if (cl == NGX_CHAIN_ERROR) {
110                 return NGX_CHAIN_ERROR;
111             }
112
113             send += trailer.size;
114
115             if (ngx_freebsd_use_tcp_nopush
116                 && c->tcp_nopush == NGX_TCP_NOPUSH_UNSET)
117             {
118                 if (ngx_tcp_nopush(c->fd) == NGX_ERROR) {
119                     err = ngx_socket_errno;
120
121                     /*
122                      * there is a tiny chance to be interrupted, however,
123                      * we continue a processing without the TCP_NOPUSH
124                      */
125
126                     if (err != NGX_EINTR) {
127                         wev->error = 1;
128                         (void) ngx_connection_error(c, err,
129                                                     ngx_tcp_nopush_n " failed");
130                         return NGX_CHAIN_ERROR;
131                     }
132
133                 } else {
134                     c->tcp_nopush = NGX_TCP_NOPUSH_SET;
135
136                     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
137                                    "tcp_nopush");
138                 }
```

```
            }

            /*
             * sendfile() does unneeded work if sf_hdtr's count is 0,
             * but corresponding pointer is not NULL
             */

            hdtr.headers = header.count ? header.iovs : NULL;
            hdtr.hdr_cnt = header.count;
            hdtr.trailers = trailer.count ? trailer.iovs : NULL;
            hdtr.trl_cnt = trailer.count;

            /*
             * the "nbytes bug" of the old sendfile() syscall:
             * http://bugs.freebsd.org/33771
             */

            if (!ngx_freebsd_sendfile_nbytes_bug) {
                header.size = 0;
            }

            sent = 0;

#if (NGX_HAVE_AIO_SENDFILE)
            flags = c->aio_sendfile ? SF_NODISKIO : 0;
#endif

            rc = sendfile(file->file->fd, c->fd, file->file_pos,
                          file_size + header.size, &hdtr, &sent, flags);

            if (rc == -1) {
                err = ngx_errno;

                switch (err) {
                case NGX_EAGAIN:
                    eagain = 1;
                    break;

                case NGX_EINTR:
                    eintr = 1;
                    break;

#if (NGX_HAVE_AIO_SENDFILE)
                case NGX_EBUSY:
                    c->busy_sendfile = file;
                    break;
#endif

                default:
                    wev->error = 1;
                    (void) ngx_connection_error(c, err, "sendfile() failed");
                    return NGX_CHAIN_ERROR;
                }

                ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, err,
                               "sendfile() sent only %O bytes", sent);

            /*
             * sendfile() in FreeBSD 3.x-4.x may return value >= 0
             * on success, although only 0 is documented
             */

            } else if (rc >= 0 && sent == 0) {

                /*
                 * if rc is OK and sent equal to zero, then someone
                 * has truncated the file, so the offset became beyond
                 * the end of the file
                 */

                ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                              "sendfile() reported that \"%s\" was truncated at %O",
                              file->file->name.data, file->file_pos);

                return NGX_CHAIN_ERROR;
            }
```

```
215
216             ngx_log_debug4(NGX_LOG_DEBUG_EVENT, c->log, 0,
217                            "sendfile: %d, @%O %O:%uz",
218                            rc, file->file_pos, sent, file_size + header.size);
219
220         } else {
221             n = ngx_writev(c, &header);
222
223             if (n == NGX_ERROR) {
224                 return NGX_CHAIN_ERROR;
225             }
226
227             sent = (n == NGX_AGAIN) ? 0 : n;
228         }
229
230         c->sent += sent;
231
232         in = ngx_chain_update_sent(in, sent);
233
234 #if (NGX_HAVE_AIO_SENDFILE)
235         if (c->busy_sendfile) {
236             return in;
237         }
238 #endif
239
240         if (eagain) {
241
242             /*
243              * sendfile() may return EAGAIN, even if it has sent a whole file
244              * part, it indicates that the successive sendfile() call would
245              * return EAGAIN right away and would not send anything.
246              * We use it as a hint.
247              */
248
249             wev->ready = 0;
250             return in;
251         }
252
253         if (eintr) {
254             send = prev_send + sent;
255             continue;
256         }
257
258         if (send - prev_send != sent) {
259             wev->ready = 0;
260             return in;
261         }
262
263         if (send >= limit || in == NULL) {
264             return in;
265         }
266     }
267 }
```

One Level Up Top Level

src/os/unix/ngx_daemon.c - nginx-1.7.10

Functions defined

- ngx_daemon

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  ngx_int_t
13  ngx_daemon(ngx_log_t *log)
14  {
15      int  fd;
16
17      switch (fork()) {
18      case -1:
19          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "fork() failed");
20          return NGX_ERROR;
21
22      case 0:
23          break;
24
25      default:
26          exit(0);
27      }
28
29      ngx_pid = ngx_getpid();
30
31      if (setsid() == -1) {
32          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "setsid() failed");
33          return NGX_ERROR;
34      }
35
36      umask(0);
37
38      fd = open("/dev/null", O_RDWR);
39      if (fd == -1) {
40          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno,
41                        "open(\"/dev/null\") failed");
42          return NGX_ERROR;
43      }
44
45      if (dup2(fd, STDIN_FILENO) == -1) {
46          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "dup2(STDIN) failed");
47          return NGX_ERROR;
48      }
49
50      if (dup2(fd, STDOUT_FILENO) == -1) {
51          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "dup2(STDOUT) failed");
52          return NGX_ERROR;
53      }
54
55  #if 0
56      if (dup2(fd, STDERR_FILENO) == -1) {
57          ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "dup2(STDERR) failed");
58          return NGX_ERROR;
59      }
60  #endif
61
62      if (fd > STDERR_FILENO) {
```

```
63          if (close(fd) == -1) {
64              ngx_log_error(NGX_LOG_EMERG, log, ngx_errno, "close() failed");
65              return NGX_ERROR;
66          }
67      }
68
69      return NGX_OK;
70  }
```

One Level Up Top Level

src/os/unix/ngx_aio_read_chain.c - nginx-1.7.10

Functions defined

- ngx_aio_read_chain

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ssize_t
14  ngx_aio_read_chain(ngx_connection_t *c, ngx_chain_t *cl, off_t limit)
15  {
16      int           n;
17      u_char       *buf, *prev;
18      size_t        size;
19      ssize_t       total;
20
21      if (c->read->pending_eof) {
22          c->read->ready = 0;
23          return 0;
24      }
25
26      total = 0;
27
28      while (cl) {
29
30          /* we can post the single aio operation only */
31
32          if (!c->read->ready) {
33              return total ? total : NGX_AGAIN;
34          }
35
36          buf = cl->buf->last;
37          prev = cl->buf->last;
38          size = 0;
39
40          /* coalesce the neighbouring bufs */
41
42          while (cl && prev == cl->buf->last) {
43              size += cl->buf->end - cl->buf->last;
44              prev = cl->buf->end;
45              cl = cl->next;
46          }
47
48          n = ngx_aio_read(c, buf, size);
49
50          ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "aio_read: %d", n);
51
52          if (n == NGX_AGAIN) {
53              return total ? total : NGX_AGAIN;
54          }
55
56          if (n == NGX_ERROR) {
57              return NGX_ERROR;
58          }
59
60          if (n == 0) {
61              c->read->pending_eof = 1;
62              if (total) {
```

```
63                  c->read->eof = 0;
64                  c->read->ready = 1;
65              }
66              return total;
67          }
68
69          if (n > 0) {
70              total += n;
71          }
72
73          ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
74                         "aio_read total: %d", total);
75      }
76
77      return total ? total : NGX_AGAIN;
78  }
```

One Level Up Top Level

src/os/unix/ngx_aio_write.c - nginx-1.7.10

Functions defined

- ngx_aio_write

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  extern int  ngx_kqueue;
14
15
16  ssize_t
17  ngx_aio_write(ngx_connection_t *c, u_char *buf, size_t size)
18  {
19      int           n;
20      ngx_event_t  *wev;
21
22      wev = c->write;
23
24      if (!wev->ready) {
25          return NGX_AGAIN;
26      }
27
28      ngx_log_debug1(NGX_LOG_DEBUG_EVENT, wev->log, 0,
29                     "aio: wev->complete: %d", wev->complete);
30
31      if (!wev->complete) {
32          ngx_memzero(&wev->aiocb, sizeof(struct aiocb));
33
34          wev->aiocb.aio_fildes = c->fd;
35          wev->aiocb.aio_buf = buf;
36          wev->aiocb.aio_nbytes = size;
37
38  #if (NGX_HAVE_KQUEUE)
39          wev->aiocb.aio_sigevent.sigev_notify_kqueue = ngx_kqueue;
40          wev->aiocb.aio_sigevent.sigev_notify = SIGEV_KEVENT;
41          wev->aiocb.aio_sigevent.sigev_value.sigval_ptr = wev;
42  #endif
43
44          if (aio_write(&wev->aiocb) == -1) {
45              ngx_log_error(NGX_LOG_CRIT, wev->log, ngx_errno,
46                            "aio_write() failed");
47              return NGX_ERROR;
48          }
49
50          ngx_log_debug0(NGX_LOG_DEBUG_EVENT, wev->log, 0, "aio_write: OK");
51
52          wev->active = 1;
53          wev->ready = 0;
54      }
55
56      wev->complete = 0;
57
58      n = aio_error(&wev->aiocb);
59      if (n == -1) {
60          ngx_log_error(NGX_LOG_CRIT, wev->log, ngx_errno, "aio_error() failed");
61          wev->error = 1;
62          return NGX_ERROR;
```

```
63      }
64
65      if (n != 0) {
66          if (n == NGX_EINPROGRESS) {
67              if (wev->ready) {
68                  ngx_log_error(NGX_LOG_ALERT, wev->log, n,
69                                "aio_write() still in progress");
70                  wev->ready = 0;
71              }
72              return NGX_AGAIN;
73          }
74
75          ngx_log_error(NGX_LOG_CRIT, wev->log, n, "aio_write() failed");
76          wev->error = 1;
77          wev->ready = 0;
78
79  #if 1
80          n = aio_return(&wev->aiocb);
81          if (n == -1) {
82              ngx_log_error(NGX_LOG_ALERT, wev->log, ngx_errno,
83                            "aio_return() failed");
84          }
85
86          ngx_log_error(NGX_LOG_CRIT, wev->log, n, "aio_return() %d", n);
87  #endif
88
89          return NGX_ERROR;
90      }
91
92      n = aio_return(&wev->aiocb);
93      if (n == -1) {
94          ngx_log_error(NGX_LOG_ALERT, wev->log, ngx_errno,
95                        "aio_return() failed");
96
97          wev->error = 1;
98          wev->ready = 0;
99          return NGX_ERROR;
100     }
101
102
103     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, wev->log, 0, "aio_write: %d", n);
104
105     wev->active = 0;
106     wev->ready = 1;
107
108     return n;
109 }
```

One Level Up Top Level

src/os/unix/ngx_aio_write_chain.c - nginx-1.7.10

Functions defined

- ngx_aio_write_chain

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ngx_chain_t *
14  ngx_aio_write_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
15  {
16      u_char       *buf, *prev;
17      off_t         send, sent;
18      size_t        len;
19      ssize_t       n, size;
20      ngx_chain_t  *cl;
21
22      /* the maximum limit size is the maximum size_t value - the page size */
23
24      if (limit == 0 || limit > (off_t) (NGX_MAX_SIZE_T_VALUE - ngx_pagesize)) {
25          limit = NGX_MAX_SIZE_T_VALUE - ngx_pagesize;
26      }
27
28      send = 0;
29      sent = 0;
30      cl = in;
31
32      while (cl) {
33
34          if (cl->buf->pos == cl->buf->last) {
35              cl = cl->next;
36              continue;
37          }
38
39          /* we can post the single aio operation only */
40
41          if (!c->write->ready) {
42              return cl;
43          }
44
45          buf = cl->buf->pos;
46          prev = buf;
47          len = 0;
48
49          /* coalesce the neighbouring bufs */
50
51          while (cl && prev == cl->buf->pos && send < limit) {
52              if (ngx_buf_special(cl->buf)) {
53                  continue;
54              }
55
56              size = cl->buf->last - cl->buf->pos;
57
58              if (send + size > limit) {
59                  size = limit - send;
60              }
61
62              len += size;
```

```
63              prev = cl->buf->pos + size;
64              send += size;
65              cl = cl->next;
66          }
67
68          n = ngx_aio_write(c, buf, len);
69
70          ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "aio_write: %z", n);
71
72          if (n == NGX_ERROR) {
73              return NGX_CHAIN_ERROR;
74          }
75
76          if (n > 0) {
77              sent += n;
78              c->sent += n;
79          }
80
81          ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0,
82                         "aio_write sent: %O", c->sent);
83
84          for (cl = in; cl; cl = cl->next) {
85
86              if (sent >= cl->buf->last - cl->buf->pos) {
87                  sent -= cl->buf->last - cl->buf->pos;
88                  cl->buf->pos = cl->buf->last;
89
90                  continue;
91              }
92
93              cl->buf->pos += sent;
94
95              break;
96          }
97      }
98
99      return cl;
100 }
```

One Level Up Top Level

src/core/ngx_cpuinfo.c - nginx-1.7.10

Functions defined

- ngx_cpuid
- ngx_cpuid
- ngx_cpuinfo
- ngx_cpuinfo

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 #if (( __i386__ || __amd64__ ) && ( __GNUC__ || __INTEL_COMPILER ))
13
14
15 static ngx_inline void ngx_cpuid(uint32_t i, uint32_t *buf);
16
17
18 #if ( __i386__ )
19
20 static ngx_inline void
21 ngx_cpuid(uint32_t i, uint32_t *buf)
22 {
23
24     /*
25      * we could not use %ebx as output parameter if gcc builds PIC,
26      * and we could not save %ebx on stack, because %esp is used,
27      * when the -fomit-frame-pointer optimization is specified.
28      */
29
30     __asm__ (
31
32     "    mov    %%ebx, %%esi;  "
33
34     "    cpuid;                "
35     "    mov    %%eax, (%1);   "
36     "    mov    %%ebx, 4(%1);  "
37     "    mov    %%edx, 8(%1);  "
38     "    mov    %%ecx, 12(%1); "
39
40     "    mov    %%esi, %%ebx;  "
41
42     : : "a" (i), "D" (buf) : "ecx", "edx", "esi", "memory" );
43 }
44
45
46 #else /* __amd64__ */
47
48
49 static ngx_inline void
50 ngx_cpuid(uint32_t i, uint32_t *buf)
51 {
52     uint32_t  eax, ebx, ecx, edx;
53
54     __asm__ (
55
56         "cpuid"
```

```
57
58      : "=a" (eax), "=b" (ebx), "=c" (ecx), "=d" (edx) : "a" (i) );
59
60      buf[0] = eax;
61      buf[1] = ebx;
62      buf[2] = edx;
63      buf[3] = ecx;
64  }
65
66
67  #endif
68
69
70  /* auto detect the L2 cache line size of modern and widespread CPUs */
71
72  void
73  ngx_cpuinfo(void)
74  {
75      u_char    *vendor;
76      uint32_t   vbuf[5], cpu[4], model;
77
78      vbuf[0] = 0;
79      vbuf[1] = 0;
80      vbuf[2] = 0;
81      vbuf[3] = 0;
82      vbuf[4] = 0;
83
84      ngx_cpuid(0, vbuf);
85
86      vendor = (u_char *) &vbuf[1];
87
88      if (vbuf[0] == 0) {
89          return;
90      }
91
92      ngx_cpuid(1, cpu);
93
94      if (ngx_strcmp(vendor, "GenuineIntel") == 0) {
95
96          switch ((cpu[0] & 0xf00) >> 8) {
97
98          /* Pentium */
99          case 5:
100             ngx_cacheline_size = 32;
101             break;
102
103         /* Pentium Pro, II, III */
104         case 6:
105             ngx_cacheline_size = 32;
106
107             model = ((cpu[0] & 0xf0000) >> 8) | (cpu[0] & 0xf0);
108
109             if (model >= 0xd0) {
110                 /* Intel Core, Core 2, Atom */
111                 ngx_cacheline_size = 64;
112             }
113
114             break;
115
116         /*
117          * Pentium 4, although its cache line size is 64 bytes,
118          * it prefetches up to two cache lines during memory read
119          */
120         case 15:
121             ngx_cacheline_size = 128;
122             break;
123         }
124
125     } else if (ngx_strcmp(vendor, "AuthenticAMD") == 0) {
126         ngx_cacheline_size = 64;
127     }
128 }
129
130 #else
131
132
```

```
133 void
134 ngx_cpuinfo(void)
135 {
136 }
137
138
139 #endif
```

One Level Up Top Level

src/os/unix/ngx_time.c - nginx-1.7.10

Functions defined

- ngx_libc_gmtime
- ngx_libc_localtime
- ngx_localtime
- ngx_timezone_update

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  /*
13  * FreeBSD does not test /etc/localtime change, however, we can workaround it
14  * by calling tzset() with TZ and then without TZ to update timezone.
15  * The trick should work since FreeBSD 2.1.0.
16  *
17  * Linux does not test /etc/localtime change in localtime(),
18  * but may stat("/etc/localtime") several times in every strftime(),
19  * therefore we use it to update timezone.
20  *
21  * Solaris does not test /etc/TIMEZONE change too and no workaround available.
22  */
23
24  void
25  ngx_timezone_update(void)
26  {
27  #if (NGX_FREEBSD)
28
29      if (getenv("TZ")) {
30          return;
31      }
32
33      putenv("TZ=UTC");
34
35      tzset();
36
37      unsetenv("TZ");
38
39      tzset();
40
41  #elif (NGX_LINUX)
42      time_t      s;
43      struct tm  *t;
44      char        buf[4];
45
46      s = time(0);
47
48      t = localtime(&s);
49
50      strftime(buf, 4, "%H", t);
51
52  #endif
53  }
54
55
56  void
```

```
57  ngx_localtime(time_t s, ngx_tm_t *tm)
58  {
59  #if (NGX_HAVE_LOCALTIME_R)
60      (void) localtime_r(&s, tm);
61
62  #else
63      ngx_tm_t  *t;
64
65      t = localtime(&s);
66      *tm = *t;
67
68  #endif
69
70      tm->ngx_tm_mon++;
71      tm->ngx_tm_year += 1900;
72  }
73
74
75  void
76  ngx_libc_localtime(time_t s, struct tm *tm)
77  {
78  #if (NGX_HAVE_LOCALTIME_R)
79      (void) localtime_r(&s, tm);
80
81  #else
82      struct tm  *t;
83
84      t = localtime(&s);
85      *tm = *t;
86
87  #endif
88  }
89
90
91  void
92  ngx_libc_gmtime(time_t s, struct tm *tm)
93  {
94  #if (NGX_HAVE_LOCALTIME_R)
95      (void) gmtime_r(&s, tm);
96
97  #else
98      struct tm  *t;
99
100      t = gmtime(&s);
101      *tm = *t;
102
103  #endif
104  }
```

One Level Up Top Level

src/http/modules/ngx_http_limit_conn_module.c - nginx-1.7.10

Global variables defined

- ngx_http_limit_conn_commands
- ngx_http_limit_conn_log_levels
- ngx_http_limit_conn_module
- ngx_http_limit_conn_module_ctx
- ngx_http_limit_conn_status_bounds

Data types defined

- ngx_http_limit_conn_cleanup_t
- ngx_http_limit_conn_conf_t
- ngx_http_limit_conn_ctx_t
- ngx_http_limit_conn_limit_t
- ngx_http_limit_conn_node_t

Functions defined

- ngx_http_limit_conn
- ngx_http_limit_conn_cleanup
- ngx_http_limit_conn_cleanup_all
- ngx_http_limit_conn_create_conf
- ngx_http_limit_conn_handler
- ngx_http_limit_conn_init
- ngx_http_limit_conn_init_zone
- ngx_http_limit_conn_lookup
- ngx_http_limit_conn_merge_conf
- ngx_http_limit_conn_rbtree_insert_value
- ngx_http_limit_conn_zone

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
```

```
11
12
13  typedef struct {
14      u_char                     color;
15      u_char                     len;
16      u_short                    conn;
17      u_char                     data[1];
18  } ngx_http_limit_conn_node_t;
19
20
21  typedef struct {
22      ngx_shm_zone_t            *shm_zone;
23      ngx_rbtree_node_t         *node;
24  } ngx_http_limit_conn_cleanup_t;
25
26
27  typedef struct {
28      ngx_rbtree_t              *rbtree;
29      ngx_http_complex_value_t   key;
30  } ngx_http_limit_conn_ctx_t;
31
32
33  typedef struct {
34      ngx_shm_zone_t            *shm_zone;
35      ngx_uint_t                 conn;
36  } ngx_http_limit_conn_limit_t;
37
38
39  typedef struct {
40      ngx_array_t                limits;
41      ngx_uint_t                 log_level;
42      ngx_uint_t                 status_code;
43  } ngx_http_limit_conn_conf_t;
44
45
46  static ngx_rbtree_node_t *ngx_http_limit_conn_lookup(ngx_rbtree_t *rbtree,
47      ngx_str_t *key, uint32_t hash);
48  static void ngx_http_limit_conn_cleanup(void *data);
49  static ngx_inline void ngx_http_limit_conn_cleanup_all(ngx_pool_t *pool);
50
51  static void *ngx_http_limit_conn_create_conf(ngx_conf_t *cf);
52  static char *ngx_http_limit_conn_merge_conf(ngx_conf_t *cf, void *parent,
53      void *child);
54  static char *ngx_http_limit_conn_zone(ngx_conf_t *cf, ngx_command_t *cmd,
55      void *conf);
56  static char *ngx_http_limit_conn(ngx_conf_t *cf, ngx_command_t *cmd,
57      void *conf);
58  static ngx_int_t ngx_http_limit_conn_init(ngx_conf_t *cf);
59
60
61  static ngx_conf_enum_t  ngx_http_limit_conn_log_levels[] = {
62      { ngx_string("info"), NGX_LOG_INFO },
63      { ngx_string("notice"), NGX_LOG_NOTICE },
64      { ngx_string("warn"), NGX_LOG_WARN },
65      { ngx_string("error"), NGX_LOG_ERR },
66      { ngx_null_string, 0 }
67  };
68
69
70  static ngx_conf_num_bounds_t  ngx_http_limit_conn_status_bounds = {
71      ngx_conf_check_num_bounds, 400, 599
72  };
73
74
75  static ngx_command_t  ngx_http_limit_conn_commands[] = {
76
77      { ngx_string("limit_conn_zone"),
78        NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE2,
79        ngx_http_limit_conn_zone,
80        0,
81        0,
82        NULL },
83
84      { ngx_string("limit_conn"),
85        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
86        ngx_http_limit_conn,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("limit_conn_log_level"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_limit_conn_conf_t, log_level),
      &ngx_http_limit_conn_log_levels },

    { ngx_string("limit_conn_status"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_limit_conn_conf_t, status_code),
      &ngx_http_limit_conn_status_bounds },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_limit_conn_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_limit_conn_init,              /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_limit_conn_create_conf,       /* create location configuration */
    ngx_http_limit_conn_merge_conf         /* merge location configuration */
};


ngx_module_t  ngx_http_limit_conn_module = {
    NGX_MODULE_V1,
    &ngx_http_limit_conn_module_ctx,       /* module context */
    ngx_http_limit_conn_commands,          /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_limit_conn_handler(ngx_http_request_t *r)
{
    size_t                      n;
    uint32_t                    hash;
    ngx_str_t                   key;
    ngx_uint_t                  i;
    ngx_slab_pool_t            *shpool;
    ngx_rbtree_node_t          *node;
    ngx_pool_cleanup_t         *cln;
    ngx_http_limit_conn_ctx_t  *ctx;
    ngx_http_limit_conn_node_t *lc;
    ngx_http_limit_conn_conf_t *lccf;
    ngx_http_limit_conn_limit_t *limits;
    ngx_http_limit_conn_cleanup_t *lccln;

    if (r->main->limit_conn_set) {
        return NGX_DECLINED;
    }

    lccf = ngx_http_get_module_loc_conf(r, ngx_http_limit_conn_module);
    limits = lccf->limits.elts;

```

```
    for (i = 0; i < lccf->limits.nelts; i++) {
        ctx = limits[i].shm_zone->data;

        if (ngx_http_complex_value(r, &ctx->key, &key) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        if (key.len == 0) {
            continue;
        }

        if (key.len > 255) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "the value of the \"%V\" key "
                          "is more than 255 bytes: \"%V\"",
                          &ctx->key.value, &key);
            continue;
        }

        r->main->limit_conn_set = 1;

        hash = ngx_crc32_short(key.data, key.len);

        shpool = (ngx_slab_pool_t *) limits[i].shm_zone->shm.addr;

        ngx_shmtx_lock(&shpool->mutex);

        node = ngx_http_limit_conn_lookup(ctx->rbtree, &key, hash);

        if (node == NULL) {

            n = offsetof(ngx_rbtree_node_t, color)
                + offsetof(ngx_http_limit_conn_node_t, data)
                + key.len;

            node = ngx_slab_alloc_locked(shpool, n);

            if (node == NULL) {
                ngx_shmtx_unlock(&shpool->mutex);
                ngx_http_limit_conn_cleanup_all(r->pool);
                return lccf->status_code;
            }

            lc = (ngx_http_limit_conn_node_t *) &node->color;

            node->key = hash;
            lc->len = (u_char) key.len;
            lc->conn = 1;
            ngx_memcpy(lc->data, key.data, key.len);

            ngx_rbtree_insert(ctx->rbtree, node);

        } else {

            lc = (ngx_http_limit_conn_node_t *) &node->color;

            if ((ngx_uint_t) lc->conn >= limits[i].conn) {

                ngx_shmtx_unlock(&shpool->mutex);

                ngx_log_error(lccf->log_level, r->connection->log, 0,
                              "limiting connections by zone \"%V\"",
                              &limits[i].shm_zone->shm.name);

                ngx_http_limit_conn_cleanup_all(r->pool);
                return lccf->status_code;
            }

            lc->conn++;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "limit conn: %08XD %d", node->key, lc->conn);

        ngx_shmtx_unlock(&shpool->mutex);

```

```
        cln = ngx_pool_cleanup_add(r->pool,
                                   sizeof(ngx_http_limit_conn_cleanup_t));
        if (cln == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        cln->handler = ngx_http_limit_conn_cleanup;
        lccln = cln->data;

        lccln->shm_zone = limits[i].shm_zone;
        lccln->node = node;
    }

    return NGX_DECLINED;
}


static void
ngx_http_limit_conn_rbtree_insert_value(ngx_rbtree_node_t *temp,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
{
    ngx_rbtree_node_t           **p;
    ngx_http_limit_conn_node_t   *lcn, *lcnt;

    for ( ;; ) {

        if (node->key < temp->key) {

            p = &temp->left;

        } else if (node->key > temp->key) {

            p = &temp->right;

        } else { /* node->key == temp->key */

            lcn = (ngx_http_limit_conn_node_t *) &node->color;
            lcnt = (ngx_http_limit_conn_node_t *) &temp->color;

            p = (ngx_memn2cmp(lcn->data, lcnt->data, lcn->len, lcnt->len) < 0)
                ? &temp->left : &temp->right;
        }

        if (*p == sentinel) {
            break;
        }

        temp = *p;
    }

    *p = node;
    node->parent = temp;
    node->left = sentinel;
    node->right = sentinel;
    ngx_rbt_red(node);
}


static ngx_rbtree_node_t *
ngx_http_limit_conn_lookup(ngx_rbtree_t *rbtree, ngx_str_t *key, uint32_t hash)
{
    ngx_int_t                    rc;
    ngx_rbtree_node_t           *node, *sentinel;
    ngx_http_limit_conn_node_t  *lcn;

    node = rbtree->root;
    sentinel = rbtree->sentinel;

    while (node != sentinel) {

        if (hash < node->key) {
            node = node->left;
            continue;
        }

        if (hash > node->key) {
```

```
            node = node->right;
            continue;
        }

        /* hash == node->key */

        lcn = (ngx_http_limit_conn_node_t *) &node->color;

        rc = ngx_memn2cmp(key->data, lcn->data, key->len, (size_t) lcn->len);

        if (rc == 0) {
            return node;
        }

        node = (rc < 0) ? node->left : node->right;
    }

    return NULL;
}


static void
ngx_http_limit_conn_cleanup(void *data)
{
    ngx_http_limit_conn_cleanup_t  *lccln = data;

    ngx_slab_pool_t             *shpool;
    ngx_rbtree_node_t           *node;
    ngx_http_limit_conn_ctx_t   *ctx;
    ngx_http_limit_conn_node_t  *lc;

    ctx = lccln->shm_zone->data;
    shpool = (ngx_slab_pool_t *) lccln->shm_zone->shm.addr;
    node = lccln->node;
    lc = (ngx_http_limit_conn_node_t *) &node->color;

    ngx_shmtx_lock(&shpool->mutex);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, lccln->shm_zone->shm.log, 0,
                   "limit conn cleanup: %08XD %d", node->key, lc->conn);

    lc->conn--;

    if (lc->conn == 0) {
        ngx_rbtree_delete(ctx->rbtree, node);
        ngx_slab_free_locked(shpool, node);
    }

    ngx_shmtx_unlock(&shpool->mutex);
}


static ngx_inline void
ngx_http_limit_conn_cleanup_all(ngx_pool_t *pool)
{
    ngx_pool_cleanup_t  *cln;

    cln = pool->cleanup;

    while (cln && cln->handler == ngx_http_limit_conn_cleanup) {
        ngx_http_limit_conn_cleanup(cln->data);
        cln = cln->next;
    }

    pool->cleanup = cln;
}


static ngx_int_t
ngx_http_limit_conn_init_zone(ngx_shm_zone_t *shm_zone, void *data)
{
    ngx_http_limit_conn_ctx_t  *octx = data;

    size_t                      len;
    ngx_slab_pool_t            *shpool;
    ngx_rbtree_node_t          *sentinel;
```

```
    ngx_http_limit_conn_ctx_t  *ctx;

    ctx = shm_zone->data;

    if (octx) {
        if (ctx->key.value.len != octx->key.value.len
            || ngx_strncmp(ctx->key.value.data, octx->key.value.data,
                           ctx->key.value.len)
               != 0)
        {
            ngx_log_error(NGX_LOG_EMERG, shm_zone->shm.log, 0,
                          "limit_conn_zone \"%V\" uses the \"%V\" key "
                          "while previously it used the \"%V\" key",
                          &shm_zone->shm.name, &ctx->key.value,
                          &octx->key.value);
            return NGX_ERROR;
        }

        ctx->rbtree = octx->rbtree;

        return NGX_OK;
    }

    shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;

    if (shm_zone->shm.exists) {
        ctx->rbtree = shpool->data;

        return NGX_OK;
    }

    ctx->rbtree = ngx_slab_alloc(shpool, sizeof(ngx_rbtree_t));
    if (ctx->rbtree == NULL) {
        return NGX_ERROR;
    }

    shpool->data = ctx->rbtree;

    sentinel = ngx_slab_alloc(shpool, sizeof(ngx_rbtree_node_t));
    if (sentinel == NULL) {
        return NGX_ERROR;
    }

    ngx_rbtree_init(ctx->rbtree, sentinel,
                    ngx_http_limit_conn_rbtree_insert_value);

    len = sizeof(" in limit_conn_zone \"\"") + shm_zone->shm.name.len;

    shpool->log_ctx = ngx_slab_alloc(shpool, len);
    if (shpool->log_ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_sprintf(shpool->log_ctx, " in limit_conn_zone \"%V\"%Z",
                &shm_zone->shm.name);

    return NGX_OK;
}


static void *
ngx_http_limit_conn_create_conf(ngx_conf_t *cf)
{
    ngx_http_limit_conn_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_limit_conn_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->limits.elts = NULL;
     */

```

```
    conf->log_level = NGX_CONF_UNSET_UINT;
    conf->status_code = NGX_CONF_UNSET_UINT;

    return conf;
}


static char *
ngx_http_limit_conn_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_limit_conn_conf_t *prev = parent;
    ngx_http_limit_conn_conf_t *conf = child;

    if (conf->limits.elts == NULL) {
        conf->limits = prev->limits;
    }

    ngx_conf_merge_uint_value(conf->log_level, prev->log_level, NGX_LOG_ERR);
    ngx_conf_merge_uint_value(conf->status_code, prev->status_code,
                              NGX_HTTP_SERVICE_UNAVAILABLE);

    return NGX_CONF_OK;
}


static char *
ngx_http_limit_conn_zone(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    u_char                              *p;
    ssize_t                              size;
    ngx_str_t                           *value, name, s;
    ngx_uint_t                           i;
    ngx_shm_zone_t                      *shm_zone;
    ngx_http_limit_conn_ctx_t           *ctx;
    ngx_http_compile_complex_value_t     ccv;

    value = cf->args->elts;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_limit_conn_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &ctx->key;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    size = 0;
    name.len = 0;

    for (i = 2; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "zone=", 5) == 0) {

            name.data = value[i].data + 5;

            p = (u_char *) ngx_strchr(name.data, ':');

            if (p == NULL) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid zone size \"%V\"", &value[i]);
                return NGX_CONF_ERROR;
            }

            name.len = p - name.data;

            s.data = p + 1;
            s.len = value[i].data + value[i].len - s.data;

            size = ngx_parse_size(&s);
```

```
543
544             if (size == NGX_ERROR) {
545                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
546                                    "invalid zone size \"%V\"", &value[i]);
547                 return NGX_CONF_ERROR;
548             }
549
550             if (size < (ssize_t) (8 * ngx_pagesize)) {
551                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
552                                    "zone \"%V\" is too small", &value[i]);
553                 return NGX_CONF_ERROR;
554             }
555
556             continue;
557         }
558
559         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
560                            "invalid parameter \"%V\"", &value[i]);
561         return NGX_CONF_ERROR;
562     }
563
564     if (name.len == 0) {
565         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
566                            "\"%V\" must have \"zone\" parameter",
567                            &cmd->name);
568         return NGX_CONF_ERROR;
569     }
570
571     shm_zone = ngx_shared_memory_add(cf, &name, size,
572                                      &ngx_http_limit_conn_module);
573     if (shm_zone == NULL) {
574         return NGX_CONF_ERROR;
575     }
576
577     if (shm_zone->data) {
578         ctx = shm_zone->data;
579
580         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
581                            "%V \"%V\" is already bound to key \"%V\"",
582                            &cmd->name, &name, &ctx->key.value);
583         return NGX_CONF_ERROR;
584     }
585
586     shm_zone->init = ngx_http_limit_conn_init_zone;
587     shm_zone->data = ctx;
588
589     return NGX_CONF_OK;
590 }
591
592
593 static char *
594 ngx_http_limit_conn(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
595 {
596     ngx_shm_zone_t               *shm_zone;
597     ngx_http_limit_conn_conf_t   *lccf = conf;
598     ngx_http_limit_conn_limit_t  *limit, *limits;
599
600     ngx_str_t  *value;
601     ngx_int_t   n;
602     ngx_uint_t  i;
603
604     value = cf->args->elts;
605
606     shm_zone = ngx_shared_memory_add(cf, &value[1], 0,
607                                      &ngx_http_limit_conn_module);
608     if (shm_zone == NULL) {
609         return NGX_CONF_ERROR;
610     }
611
612     limits = lccf->limits.elts;
613
614     if (limits == NULL) {
615         if (ngx_array_init(&lccf->limits, cf->pool, 1,
616                            sizeof(ngx_http_limit_conn_limit_t))
617             != NGX_OK)
618         {
```

```
619             return NGX_CONF_ERROR;
620         }
621     }
622
623     for (i = 0; i < lccf->limits.nelts; i++) {
624         if (shm_zone == limits[i].shm_zone) {
625             return "is duplicate";
626         }
627     }
628
629     n = ngx_atoi(value[2].data, value[2].len);
630     if (n <= 0) {
631         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
632                            "invalid number of connections \"%V\"", &value[2]);
633         return NGX_CONF_ERROR;
634     }
635
636     if (n > 65535) {
637         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
638                            "connection limit must be less 65536");
639         return NGX_CONF_ERROR;
640     }
641
642     limit = ngx_array_push(&lccf->limits);
643     if (limit == NULL) {
644         return NGX_CONF_ERROR;
645     }
646
647     limit->conn = n;
648     limit->shm_zone = shm_zone;
649
650     return NGX_CONF_OK;
651 }
652
653
654 static ngx_int_t
655 ngx_http_limit_conn_init(ngx_conf_t *cf)
656 {
657     ngx_http_handler_pt        *h;
658     ngx_http_core_main_conf_t  *cmcf;
659
660     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
661
662     h = ngx_array_push(&cmcf->phases[NGX_HTTP_PREACCESS_PHASE].handlers);
663     if (h == NULL) {
664         return NGX_ERROR;
665     }
666
667     *h = ngx_http_limit_conn_handler;
668
669     return NGX_OK;
670 }
```

One Level Up Top Level

src/http/modules/ngx_http_limit_req_module.c - nginx-1.7.10

Global variables defined

- ngx_http_limit_req_commands
- ngx_http_limit_req_log_levels
- ngx_http_limit_req_module
- ngx_http_limit_req_module_ctx
- ngx_http_limit_req_status_bounds

Data types defined

- ngx_http_limit_req_conf_t
- ngx_http_limit_req_ctx_t
- ngx_http_limit_req_limit_t
- ngx_http_limit_req_node_t
- ngx_http_limit_req_shctx_t

Functions defined

- ngx_http_limit_req
- ngx_http_limit_req_account
- ngx_http_limit_req_create_conf
- ngx_http_limit_req_delay
- ngx_http_limit_req_expire
- ngx_http_limit_req_handler
- ngx_http_limit_req_init
- ngx_http_limit_req_init_zone
- ngx_http_limit_req_lookup
- ngx_http_limit_req_merge_conf
- ngx_http_limit_req_rbtree_insert_value
- ngx_http_limit_req_zone

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
```

```c
#include <ngx_core.h>
#include <ngx_http.h>


typedef struct {
    u_char                       color;
    u_char                       dummy;
    u_short                      len;
    ngx_queue_t                  queue;
    ngx_msec_t                   last;
    /* integer value, 1 corresponds to 0.001 r/s */
    ngx_uint_t                   excess;
    ngx_uint_t                   count;
    u_char                       data[1];
} ngx_http_limit_req_node_t;


typedef struct {
    ngx_rbtree_t                  rbtree;
    ngx_rbtree_node_t             sentinel;
    ngx_queue_t                   queue;
} ngx_http_limit_req_shctx_t;


typedef struct {
    ngx_http_limit_req_shctx_t  *sh;
    ngx_slab_pool_t             *shpool;
    /* integer value, 1 corresponds to 0.001 r/s */
    ngx_uint_t                   rate;
    ngx_http_complex_value_t     key;
    ngx_http_limit_req_node_t   *node;
} ngx_http_limit_req_ctx_t;


typedef struct {
    ngx_shm_zone_t              *shm_zone;
    /* integer value, 1 corresponds to 0.001 r/s */
    ngx_uint_t                   burst;
    ngx_uint_t                   nodelay; /* unsigned  nodelay:1 */
} ngx_http_limit_req_limit_t;


typedef struct {
    ngx_array_t                  limits;
    ngx_uint_t                   limit_log_level;
    ngx_uint_t                   delay_log_level;
    ngx_uint_t                   status_code;
} ngx_http_limit_req_conf_t;


static void ngx_http_limit_req_delay(ngx_http_request_t *r);
static ngx_int_t ngx_http_limit_req_lookup(ngx_http_limit_req_limit_t *limit,
    ngx_uint_t hash, ngx_str_t *key, ngx_uint_t *ep, ngx_uint_t account);
static ngx_msec_t ngx_http_limit_req_account(ngx_http_limit_req_limit_t *limits,
    ngx_uint_t n, ngx_uint_t *ep, ngx_http_limit_req_limit_t **limit);
static void ngx_http_limit_req_expire(ngx_http_limit_req_ctx_t *ctx,
    ngx_uint_t n);

static void *ngx_http_limit_req_create_conf(ngx_conf_t *cf);
static char *ngx_http_limit_req_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);
static char *ngx_http_limit_req_zone(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_limit_req(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static ngx_int_t ngx_http_limit_req_init(ngx_conf_t *cf);


static ngx_conf_enum_t  ngx_http_limit_req_log_levels[] = {
    { ngx_string("info"), NGX_LOG_INFO },
    { ngx_string("notice"), NGX_LOG_NOTICE },
    { ngx_string("warn"), NGX_LOG_WARN },
    { ngx_string("error"), NGX_LOG_ERR },
    { ngx_null_string, 0 }
};

```

```
85
86  static ngx_conf_num_bounds_t  ngx_http_limit_req_status_bounds = {
87      ngx_conf_check_num_bounds, 400, 599
88  };
89
90
91  static ngx_command_t  ngx_http_limit_req_commands[] = {
92
93      { ngx_string("limit_req_zone"),
94        NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE3,
95        ngx_http_limit_req_zone,
96        0,
97        0,
98        NULL },
99
100     { ngx_string("limit_req"),
101       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
102       ngx_http_limit_req,
103       NGX_HTTP_LOC_CONF_OFFSET,
104       0,
105       NULL },
106
107     { ngx_string("limit_req_log_level"),
108       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
109       ngx_conf_set_enum_slot,
110       NGX_HTTP_LOC_CONF_OFFSET,
111       offsetof(ngx_http_limit_req_conf_t, limit_log_level),
112       &ngx_http_limit_req_log_levels },
113
114     { ngx_string("limit_req_status"),
115       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
116       ngx_conf_set_num_slot,
117       NGX_HTTP_LOC_CONF_OFFSET,
118       offsetof(ngx_http_limit_req_conf_t, status_code),
119       &ngx_http_limit_req_status_bounds },
120
121       ngx_null_command
122  };
123
124
125  static ngx_http_module_t  ngx_http_limit_req_module_ctx = {
126      NULL,                                  /* preconfiguration */
127      ngx_http_limit_req_init,               /* postconfiguration */
128
129      NULL,                                  /* create main configuration */
130      NULL,                                  /* init main configuration */
131
132      NULL,                                  /* create server configuration */
133      NULL,                                  /* merge server configuration */
134
135      ngx_http_limit_req_create_conf,        /* create location configuration */
136      ngx_http_limit_req_merge_conf          /* merge location configuration */
137  };
138
139
140  ngx_module_t  ngx_http_limit_req_module = {
141      NGX_MODULE_V1,
142      &ngx_http_limit_req_module_ctx,        /* module context */
143      ngx_http_limit_req_commands,           /* module directives */
144      NGX_HTTP_MODULE,                       /* module type */
145      NULL,                                  /* init master */
146      NULL,                                  /* init module */
147      NULL,                                  /* init process */
148      NULL,                                  /* init thread */
149      NULL,                                  /* exit thread */
150      NULL,                                  /* exit process */
151      NULL,                                  /* exit master */
152      NGX_MODULE_V1_PADDING
153  };
154
155
156  static ngx_int_t
157  ngx_http_limit_req_handler(ngx_http_request_t *r)
158  {
159      uint32_t                    hash;
160      ngx_str_t                   key;
```

```
    ngx_int_t                   rc;
    ngx_uint_t                  n, excess;
    ngx_msec_t                  delay;
    ngx_http_limit_req_ctx_t    *ctx;
    ngx_http_limit_req_conf_t   *lrcf;
    ngx_http_limit_req_limit_t  *limit, *limits;

    if (r->main->limit_req_set) {
        return NGX_DECLINED;
    }

    lrcf = ngx_http_get_module_loc_conf(r, ngx_http_limit_req_module);
    limits = lrcf->limits.elts;

    excess = 0;

    rc = NGX_DECLINED;

#if (NGX_SUPPRESS_WARN)
    limit = NULL;
#endif

    for (n = 0; n < lrcf->limits.nelts; n++) {

        limit = &limits[n];

        ctx = limit->shm_zone->data;

        if (ngx_http_complex_value(r, &ctx->key, &key) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        if (key.len == 0) {
            continue;
        }

        if (key.len > 65535) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "the value of the \"%V\" key "
                          "is more than 65535 bytes: \"%V\"",
                          &ctx->key.value, &key);
            continue;
        }

        hash = ngx_crc32_short(key.data, key.len);

        ngx_shmtx_lock(&ctx->shpool->mutex);

        rc = ngx_http_limit_req_lookup(limit, hash, &key, &excess,
                                       (n == lrcf->limits.nelts - 1));

        ngx_shmtx_unlock(&ctx->shpool->mutex);

        ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "limit_req[%ui]: %i %ui.%03ui",
                       n, rc, excess / 1000, excess % 1000);

        if (rc != NGX_AGAIN) {
            break;
        }
    }

    if (rc == NGX_DECLINED) {
        return NGX_DECLINED;
    }

    r->main->limit_req_set = 1;

    if (rc == NGX_BUSY || rc == NGX_ERROR) {

        if (rc == NGX_BUSY) {
            ngx_log_error(lrcf->limit_log_level, r->connection->log, 0,
                          "limiting requests, excess: %ui.%03ui by zone \"%V\"",
                          excess / 1000, excess % 1000,
                          &limit->shm_zone->shm.name);
        }
```

```
237
238         while (n--) {
239             ctx = limits[n].shm_zone->data;
240
241             if (ctx->node == NULL) {
242                 continue;
243             }
244
245             ngx_shmtx_lock(&ctx->shpool->mutex);
246
247             ctx->node->count--;
248
249             ngx_shmtx_unlock(&ctx->shpool->mutex);
250
251             ctx->node = NULL;
252         }
253
254         return lrcf->status_code;
255     }
256
257     /* rc == NGX_AGAIN || rc == NGX_OK */
258
259     if (rc == NGX_AGAIN) {
260         excess = 0;
261     }
262
263     delay = ngx_http_limit_req_account(limits, n, &excess, &limit);
264
265     if (!delay) {
266         return NGX_DECLINED;
267     }
268
269     ngx_log_error(lrcf->delay_log_level, r->connection->log, 0,
270                   "delaying request, excess: %ui.%03ui, by zone \"%V\"",
271                   excess / 1000, excess % 1000, &limit->shm_zone->shm.name);
272
273     if (ngx_handle_read_event(r->connection->read, 0) != NGX_OK) {
274         return NGX_HTTP_INTERNAL_SERVER_ERROR;
275     }
276
277     r->read_event_handler = ngx_http_test_reading;
278     r->write_event_handler = ngx_http_limit_req_delay;
279     ngx_add_timer(r->connection->write, delay);
280
281     return NGX_AGAIN;
282 }
283
284
285 static void
286 ngx_http_limit_req_delay(ngx_http_request_t *r)
287 {
288     ngx_event_t  *wev;
289
290     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
291                    "limit_req delay");
292
293     wev = r->connection->write;
294
295     if (!wev->timedout) {
296
297         if (ngx_handle_write_event(wev, 0) != NGX_OK) {
298             ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
299         }
300
301         return;
302     }
303
304     wev->timedout = 0;
305
306     if (ngx_handle_read_event(r->connection->read, 0) != NGX_OK) {
307         ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
308         return;
309     }
310
311     r->read_event_handler = ngx_http_block_reading;
312     r->write_event_handler = ngx_http_core_run_phases;
```

```
313
314     ngx_http_core_run_phases(r);
315 }
316
317
318 static void
319 ngx_http_limit_req_rbtree_insert_value(ngx_rbtree_node_t *temp,
320     ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel)
321 {
322     ngx_rbtree_node_t          **p;
323     ngx_http_limit_req_node_t   *lrn, *lrnt;
324
325     for ( ;; ) {
326
327         if (node->key < temp->key) {
328
329             p = &temp->left;
330
331         } else if (node->key > temp->key) {
332
333             p = &temp->right;
334
335         } else { /* node->key == temp->key */
336
337             lrn = (ngx_http_limit_req_node_t *) &node->color;
338             lrnt = (ngx_http_limit_req_node_t *) &temp->color;
339
340             p = (ngx_memn2cmp(lrn->data, lrnt->data, lrn->len, lrnt->len) < 0)
341                 ? &temp->left : &temp->right;
342         }
343
344         if (*p == sentinel) {
345             break;
346         }
347
348         temp = *p;
349     }
350
351     *p = node;
352     node->parent = temp;
353     node->left = sentinel;
354     node->right = sentinel;
355     ngx_rbt_red(node);
356 }
357
358
359 static ngx_int_t
360 ngx_http_limit_req_lookup(ngx_http_limit_req_limit_t *limit, ngx_uint_t hash,
361     ngx_str_t *key, ngx_uint_t *ep, ngx_uint_t account)
362 {
363     size_t                      size;
364     ngx_int_t                   rc, excess;
365     ngx_time_t                 *tp;
366     ngx_msec_t                  now;
367     ngx_msec_int_t              ms;
368     ngx_rbtree_node_t          *node, *sentinel;
369     ngx_http_limit_req_ctx_t   *ctx;
370     ngx_http_limit_req_node_t  *lr;
371
372     tp = ngx_timeofday();
373     now = (ngx_msec_t) (tp->sec * 1000 + tp->msec);
374
375     ctx = limit->shm_zone->data;
376
377     node = ctx->sh->rbtree.root;
378     sentinel = ctx->sh->rbtree.sentinel;
379
380     while (node != sentinel) {
381
382         if (hash < node->key) {
383             node = node->left;
384             continue;
385         }
386
387         if (hash > node->key) {
388             node = node->right;
```

```
            continue;
        }

        /* hash == node->key */

        lr = (ngx_http_limit_req_node_t *) &node->color;

        rc = ngx_memn2cmp(key->data, lr->data, key->len, (size_t) lr->len);

        if (rc == 0) {
            ngx_queue_remove(&lr->queue);
            ngx_queue_insert_head(&ctx->sh->queue, &lr->queue);

            ms = (ngx_msec_int_t) (now - lr->last);

            excess = lr->excess - ctx->rate * ngx_abs(ms) / 1000 + 1000;

            if (excess < 0) {
                excess = 0;
            }

            *ep = excess;

            if ((ngx_uint_t) excess > limit->burst) {
                return NGX_BUSY;
            }

            if (account) {
                lr->excess = excess;
                lr->last = now;
                return NGX_OK;
            }

            lr->count++;

            ctx->node = lr;

            return NGX_AGAIN;
        }

        node = (rc < 0) ? node->left : node->right;
    }

    *ep = 0;

    size = offsetof(ngx_rbtree_node_t, color)
           + offsetof(ngx_http_limit_req_node_t, data)
           + key->len;

    ngx_http_limit_req_expire(ctx, 1);

    node = ngx_slab_alloc_locked(ctx->shpool, size);

    if (node == NULL) {
        ngx_http_limit_req_expire(ctx, 0);

        node = ngx_slab_alloc_locked(ctx->shpool, size);
        if (node == NULL) {
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, 0,
                          "could not allocate node%s", ctx->shpool->log_ctx);
            return NGX_ERROR;
        }
    }

    node->key = hash;

    lr = (ngx_http_limit_req_node_t *) &node->color;

    lr->len = (u_short) key->len;
    lr->excess = 0;

    ngx_memcpy(lr->data, key->data, key->len);

    ngx_rbtree_insert(&ctx->sh->rbtree, node);

    ngx_queue_insert_head(&ctx->sh->queue, &lr->queue);
```

```
465
466     if (account) {
467         lr->last = now;
468         lr->count = 0;
469         return NGX_OK;
470     }
471
472     lr->last = 0;
473     lr->count = 1;
474
475     ctx->node = lr;
476
477     return NGX_AGAIN;
478 }
479
480
481 static ngx_msec_t
482 ngx_http_limit_req_account(ngx_http_limit_req_limit_t *limits, ngx_uint_t n,
483     ngx_uint_t *ep, ngx_http_limit_req_limit_t **limit)
484 {
485     ngx_int_t                   excess;
486     ngx_time_t                 *tp;
487     ngx_msec_t                  now, delay, max_delay;
488     ngx_msec_int_t              ms;
489     ngx_http_limit_req_ctx_t   *ctx;
490     ngx_http_limit_req_node_t  *lr;
491
492     excess = *ep;
493
494     if (excess == 0 || (*limit)->nodelay) {
495         max_delay = 0;
496
497     } else {
498         ctx = (*limit)->shm_zone->data;
499         max_delay = excess * 1000 / ctx->rate;
500     }
501
502     while (n--) {
503         ctx = limits[n].shm_zone->data;
504         lr = ctx->node;
505
506         if (lr == NULL) {
507             continue;
508         }
509
510         ngx_shmtx_lock(&ctx->shpool->mutex);
511
512         tp = ngx_timeofday();
513
514         now = (ngx_msec_t) (tp->sec * 1000 + tp->msec);
515         ms = (ngx_msec_int_t) (now - lr->last);
516
517         excess = lr->excess - ctx->rate * ngx_abs(ms) / 1000 + 1000;
518
519         if (excess < 0) {
520             excess = 0;
521         }
522
523         lr->last = now;
524         lr->excess = excess;
525         lr->count--;
526
527         ngx_shmtx_unlock(&ctx->shpool->mutex);
528
529         ctx->node = NULL;
530
531         if (limits[n].nodelay) {
532             continue;
533         }
534
535         delay = excess * 1000 / ctx->rate;
536
537         if (delay > max_delay) {
538             max_delay = delay;
539             *ep = excess;
540             *limit = &limits[n];
```

```
        }
    }

    return max_delay;
}


static void
ngx_http_limit_req_expire(ngx_http_limit_req_ctx_t *ctx, ngx_uint_t n)
{
    ngx_int_t                   excess;
    ngx_time_t                 *tp;
    ngx_msec_t                  now;
    ngx_queue_t                *q;
    ngx_msec_int_t              ms;
    ngx_rbtree_node_t          *node;
    ngx_http_limit_req_node_t  *lr;

    tp = ngx_timeofday();

    now = (ngx_msec_t) (tp->sec * 1000 + tp->msec);

    /*
     * n == 1 deletes one or two zero rate entries
     * n == 0 deletes oldest entry by force
     *        and one or two zero rate entries
     */

    while (n < 3) {

        if (ngx_queue_empty(&ctx->sh->queue)) {
            return;
        }

        q = ngx_queue_last(&ctx->sh->queue);

        lr = ngx_queue_data(q, ngx_http_limit_req_node_t, queue);

        if (lr->count) {

            /*
             * There is not much sense in looking further,
             * because we bump nodes on the lookup stage.
             */

            return;
        }

        if (n++ != 0) {

            ms = (ngx_msec_int_t) (now - lr->last);
            ms = ngx_abs(ms);

            if (ms < 60000) {
                return;
            }

            excess = lr->excess - ctx->rate * ms / 1000;

            if (excess > 0) {
                return;
            }
        }

        ngx_queue_remove(q);

        node = (ngx_rbtree_node_t *)
                   ((u_char *) lr - offsetof(ngx_rbtree_node_t, color));

        ngx_rbtree_delete(&ctx->sh->rbtree, node);

        ngx_slab_free_locked(ctx->shpool, node);
    }
}
```

```
static ngx_int_t
ngx_http_limit_req_init_zone(ngx_shm_zone_t *shm_zone, void *data)
{
    ngx_http_limit_req_ctx_t  *octx = data;

    size_t                     len;
    ngx_http_limit_req_ctx_t  *ctx;

    ctx = shm_zone->data;

    if (octx) {
        if (ctx->key.value.len != octx->key.value.len
            || ngx_strncmp(ctx->key.value.data, octx->key.value.data,
                           ctx->key.value.len)
               != 0)
        {
            ngx_log_error(NGX_LOG_EMERG, shm_zone->shm.log, 0,
                          "limit_req \"%V\" uses the \"%V\" key "
                          "while previously it used the \"%V\" key",
                          &shm_zone->shm.name, &ctx->key.value,
                          &octx->key.value);
            return NGX_ERROR;
        }

        ctx->sh = octx->sh;
        ctx->shpool = octx->shpool;

        return NGX_OK;
    }

    ctx->shpool = (ngx_slab_pool_t *) shm_zone->shm.addr;

    if (shm_zone->shm.exists) {
        ctx->sh = ctx->shpool->data;

        return NGX_OK;
    }

    ctx->sh = ngx_slab_alloc(ctx->shpool, sizeof(ngx_http_limit_req_shctx_t));
    if (ctx->sh == NULL) {
        return NGX_ERROR;
    }

    ctx->shpool->data = ctx->sh;

    ngx_rbtree_init(&ctx->sh->rbtree, &ctx->sh->sentinel,
                    ngx_http_limit_req_rbtree_insert_value);

    ngx_queue_init(&ctx->sh->queue);

    len = sizeof(" in limit_req zone \"\"") + shm_zone->shm.name.len;

    ctx->shpool->log_ctx = ngx_slab_alloc(ctx->shpool, len);
    if (ctx->shpool->log_ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_sprintf(ctx->shpool->log_ctx, " in limit_req zone \"%V\"%Z",
                &shm_zone->shm.name);

    ctx->shpool->log_nomem = 0;

    return NGX_OK;
}


static void *
ngx_http_limit_req_create_conf(ngx_conf_t *cf)
{
    ngx_http_limit_req_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_limit_req_conf_t));
    if (conf == NULL) {
        return NULL;
    }

```

```
    /*
     * set by ngx_pcalloc():
     *
     *     conf->limits.elts = NULL;
     */

    conf->limit_log_level = NGX_CONF_UNSET_UINT;
    conf->status_code = NGX_CONF_UNSET_UINT;

    return conf;
}


static char *
ngx_http_limit_req_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_limit_req_conf_t *prev = parent;
    ngx_http_limit_req_conf_t *conf = child;

    if (conf->limits.elts == NULL) {
        conf->limits = prev->limits;
    }

    ngx_conf_merge_uint_value(conf->limit_log_level, prev->limit_log_level,
                              NGX_LOG_ERR);

    conf->delay_log_level = (conf->limit_log_level == NGX_LOG_INFO) ?
                                NGX_LOG_INFO : conf->limit_log_level + 1;

    ngx_conf_merge_uint_value(conf->status_code, prev->status_code,
                              NGX_HTTP_SERVICE_UNAVAILABLE);

    return NGX_CONF_OK;
}


static char *
ngx_http_limit_req_zone(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    u_char                           *p;
    size_t                            len;
    ssize_t                           size;
    ngx_str_t                        *value, name, s;
    ngx_int_t                         rate, scale;
    ngx_uint_t                        i;
    ngx_shm_zone_t                   *shm_zone;
    ngx_http_limit_req_ctx_t         *ctx;
    ngx_http_compile_complex_value_t  ccv;

    value = cf->args->elts;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_limit_req_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &ctx->key;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    size = 0;
    rate = 1;
    scale = 1;
    name.len = 0;

    for (i = 2; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "zone=", 5) == 0) {

            name.data = value[i].data + 5;
```

```
769
770             p = (u_char *) ngx_strchr(name.data, ':');
771
772             if (p == NULL) {
773                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
774                                    "invalid zone size \"%V\"", &value[i]);
775                 return NGX_CONF_ERROR;
776             }
777
778             name.len = p - name.data;
779
780             s.data = p + 1;
781             s.len = value[i].data + value[i].len - s.data;
782
783             size = ngx_parse_size(&s);
784
785             if (size == NGX_ERROR) {
786                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
787                                    "invalid zone size \"%V\"", &value[i]);
788                 return NGX_CONF_ERROR;
789             }
790
791             if (size < (ssize_t) (8 * ngx_pagesize)) {
792                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
793                                    "zone \"%V\" is too small", &value[i]);
794                 return NGX_CONF_ERROR;
795             }
796
797             continue;
798         }
799
800         if (ngx_strncmp(value[i].data, "rate=", 5) == 0) {
801
802             len = value[i].len;
803             p = value[i].data + len - 3;
804
805             if (ngx_strncmp(p, "r/s", 3) == 0) {
806                 scale = 1;
807                 len -= 3;
808
809             } else if (ngx_strncmp(p, "r/m", 3) == 0) {
810                 scale = 60;
811                 len -= 3;
812             }
813
814             rate = ngx_atoi(value[i].data + 5, len - 5);
815             if (rate <= 0) {
816                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
817                                    "invalid rate \"%V\"", &value[i]);
818                 return NGX_CONF_ERROR;
819             }
820
821             continue;
822         }
823
824         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
825                            "invalid parameter \"%V\"", &value[i]);
826         return NGX_CONF_ERROR;
827     }
828
829     if (name.len == 0) {
830         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
831                            "\"%V\" must have \"zone\" parameter",
832                            &cmd->name);
833         return NGX_CONF_ERROR;
834     }
835
836     ctx->rate = rate * 1000 / scale;
837
838     shm_zone = ngx_shared_memory_add(cf, &name, size,
839                                      &ngx_http_limit_req_module);
840     if (shm_zone == NULL) {
841         return NGX_CONF_ERROR;
842     }
843
844     if (shm_zone->data) {
```

```
        ctx = shm_zone->data;

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "%V \"%V\" is already bound to key \"%V\"",
                           &cmd->name, &name, &ctx->key.value);
        return NGX_CONF_ERROR;
    }

    shm_zone->init = ngx_http_limit_req_init_zone;
    shm_zone->data = ctx;

    return NGX_CONF_OK;
}


static char *
ngx_http_limit_req(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_limit_req_conf_t  *lrcf = conf;

    ngx_int_t                   burst;
    ngx_str_t                  *value, s;
    ngx_uint_t                  i, nodelay;
    ngx_shm_zone_t             *shm_zone;
    ngx_http_limit_req_limit_t  *limit, *limits;

    value = cf->args->elts;

    shm_zone = NULL;
    burst = 0;
    nodelay = 0;

    for (i = 1; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "zone=", 5) == 0) {

            s.len = value[i].len - 5;
            s.data = value[i].data + 5;

            shm_zone = ngx_shared_memory_add(cf, &s, 0,
                                             &ngx_http_limit_req_module);
            if (shm_zone == NULL) {
                return NGX_CONF_ERROR;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "burst=", 6) == 0) {

            burst = ngx_atoi(value[i].data + 6, value[i].len - 6);
            if (burst <= 0) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid burst rate \"%V\"", &value[i]);
                return NGX_CONF_ERROR;
            }

            continue;
        }

        if (ngx_strcmp(value[i].data, "nodelay") == 0) {
            nodelay = 1;
            continue;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid parameter \"%V\"", &value[i]);
        return NGX_CONF_ERROR;
    }

    if (shm_zone == NULL) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"%V\" must have \"zone\" parameter",
                           &cmd->name);
        return NGX_CONF_ERROR;
    }
```

```
921
922     if (shm_zone->data == NULL) {
923         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
924                            "unknown limit_req_zone \"%V\"",
925                            &shm_zone->shm.name);
926         return NGX_CONF_ERROR;
927     }
928
929     limits = lrcf->limits.elts;
930
931     if (limits == NULL) {
932         if (ngx_array_init(&lrcf->limits, cf->pool, 1,
933                            sizeof(ngx_http_limit_req_limit_t))
934             != NGX_OK)
935         {
936             return NGX_CONF_ERROR;
937         }
938     }
939
940     for (i = 0; i < lrcf->limits.nelts; i++) {
941         if (shm_zone == limits[i].shm_zone) {
942             return "is duplicate";
943         }
944     }
945
946     limit = ngx_array_push(&lrcf->limits);
947     if (limit == NULL) {
948         return NGX_CONF_ERROR;
949     }
950
951     limit->shm_zone = shm_zone;
952     limit->burst = burst * 1000;
953     limit->nodelay = nodelay;
954
955     return NGX_CONF_OK;
956 }
957
958
959 static ngx_int_t
960 ngx_http_limit_req_init(ngx_conf_t *cf)
961 {
962     ngx_http_handler_pt        *h;
963     ngx_http_core_main_conf_t  *cmcf;
964
965     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
966
967     h = ngx_array_push(&cmcf->phases[NGX_HTTP_PREACCESS_PHASE].handlers);
968     if (h == NULL) {
969         return NGX_ERROR;
970     }
971
972     *h = ngx_http_limit_req_handler;
973
974     return NGX_OK;
975 }
```

One Level Up Top Level

src/http/ngx_http_busy_lock.c - nginx-1.7.10

Functions defined

- ngx_http_busy_lock
- ngx_http_busy_lock_cacheable
- ngx_http_busy_lock_look_cacheable
- ngx_http_busy_unlock
- ngx_http_set_busy_lock_slot

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13
14  static int ngx_http_busy_lock_look_cacheable(ngx_http_busy_lock_t *bl,
15                                               ngx_http_busy_lock_ctx_t *bc,
16                                               int lock);
17
18
19  int ngx_http_busy_lock(ngx_http_busy_lock_t *bl, ngx_http_busy_lock_ctx_t *bc)
20  {
21      if (bl->busy < bl->max_busy) {
22          bl->busy++;
23
24          if (bc->time) {
25              bc->time = 0;
26              bl->waiting--;
27          }
28
29          return NGX_OK;
30      }
31
32      if (bc->time) {
33          if (bc->time < bl->timeout) {
34              ngx_add_timer(bc->event, 1000);
35              return NGX_AGAIN;
36          }
37
38          bl->waiting--;
39          return NGX_DONE;
40
41      }
42
43      if (bl->timeout == 0) {
44          return NGX_DONE;
45      }
46
47      if (bl->waiting < bl->max_waiting) {
48          bl->waiting++;
49
50  #if 0
51          ngx_add_timer(bc->event, 1000);
52          bc->event->event_handler = bc->event_handler;
53  #endif
```

```
54
55          /* TODO: ngx_handle_level_read_event() */
56
57          return NGX_AGAIN;
58      }
59
60      return NGX_ERROR;
61  }
62
63
64  int ngx_http_busy_lock_cacheable(ngx_http_busy_lock_t *bl,
65                                   ngx_http_busy_lock_ctx_t *bc, int lock)
66  {
67      int  rc;
68
69      rc = ngx_http_busy_lock_look_cacheable(bl, bc, lock);
70
71      ngx_log_debug3(NGX_LOG_DEBUG_HTTP, bc->event->log, 0,
72                     "http busylock: %d w:%d mw::%d",
73                     rc, bl->waiting, bl->max_waiting);
74
75      if (rc == NGX_OK) {  /* no the same request, there's free slot */
76          return NGX_OK;
77      }
78
79      if (rc == NGX_ERROR && !lock) { /* no the same request, no free slot */
80          return NGX_OK;
81      }
82
83      /* rc == NGX_AGAIN:  the same request */
84
85      if (bc->time) {
86          if (bc->time < bl->timeout) {
87              ngx_add_timer(bc->event, 1000);
88              return NGX_AGAIN;
89          }
90
91          bl->waiting--;
92          return NGX_DONE;
93
94      }
95
96      if (bl->timeout == 0) {
97          return NGX_DONE;
98      }
99
100      if (bl->waiting < bl->max_waiting) {
101  #if 0
102          bl->waiting++;
103          ngx_add_timer(bc->event, 1000);
104          bc->event->event_handler = bc->event_handler;
105  #endif
106
107          /* TODO: ngx_handle_level_read_event() */
108
109          return NGX_AGAIN;
110      }
111
112      return NGX_ERROR;
113  }
114
115
116  void ngx_http_busy_unlock(ngx_http_busy_lock_t *bl,
117                            ngx_http_busy_lock_ctx_t *bc)
118  {
119      if (bl == NULL) {
120          return;
121      }
122
123      if (bl->md5) {
124          bl->md5_mask[bc->slot / 8] &= ~(1 << (bc->slot & 7));
125          bl->cacheable--;
126      }
127
128      bl->busy--;
129  }
```

```
130
131
132 static int ngx_http_busy_lock_look_cacheable(ngx_http_busy_lock_t *bl,
133                                              ngx_http_busy_lock_ctx_t *bc,
134                                              int lock)
135 {
136     int    i, b, cacheable, free;
137     u_int  mask;
138
139     b = 0;
140     cacheable = 0;
141     free = -1;
142
143 #if (NGX_SUPPRESS_WARN)
144     mask = 0;
145 #endif
146
147     for (i = 0; i < bl->max_busy; i++) {
148
149         if ((b & 7) == 0) {
150             mask = bl->md5_mask[i / 8];
151         }
152
153         if (mask & 1) {
154             if (ngx_memcmp(&bl->md5[i * 16], bc->md5, 16) == 0) {
155                 return NGX_AGAIN;
156             }
157             cacheable++;
158
159         } else if (free == -1) {
160             free = i;
161         }
162
163 #if 1
164         if (cacheable == bl->cacheable) {
165             if (free == -1 && cacheable < bl->max_busy) {
166                 free = i + 1;
167             }
168
169             break;
170         }
171 #endif
172
173         mask >>= 1;
174         b++;
175     }
176
177     if (free == -1) {
178         return NGX_ERROR;
179     }
180
181     if (lock) {
182         if (bl->busy == bl->max_busy) {
183             return NGX_ERROR;
184         }
185
186         ngx_memcpy(&bl->md5[free * 16], bc->md5, 16);
187         bl->md5_mask[free / 8] |= 1 << (free & 7);
188         bc->slot = free;
189
190         bl->cacheable++;
191         bl->busy++;
192     }
193
194     return NGX_OK;
195 }
196
197
198 char *ngx_http_set_busy_lock_slot(ngx_conf_t *cf, ngx_command_t *cmd,
199                                   void *conf)
200 {
201     char  *p = conf;
202
203     ngx_uint_t             i, dup, invalid;
204     ngx_str_t             *value, line;
205     ngx_http_busy_lock_t  *bl, **blp;
```

```
206
207     blp = (ngx_http_busy_lock_t **) (p + cmd->offset);
208     if (*blp) {
209         return "is duplicate";
210     }
211
212     /* ngx_calloc_shared() */
213     bl = ngx_pcalloc(cf->pool, sizeof(ngx_http_busy_lock_t));
214     if (bl == NULL) {
215         return NGX_CONF_ERROR;
216     }
217     *blp = bl;
218
219     /* ngx_calloc_shared() */
220     bl->mutex = ngx_pcalloc(cf->pool, sizeof(ngx_event_mutex_t));
221     if (bl->mutex == NULL) {
222         return NGX_CONF_ERROR;
223     }
224
225     dup = 0;
226     invalid = 0;
227     value = cf->args->elts;
228
229     for (i = 1; i < cf->args->nelts; i++) {
230
231         if (value[i].data[1] != '=') {
232             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
233                                "invalid value \"%s\"", value[i].data);
234             return NGX_CONF_ERROR;
235         }
236
237         switch (value[i].data[0]) {
238
239         case 'b':
240             if (bl->max_busy) {
241                 dup = 1;
242                 break;
243             }
244
245             bl->max_busy = ngx_atoi(value[i].data + 2, value[i].len - 2);
246             if (bl->max_busy == NGX_ERROR) {
247                 invalid = 1;
248                 break;
249             }
250
251             continue;
252
253         case 'w':
254             if (bl->max_waiting) {
255                 dup = 1;
256                 break;
257             }
258
259             bl->max_waiting = ngx_atoi(value[i].data + 2, value[i].len - 2);
260             if (bl->max_waiting == NGX_ERROR) {
261                 invalid = 1;
262                 break;
263             }
264
265             continue;
266
267         case 't':
268             if (bl->timeout) {
269                 dup = 1;
270                 break;
271             }
272
273             line.len = value[i].len - 2;
274             line.data = value[i].data + 2;
275
276             bl->timeout = ngx_parse_time(&line, 1);
277             if (bl->timeout == (time_t) NGX_ERROR) {
278                 invalid = 1;
279                 break;
280             }
281
```

```
282             continue;
283
284         default:
285             invalid = 1;
286         }
287
288         if (dup) {
289             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
290                                "duplicate value \"%s\"", value[i].data);
291             return NGX_CONF_ERROR;
292         }
293
294         if (invalid) {
295             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
296                                "invalid value \"%s\"", value[i].data);
297             return NGX_CONF_ERROR;
298         }
299     }
300
301     if (bl->timeout == 0 && bl->max_waiting) {
302         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
303                    "busy lock waiting is useless with zero timeout, ignoring");
304     }
305
306     return NGX_CONF_OK;
307 }
```

One Level Up Top Level

src/http/ngx_http_busy_lock.h - nginx-1.7.10

Data types defined

- ngx_http_busy_lock_ctx_t
- ngx_http_busy_lock_t

Macros defined

- _NGX_HTTP_BUSY_LOCK_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_HTTP_BUSY_LOCK_H_INCLUDED_
9  #define _NGX_HTTP_BUSY_LOCK_H_INCLUDED_
10
11
12 #include <ngx_config.h>
13 #include <ngx_core.h>
14 #include <ngx_event.h>
15 #include <ngx_http.h>
16
17
18 typedef struct {
19     u_char             *md5_mask;
20     char               *md5;
21     int                 cacheable;
22
23     int                 busy;
24     int                 max_busy;
25
26     int                 waiting;
27     int                 max_waiting;
28
29     time_t              timeout;
30
31     ngx_event_mutex_t  *mutex;
32 } ngx_http_busy_lock_t;
33
34
35 typedef struct {
36     time_t         time;
37     ngx_event_t   *event;
38     void         (*event_handler)(ngx_event_t *ev);
39     u_char        *md5;
40     int            slot;
41 } ngx_http_busy_lock_ctx_t;
42
43
44 int ngx_http_busy_lock(ngx_http_busy_lock_t *bl, ngx_http_busy_lock_ctx_t *bc);
45 int ngx_http_busy_lock_cacheable(ngx_http_busy_lock_t *bl,
46                                  ngx_http_busy_lock_ctx_t *bc, int lock);
47 void ngx_http_busy_unlock(ngx_http_busy_lock_t *bl,
48                           ngx_http_busy_lock_ctx_t *bc);
49
50 char *ngx_http_set_busy_lock_slot(ngx_conf_t *cf, ngx_command_t *cmd,
51                                   void *conf);
52
53
54 #endif /* _NGX_HTTP_BUSY_LOCK_H_INCLUDED_ */
```

src/http/modules/ngx_http_ssi_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_body_filter
- ngx_http_next_header_filter
- ngx_http_ssi_block_params
- ngx_http_ssi_commands
- ngx_http_ssi_config_params
- ngx_http_ssi_echo_params
- ngx_http_ssi_filter_commands
- ngx_http_ssi_filter_module
- ngx_http_ssi_filter_module_ctx
- ngx_http_ssi_if_params
- ngx_http_ssi_include_params
- ngx_http_ssi_no_params
- ngx_http_ssi_none
- ngx_http_ssi_null_string
- ngx_http_ssi_set_params
- ngx_http_ssi_string
- ngx_http_ssi_timefmt
- ngx_http_ssi_vars

Data types defined

- ngx_http_ssi_block_t
- ngx_http_ssi_loc_conf_t
- ngx_http_ssi_state_e
- ngx_http_ssi_var_t

Functions defined

- ngx_http_ssi_block
- ngx_http_ssi_body_filter
- ngx_http_ssi_buffered
- ngx_http_ssi_config
- ngx_http_ssi_create_loc_conf

- ngx_http_ssi_create_main_conf
- ngx_http_ssi_date_gmt_local_variable
- ngx_http_ssi_echo
- ngx_http_ssi_else
- ngx_http_ssi_endblock
- ngx_http_ssi_endif
- ngx_http_ssi_evaluate_string
- ngx_http_ssi_filter_init
- ngx_http_ssi_get_variable
- ngx_http_ssi_header_filter
- ngx_http_ssi_if
- ngx_http_ssi_include
- ngx_http_ssi_init_main_conf
- ngx_http_ssi_merge_loc_conf
- ngx_http_ssi_output
- ngx_http_ssi_parse
- ngx_http_ssi_preconfiguration
- ngx_http_ssi_regex_match
- ngx_http_ssi_set
- ngx_http_ssi_set_variable
- ngx_http_ssi_stub_output

Macros defined

- NGX_HTTP_SSI_ADD_PREFIX
- NGX_HTTP_SSI_ADD_ZERO
- NGX_HTTP_SSI_BLOCK_NAME
- NGX_HTTP_SSI_CONFIG_ERRMSG
- NGX_HTTP_SSI_CONFIG_TIMEFMT
- NGX_HTTP_SSI_DATE_LEN
- NGX_HTTP_SSI_ECHO_DEFAULT
- NGX_HTTP_SSI_ECHO_ENCODING
- NGX_HTTP_SSI_ECHO_VAR
- NGX_HTTP_SSI_ERROR
- NGX_HTTP_SSI_IF_EXPR

- NGX_HTTP_SSI_INCLUDE_FILE
- NGX_HTTP_SSI_INCLUDE_SET
- NGX_HTTP_SSI_INCLUDE_STUB
- NGX_HTTP_SSI_INCLUDE_VIRTUAL
- NGX_HTTP_SSI_INCLUDE_WAIT
- NGX_HTTP_SSI_SET_VALUE
- NGX_HTTP_SSI_SET_VAR

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12  #define NGX_HTTP_SSI_ERROR          1
13
14  #define NGX_HTTP_SSI_DATE_LEN       2048
15
16  #define NGX_HTTP_SSI_ADD_PREFIX     1
17  #define NGX_HTTP_SSI_ADD_ZERO       2
18
19
20  typedef struct {
21      ngx_flag_t    enable;
22      ngx_flag_t    silent_errors;
23      ngx_flag_t    ignore_recycled_buffers;
24      ngx_flag_t    last_modified;
25
26      ngx_hash_t    types;
27
28      size_t        min_file_chunk;
29      size_t        value_len;
30
31      ngx_array_t  *types_keys;
32  } ngx_http_ssi_loc_conf_t;
33
34
35  typedef struct {
36      ngx_str_t     name;
37      ngx_uint_t    key;
38      ngx_str_t     value;
39  } ngx_http_ssi_var_t;
40
41
42  typedef struct {
43      ngx_str_t     name;
44      ngx_chain_t  *bufs;
45      ngx_uint_t    count;
46  } ngx_http_ssi_block_t;
47
48
49  typedef enum {
50      ssi_start_state = 0,
51      ssi_tag_state,
52      ssi_comment0_state,
53      ssi_comment1_state,
54      ssi_sharp_state,
55      ssi_precommand_state,
56      ssi_command_state,
```

```
    ssi_preparam_state,
    ssi_param_state,
    ssi_preequal_state,
    ssi_prevalue_state,
    ssi_double_quoted_value_state,
    ssi_quoted_value_state,
    ssi_quoted_symbol_state,
    ssi_postparam_state,
    ssi_comment_end0_state,
    ssi_comment_end1_state,
    ssi_error_state,
    ssi_error_end0_state,
    ssi_error_end1_state
} ngx_http_ssi_state_e;


static ngx_int_t ngx_http_ssi_output(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx);
static void ngx_http_ssi_buffered(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx);
static ngx_int_t ngx_http_ssi_parse(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx);
static ngx_str_t *ngx_http_ssi_get_variable(ngx_http_request_t *r,
    ngx_str_t *name, ngx_uint_t key);
static ngx_int_t ngx_http_ssi_evaluate_string(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t *text, ngx_uint_t flags);
static ngx_int_t ngx_http_ssi_regex_match(ngx_http_request_t *r,
    ngx_str_t *pattern, ngx_str_t *str);

static ngx_int_t ngx_http_ssi_include(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
static ngx_int_t ngx_http_ssi_stub_output(ngx_http_request_t *r, void *data,
    ngx_int_t rc);
static ngx_int_t ngx_http_ssi_set_variable(ngx_http_request_t *r, void *data,
    ngx_int_t rc);
static ngx_int_t ngx_http_ssi_echo(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
static ngx_int_t ngx_http_ssi_config(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
static ngx_int_t ngx_http_ssi_set(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
static ngx_int_t ngx_http_ssi_if(ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
 static ngx_int_t ngx_http_ssi_else(ngx_http_request_t *r,
     ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
 static ngx_int_t ngx_http_ssi_endif(ngx_http_request_t *r,
     ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
 static ngx_int_t ngx_http_ssi_block(ngx_http_request_t *r,
     ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);
 static ngx_int_t ngx_http_ssi_endblock(ngx_http_request_t *r,
     ngx_http_ssi_ctx_t *ctx, ngx_str_t **params);

 static ngx_int_t ngx_http_ssi_date_gmt_local_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t gmt);

 static ngx_int_t ngx_http_ssi_preconfiguration(ngx_conf_t *cf);
 static void *ngx_http_ssi_create_main_conf(ngx_conf_t *cf);
 static char *ngx_http_ssi_init_main_conf(ngx_conf_t *cf, void *conf);
 static void *ngx_http_ssi_create_loc_conf(ngx_conf_t *cf);
 static char *ngx_http_ssi_merge_loc_conf(ngx_conf_t *cf,
     void *parent, void *child);
 static ngx_int_t ngx_http_ssi_filter_init(ngx_conf_t *cf);


 static ngx_command_t  ngx_http_ssi_filter_commands[] = {

     { ngx_string("ssi"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                         |NGX_CONF_FLAG,
       ngx_conf_set_flag_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_ssi_loc_conf_t, enable),
       NULL },

     { ngx_string("ssi_silent_errors"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
```

```
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_ssi_loc_conf_t, silent_errors),
      NULL },

    { ngx_string("ssi_ignore_recycled_buffers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_ssi_loc_conf_t, ignore_recycled_buffers),
      NULL },

    { ngx_string("ssi_min_file_chunk"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_ssi_loc_conf_t, min_file_chunk),
      NULL },

    { ngx_string("ssi_value_length"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_ssi_loc_conf_t, value_len),
      NULL },

    { ngx_string("ssi_types"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_types_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_ssi_loc_conf_t, types_keys),
      &ngx_http_html_default_types[0] },

    { ngx_string("ssi_last_modified"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_ssi_loc_conf_t, last_modified),
      NULL },

      ngx_null_command
};



static ngx_http_module_t  ngx_http_ssi_filter_module_ctx = {
    ngx_http_ssi_preconfiguration,         /* preconfiguration */
    ngx_http_ssi_filter_init,              /* postconfiguration */

    ngx_http_ssi_create_main_conf,         /* create main configuration */
    ngx_http_ssi_init_main_conf,           /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_ssi_create_loc_conf,          /* create location configuration */
    ngx_http_ssi_merge_loc_conf            /* merge location configuration */
};


ngx_module_t  ngx_http_ssi_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_ssi_filter_module_ctx,       /* module context */
    ngx_http_ssi_filter_commands,          /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};
```

```
static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static u_char ngx_http_ssi_string[] = "<!--";

static ngx_str_t ngx_http_ssi_none = ngx_string("(none)");
static ngx_str_t ngx_http_ssi_timefmt = ngx_string("%A, %d-%b-%Y %H:%M:%S %Z");
static ngx_str_t ngx_http_ssi_null_string = ngx_null_string;


#define  NGX_HTTP_SSI_INCLUDE_VIRTUAL  0
#define  NGX_HTTP_SSI_INCLUDE_FILE     1
#define  NGX_HTTP_SSI_INCLUDE_WAIT     2
#define  NGX_HTTP_SSI_INCLUDE_SET      3
#define  NGX_HTTP_SSI_INCLUDE_STUB     4

#define  NGX_HTTP_SSI_ECHO_VAR         0
#define  NGX_HTTP_SSI_ECHO_DEFAULT     1
#define  NGX_HTTP_SSI_ECHO_ENCODING    2

#define  NGX_HTTP_SSI_CONFIG_ERRMSG    0
#define  NGX_HTTP_SSI_CONFIG_TIMEFMT   1

#define  NGX_HTTP_SSI_SET_VAR          0
#define  NGX_HTTP_SSI_SET_VALUE        1

#define  NGX_HTTP_SSI_IF_EXPR          0

#define  NGX_HTTP_SSI_BLOCK_NAME       0


static ngx_http_ssi_param_t  ngx_http_ssi_include_params[] = {
    { ngx_string("virtual"), NGX_HTTP_SSI_INCLUDE_VIRTUAL, 0, 0 },
    { ngx_string("file"), NGX_HTTP_SSI_INCLUDE_FILE, 0, 0 },
    { ngx_string("wait"), NGX_HTTP_SSI_INCLUDE_WAIT, 0, 0 },
    { ngx_string("set"), NGX_HTTP_SSI_INCLUDE_SET, 0, 0 },
    { ngx_string("stub"), NGX_HTTP_SSI_INCLUDE_STUB, 0, 0 },
    { ngx_null_string, 0, 0, 0 }
};


static ngx_http_ssi_param_t  ngx_http_ssi_echo_params[] = {
    { ngx_string("var"), NGX_HTTP_SSI_ECHO_VAR, 1, 0 },
    { ngx_string("default"), NGX_HTTP_SSI_ECHO_DEFAULT, 0, 0 },
    { ngx_string("encoding"), NGX_HTTP_SSI_ECHO_ENCODING, 0, 0 },
    { ngx_null_string, 0, 0, 0 }
};


static ngx_http_ssi_param_t  ngx_http_ssi_config_params[] = {
    { ngx_string("errmsg"), NGX_HTTP_SSI_CONFIG_ERRMSG, 0, 0 },
    { ngx_string("timefmt"), NGX_HTTP_SSI_CONFIG_TIMEFMT, 0, 0 },
    { ngx_null_string, 0, 0, 0 }
};


static ngx_http_ssi_param_t  ngx_http_ssi_set_params[] = {
    { ngx_string("var"), NGX_HTTP_SSI_SET_VAR, 1, 0 },
    { ngx_string("value"), NGX_HTTP_SSI_SET_VALUE, 1, 0 },
    { ngx_null_string, 0, 0, 0 }
};


static ngx_http_ssi_param_t  ngx_http_ssi_if_params[] = {
    { ngx_string("expr"), NGX_HTTP_SSI_IF_EXPR, 1, 0 },
    { ngx_null_string, 0, 0, 0 }
};


static ngx_http_ssi_param_t  ngx_http_ssi_block_params[] = {
    { ngx_string("name"), NGX_HTTP_SSI_BLOCK_NAME, 1, 0 },
    { ngx_null_string, 0, 0, 0 }
};
```

```
static ngx_http_ssi_param_t  ngx_http_ssi_no_params[] = {
    { ngx_null_string, 0, 0, 0 }
};


static ngx_http_ssi_command_t  ngx_http_ssi_commands[] = {
    { ngx_string("include"), ngx_http_ssi_include,
                      ngx_http_ssi_include_params, 0, 0, 1 },
    { ngx_string("echo"), ngx_http_ssi_echo,
                      ngx_http_ssi_echo_params, 0, 0, 0 },
    { ngx_string("config"), ngx_http_ssi_config,
                      ngx_http_ssi_config_params, 0, 0, 0 },
    { ngx_string("set"), ngx_http_ssi_set, ngx_http_ssi_set_params, 0, 0, 0 },

    { ngx_string("if"), ngx_http_ssi_if, ngx_http_ssi_if_params, 0, 0, 0 },
    { ngx_string("elif"), ngx_http_ssi_if, ngx_http_ssi_if_params,
                      NGX_HTTP_SSI_COND_IF, 0, 0 },
    { ngx_string("else"), ngx_http_ssi_else, ngx_http_ssi_no_params,
                      NGX_HTTP_SSI_COND_IF, 0, 0 },
    { ngx_string("endif"), ngx_http_ssi_endif, ngx_http_ssi_no_params,
                      NGX_HTTP_SSI_COND_ELSE, 0, 0 },

    { ngx_string("block"), ngx_http_ssi_block,
                      ngx_http_ssi_block_params, 0, 0, 0 },
    { ngx_string("endblock"), ngx_http_ssi_endblock,
                      ngx_http_ssi_no_params, 0, 1, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static ngx_http_variable_t  ngx_http_ssi_vars[] = {

    { ngx_string("date_local"), NULL, ngx_http_ssi_date_gmt_local_variable, 0,
      NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("date_gmt"), NULL, ngx_http_ssi_date_gmt_local_variable, 1,
      NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};



static ngx_int_t
ngx_http_ssi_header_filter(ngx_http_request_t *r)
{
    ngx_http_ssi_ctx_t       *ctx;
    ngx_http_ssi_loc_conf_t  *slcf;

    slcf = ngx_http_get_module_loc_conf(r, ngx_http_ssi_filter_module);

    if (!slcf->enable
        || r->headers_out.content_length_n == 0
        || ngx_http_test_content_type(r, &slcf->types) == NULL)
    {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_ssi_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_ssi_filter_module);


    ctx->value_len = slcf->value_len;
    ctx->last_out = &ctx->out;

    ctx->encoding = NGX_HTTP_SSI_ENTITY_ENCODING;
    ctx->output = 1;

    ctx->params.elts = ctx->params_array;
    ctx->params.size = sizeof(ngx_table_elt_t);
    ctx->params.nalloc = NGX_HTTP_SSI_PARAMS_N;
```

```
    ctx->params.pool = r->pool;

    ctx->timefmt = ngx_http_ssi_timefmt;
    ngx_str_set(&ctx->errmsg,
                "[an error occurred while processing the directive]");

    r->filter_need_in_memory = 1;

    if (r == r->main) {
        ngx_http_clear_content_length(r);
        ngx_http_clear_accept_ranges(r);

        if (!slcf->last_modified) {
            ngx_http_clear_last_modified(r);
            ngx_http_clear_etag(r);

        } else {
            ngx_http_weak_etag(r);
        }
    }

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_ssi_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    size_t                     len;
    ngx_int_t                  rc;
    ngx_buf_t                 *b;
    ngx_uint_t                 i, index;
    ngx_chain_t               *cl, **ll;
    ngx_table_elt_t           *param;
    ngx_http_ssi_ctx_t        *ctx, *mctx;
    ngx_http_ssi_block_t      *bl;
    ngx_http_ssi_param_t      *prm;
    ngx_http_ssi_command_t    *cmd;
    ngx_http_ssi_loc_conf_t   *slcf;
    ngx_http_ssi_main_conf_t  *smcf;
    ngx_str_t                 *params[NGX_HTTP_SSI_MAX_PARAMS + 1];

    ctx = ngx_http_get_module_ctx(r, ngx_http_ssi_filter_module);

    if (ctx == NULL
        || (in == NULL
            && ctx->buf == NULL
            && ctx->in == NULL
            && ctx->busy == NULL))
    {
        return ngx_http_next_body_filter(r, in);
    }

    /* add the incoming chain to the chain ctx->in */

    if (in) {
        if (ngx_chain_add_copy(r->pool, &ctx->in, in) != NGX_OK) {
            return NGX_ERROR;
        }
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http ssi filter \"%V?%V\"", &r->uri, &r->args);

    if (ctx->wait) {

        if (r != r->connection->data) {
            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http ssi filter wait \"%V?%V\" non-active",
                           &ctx->wait->uri, &ctx->wait->args);

            return NGX_AGAIN;
        }

        if (ctx->wait->done) {
            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
```

```
                          "http ssi filter wait \"%V?%V\" done",
                          &ctx->wait->uri, &ctx->wait->args);

            ctx->wait = NULL;

        } else {
            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http ssi filter wait \"%V?%V\"",
                           &ctx->wait->uri, &ctx->wait->args);

            return ngx_http_next_body_filter(r, NULL);
        }
    }

    slcf = ngx_http_get_module_loc_conf(r, ngx_http_ssi_filter_module);

    while (ctx->in || ctx->buf) {

        if (ctx->buf == NULL) {
            ctx->buf = ctx->in->buf;
            ctx->in = ctx->in->next;
            ctx->pos = ctx->buf->pos;
        }

        if (ctx->state == ssi_start_state) {
            ctx->copy_start = ctx->pos;
            ctx->copy_end = ctx->pos;
        }

        b = NULL;

        while (ctx->pos < ctx->buf->last) {

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "saved: %d state: %d", ctx->saved, ctx->state);

            rc = ngx_http_ssi_parse(r, ctx);

            ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "parse: %d, looked: %d %p-%p",
                           rc, ctx->looked, ctx->copy_start, ctx->copy_end);

            if (rc == NGX_ERROR) {
                return rc;
            }

            if (ctx->copy_start != ctx->copy_end) {

                if (ctx->output) {

                    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                                   "saved: %d", ctx->saved);

                    if (ctx->saved) {

                        if (ctx->free) {
                            cl = ctx->free;
                            ctx->free = ctx->free->next;
                            b = cl->buf;
                            ngx_memzero(b, sizeof(ngx_buf_t));

                        } else {
                            b = ngx_calloc_buf(r->pool);
                            if (b == NULL) {
                                return NGX_ERROR;
                            }

                            cl = ngx_alloc_chain_link(r->pool);
                            if (cl == NULL) {
                                return NGX_ERROR;
                            }

                            cl->buf = b;
                        }

                        b->memory = 1;
```

```
                    b->pos = ngx_http_ssi_string;
                    b->last = ngx_http_ssi_string + ctx->saved;

                    *ctx->last_out = cl;
                    ctx->last_out = &cl->next;

                    ctx->saved = 0;
                }

                if (ctx->free) {
                    cl = ctx->free;
                    ctx->free = ctx->free->next;
                    b = cl->buf;

                } else {
                    b = ngx_alloc_buf(r->pool);
                    if (b == NULL) {
                        return NGX_ERROR;
                    }

                    cl = ngx_alloc_chain_link(r->pool);
                    if (cl == NULL) {
                        return NGX_ERROR;
                    }

                    cl->buf = b;
                }

                ngx_memcpy(b, ctx->buf, sizeof(ngx_buf_t));

                b->pos = ctx->copy_start;
                b->last = ctx->copy_end;
                b->shadow = NULL;
                b->last_buf = 0;
                b->recycled = 0;

                if (b->in_file) {
                    if (slcf->min_file_chunk < (size_t) (b->last - b->pos))
                    {
                        b->file_last = b->file_pos
                                       + (b->last - ctx->buf->pos);
                        b->file_pos += b->pos - ctx->buf->pos;

                    } else {
                        b->in_file = 0;
                    }
                }

                cl->next = NULL;
                *ctx->last_out = cl;
                ctx->last_out = &cl->next;

            } else {
                if (ctx->block
                    && ctx->saved + (ctx->copy_end - ctx->copy_start))
                {
                    b = ngx_create_temp_buf(r->pool,
                           ctx->saved + (ctx->copy_end - ctx->copy_start));

                    if (b == NULL) {
                        return NGX_ERROR;
                    }

                    if (ctx->saved) {
                        b->last = ngx_cpymem(b->pos, ngx_http_ssi_string,
                                             ctx->saved);
                    }

                    b->last = ngx_cpymem(b->last, ctx->copy_start,
                                         ctx->copy_end - ctx->copy_start);

                    cl = ngx_alloc_chain_link(r->pool);
                    if (cl == NULL) {
                        return NGX_ERROR;
                    }

```

```
                    cl->buf = b;
                    cl->next = NULL;

                    b = NULL;

                    mctx = ngx_http_get_module_ctx(r->main,
                                                   ngx_http_ssi_filter_module);
                    bl = mctx->blocks->elts;
                    for (ll = &bl[mctx->blocks->nelts - 1].bufs;
                         *ll;
                         ll = &(*ll)->next)
                    {
                        /* void */
                    }

                    *ll = cl;
                }

                ctx->saved = 0;
            }
        }

        if (ctx->state == ssi_start_state) {
            ctx->copy_start = ctx->pos;
            ctx->copy_end = ctx->pos;

        } else {
            ctx->copy_start = NULL;
            ctx->copy_end = NULL;
        }

        if (rc == NGX_AGAIN) {
            continue;
        }


        b = NULL;

        if (rc == NGX_OK) {

            smcf = ngx_http_get_module_main_conf(r,
                                                 ngx_http_ssi_filter_module);

            cmd = ngx_hash_find(&smcf->hash, ctx->key, ctx->command.data,
                                ctx->command.len);

            if (cmd == NULL) {
                if (ctx->output) {
                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                                  "invalid SSI command: \"%V\"",
                                  &ctx->command);
                    goto ssi_error;
                }

                continue;
            }

            if (!ctx->output && !cmd->block) {

                if (ctx->block) {

                    /* reconstruct the SSI command text */

                    len = 5 + ctx->command.len + 4;

                    param = ctx->params.elts;
                    for (i = 0; i < ctx->params.nelts; i++) {
                        len += 1 + param[i].key.len + 2
                            + param[i].value.len + 1;
                    }

                    b = ngx_create_temp_buf(r->pool, len);

                    if (b == NULL) {
                        return NGX_ERROR;
                    }
```

```
665
666                         cl = ngx_alloc_chain_link(r->pool);
667                         if (cl == NULL) {
668                             return NGX_ERROR;
669                         }
670
671                         cl->buf = b;
672                         cl->next = NULL;
673
674                         *b->last++ = '<';
675                         *b->last++ = '!';
676                         *b->last++ = '-';
677                         *b->last++ = '-';
678                         *b->last++ = '#';
679
680                         b->last = ngx_cpymem(b->last, ctx->command.data,
681                                              ctx->command.len);
682
683                         for (i = 0; i < ctx->params.nelts; i++) {
684                             *b->last++ = ' ';
685                             b->last = ngx_cpymem(b->last, param[i].key.data,
686                                                  param[i].key.len);
687                             *b->last++ = '=';
688                             *b->last++ = '"';
689                             b->last = ngx_cpymem(b->last, param[i].value.data,
690                                                  param[i].value.len);
691                             *b->last++ = '"';
692                         }
693
694                         *b->last++ = ' ';
695                         *b->last++ = '-';
696                         *b->last++ = '-';
697                         *b->last++ = '>';
698
699                         mctx = ngx_http_get_module_ctx(r->main,
700                                                        ngx_http_ssi_filter_module);
701                         bl = mctx->blocks->elts;
702                         for (ll = &bl[mctx->blocks->nelts - 1].bufs;
703                              *ll;
704                              ll = &(*ll)->next)
705                         {
706                             /* void */
707                         }
708
709                         *ll = cl;
710
711                         b = NULL;
712
713                         continue;
714                     }
715
716                     if (cmd->conditional == 0) {
717                         continue;
718                     }
719                 }
720
721                 if (cmd->conditional
722                     && (ctx->conditional == 0
723                         || ctx->conditional > cmd->conditional))
724                 {
725                     ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
726                                   "invalid context of SSI command: \"%V\"",
727                                   &ctx->command);
728                     goto ssi_error;
729                 }
730
731                 if (ctx->params.nelts > NGX_HTTP_SSI_MAX_PARAMS) {
732                     ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
733                                   "too many SSI command parameters: \"%V\"",
734                                   &ctx->command);
735                     goto ssi_error;
736                 }
737
738                 ngx_memzero(params,
739                             (NGX_HTTP_SSI_MAX_PARAMS + 1) * sizeof(ngx_str_t *));
740
```

```
741                 param = ctx->params.elts;
742
743                 for (i = 0; i < ctx->params.nelts; i++) {
744
745                     for (prm = cmd->params; prm->name.len; prm++) {
746
747                         if (param[i].key.len != prm->name.len
748                             || ngx_strncmp(param[i].key.data, prm->name.data,
749                                            prm->name.len) != 0)
750                         {
751                             continue;
752                         }
753
754                         if (!prm->multiple) {
755                             if (params[prm->index]) {
756                                 ngx_log_error(NGX_LOG_ERR,
757                                               r->connection->log, 0,
758                                               "duplicate \"%V\" parameter "
759                                               "in \"%V\" SSI command",
760                                               &param[i].key, &ctx->command);
761
762                                 goto ssi_error;
763                             }
764
765                             params[prm->index] = &param[i].value;
766
767                             break;
768                         }
769
770                         for (index = prm->index; params[index]; index++) {
771                             /* void */
772                         }
773
774                         params[index] = &param[i].value;
775
776                         break;
777                     }
778
779                     if (prm->name.len == 0) {
780                         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
781                                       "invalid parameter name: \"%V\" "
782                                       "in \"%V\" SSI command",
783                                       &param[i].key, &ctx->command);
784
785                         goto ssi_error;
786                     }
787                 }
788
789                 for (prm = cmd->params; prm->name.len; prm++) {
790                     if (prm->mandatory && params[prm->index] == 0) {
791                         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
792                                       "mandatory \"%V\" parameter is absent "
793                                       "in \"%V\" SSI command",
794                                       &prm->name, &ctx->command);
795
796                         goto ssi_error;
797                     }
798                 }
799
800                 if (cmd->flush && ctx->out) {
801
802                     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
803                                    "ssi flush");
804
805                     if (ngx_http_ssi_output(r, ctx) == NGX_ERROR) {
806                         return NGX_ERROR;
807                     }
808                 }
809
810                 rc = cmd->handler(r, ctx, params);
811
812                 if (rc == NGX_OK) {
813                     continue;
814                 }
815
816                 if (rc == NGX_DONE || rc == NGX_AGAIN || rc == NGX_ERROR) {
```

```
                    ngx_http_ssi_buffered(r, ctx);
                    return rc;
                }
            }


            /* rc == NGX_HTTP_SSI_ERROR */

        ssi_error:

            if (slcf->silent_errors) {
                continue;
            }

            if (ctx->free) {
                cl = ctx->free;
                ctx->free = ctx->free->next;
                b = cl->buf;
                ngx_memzero(b, sizeof(ngx_buf_t));

            } else {
                b = ngx_calloc_buf(r->pool);
                if (b == NULL) {
                    return NGX_ERROR;
                }

                cl = ngx_alloc_chain_link(r->pool);
                if (cl == NULL) {
                    return NGX_ERROR;
                }

                cl->buf = b;
            }

            b->memory = 1;
            b->pos = ctx->errmsg.data;
            b->last = ctx->errmsg.data + ctx->errmsg.len;

            cl->next = NULL;
            *ctx->last_out = cl;
            ctx->last_out = &cl->next;

            continue;
        }

        if (ctx->buf->last_buf || ngx_buf_in_memory(ctx->buf)) {
            if (b == NULL) {
                if (ctx->free) {
                    cl = ctx->free;
                    ctx->free = ctx->free->next;
                    b = cl->buf;
                    ngx_memzero(b, sizeof(ngx_buf_t));

                } else {
                    b = ngx_calloc_buf(r->pool);
                    if (b == NULL) {
                        return NGX_ERROR;
                    }

                    cl = ngx_alloc_chain_link(r->pool);
                    if (cl == NULL) {
                        return NGX_ERROR;
                    }

                    cl->buf = b;
                }

                b->sync = 1;

                cl->next = NULL;
                *ctx->last_out = cl;
                ctx->last_out = &cl->next;
            }

            b->last_buf = ctx->buf->last_buf;
            b->shadow = ctx->buf;
```

```
            if (slcf->ignore_recycled_buffers == 0)  {
                b->recycled = ctx->buf->recycled;
            }
        }

        ctx->buf = NULL;

        ctx->saved = ctx->looked;
    }

    if (ctx->out == NULL && ctx->busy == NULL) {
        return NGX_OK;
    }

    return ngx_http_ssi_output(r, ctx);
}


static ngx_int_t
ngx_http_ssi_output(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx)
{
    ngx_int_t     rc;
    ngx_buf_t    *b;
    ngx_chain_t  *cl;

#if 1
    b = NULL;
    for (cl = ctx->out; cl; cl = cl->next) {
        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "ssi out: %p %p", cl->buf, cl->buf->pos);
        if (cl->buf == b) {
            ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                          "the same buf was used in ssi");
            ngx_debug_point();
            return NGX_ERROR;
        }
        b = cl->buf;
    }
#endif

    rc = ngx_http_next_body_filter(r, ctx->out);

    if (ctx->busy == NULL) {
        ctx->busy = ctx->out;

    } else {
        for (cl = ctx->busy; cl->next; cl = cl->next) { /* void */ }
        cl->next = ctx->out;
    }

    ctx->out = NULL;
    ctx->last_out = &ctx->out;

    while (ctx->busy) {

        cl = ctx->busy;
        b = cl->buf;

        if (ngx_buf_size(b) != 0) {
            break;
        }

        if (b->shadow) {
            b->shadow->pos = b->shadow->last;
        }

        ctx->busy = cl->next;

        if (ngx_buf_in_memory(b) || b->in_file) {
            /* add data bufs only to the free buf chain */

            cl->next = ctx->free;
            ctx->free = cl;
        }
    }
```

```
969
970     ngx_http_ssi_buffered(r, ctx);
971
972     return rc;
973 }
974
975
976 static void
977 ngx_http_ssi_buffered(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx)
978 {
979     if (ctx->in || ctx->buf) {
980         r->buffered |= NGX_HTTP_SSI_BUFFERED;
981
982     } else {
983         r->buffered &= ~NGX_HTTP_SSI_BUFFERED;
984     }
985 }
986
987
988 static ngx_int_t
989 ngx_http_ssi_parse(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx)
990 {
991     u_char                *p, *value, *last, *copy_end, ch;
992     size_t                 looked;
993     ngx_http_ssi_state_e   state;
994
995     state = ctx->state;
996     looked = ctx->looked;
997     last = ctx->buf->last;
998     copy_end = ctx->copy_end;
999
1000     for (p = ctx->pos; p < last; p++) {
1001
1002         ch = *p;
1003
1004         if (state == ssi_start_state) {
1005
1006             /* the tight loop */
1007
1008             for ( ;; ) {
1009                 if (ch == '<') {
1010                     copy_end = p;
1011                     looked = 1;
1012                     state = ssi_tag_state;
1013
1014                     goto tag_started;
1015                 }
1016
1017                 if (++p == last) {
1018                     break;
1019                 }
1020
1021                 ch = *p;
1022             }
1023
1024             ctx->state = state;
1025             ctx->pos = p;
1026             ctx->looked = looked;
1027             ctx->copy_end = p;
1028
1029             if (ctx->copy_start == NULL) {
1030                 ctx->copy_start = ctx->buf->pos;
1031             }
1032
1033             return NGX_AGAIN;
1034
1035         tag_started:
1036
1037             continue;
1038         }
1039
1040         switch (state) {
1041
1042         case ssi_start_state:
1043             /* not reached */
1044             break;
```

```
1045
1046        case ssi_tag_state:
1047            switch (ch) {
1048            case '!':
1049                looked = 2;
1050                state = ssi_comment0_state;
1051                break;
1052
1053            case '<':
1054                copy_end = p;
1055                break;
1056
1057            default:
1058                copy_end = p;
1059                looked = 0;
1060                state = ssi_start_state;
1061                break;
1062            }
1063
1064            break;
1065
1066        case ssi_comment0_state:
1067            switch (ch) {
1068            case '-':
1069                looked = 3;
1070                state = ssi_comment1_state;
1071                break;
1072
1073            case '<':
1074                copy_end = p;
1075                looked = 1;
1076                state = ssi_tag_state;
1077                break;
1078
1079            default:
1080                copy_end = p;
1081                looked = 0;
1082                state = ssi_start_state;
1083                break;
1084            }
1085
1086            break;
1087
1088        case ssi_comment1_state:
1089            switch (ch) {
1090            case '-':
1091                looked = 4;
1092                state = ssi_sharp_state;
1093                break;
1094
1095            case '<':
1096                copy_end = p;
1097                looked = 1;
1098                state = ssi_tag_state;
1099                break;
1100
1101            default:
1102                copy_end = p;
1103                looked = 0;
1104                state = ssi_start_state;
1105                break;
1106            }
1107
1108            break;
1109
1110        case ssi_sharp_state:
1111            switch (ch) {
1112            case '#':
1113                if (p - ctx->pos < 4) {
1114                    ctx->saved = 0;
1115                }
1116                looked = 0;
1117                state = ssi_precommand_state;
1118                break;
1119
1120            case '<':
```

```
                copy_end = p;
                looked = 1;
                state = ssi_tag_state;
                break;

            default:
                copy_end = p;
                looked = 0;
                state = ssi_start_state;
                break;
            }

            break;

        case ssi_precommand_state:
            switch (ch) {
            case ' ':
            case CR:
            case LF:
            case '\t':
                break;

            default:
                ctx->command.len = 1;
                ctx->command.data = ngx_pnalloc(r->pool,
                                                NGX_HTTP_SSI_COMMAND_LEN);
                if (ctx->command.data == NULL) {
                    return NGX_ERROR;
                }

                ctx->command.data[0] = ch;

                ctx->key = 0;
                ctx->key = ngx_hash(ctx->key, ch);

                ctx->params.nelts = 0;

                state = ssi_command_state;
                break;
            }

            break;

        case ssi_command_state:
            switch (ch) {
            case ' ':
            case CR:
            case LF:
            case '\t':
                state = ssi_preparam_state;
                break;

            case '-':
                state = ssi_comment_end0_state;
                break;

            default:
                if (ctx->command.len == NGX_HTTP_SSI_COMMAND_LEN) {
                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                                  "the \"%V%c...\" SSI command is too long",
                                  &ctx->command, ch);

                    state = ssi_error_state;
                    break;
                }

                ctx->command.data[ctx->command.len++] = ch;
                ctx->key = ngx_hash(ctx->key, ch);
            }

            break;

        case ssi_preparam_state:
            switch (ch) {
            case ' ':
            case CR:
```

```
            case LF:
            case '\t':
                break;

            case '-':
                state = ssi_comment_end0_state;
                break;

            default:
                ctx->param = ngx_array_push(&ctx->params);
                if (ctx->param == NULL) {
                    return NGX_ERROR;
                }

                ctx->param->key.len = 1;
                ctx->param->key.data = ngx_pnalloc(r->pool,
                                                   NGX_HTTP_SSI_PARAM_LEN);
                if (ctx->param->key.data == NULL) {
                    return NGX_ERROR;
                }

                ctx->param->key.data[0] = ch;

                ctx->param->value.len = 0;

                if (ctx->value_buf == NULL) {
                    ctx->param->value.data = ngx_pnalloc(r->pool,
                                                         ctx->value_len + 1);
                    if (ctx->param->value.data == NULL) {
                        return NGX_ERROR;
                    }

                } else {
                    ctx->param->value.data = ctx->value_buf;
                }

                state = ssi_param_state;
                break;
            }

            break;

        case ssi_param_state:
            switch (ch) {
            case ' ':
            case CR:
            case LF:
            case '\t':
                state = ssi_preequal_state;
                break;

            case '=':
                state = ssi_prevalue_state;
                break;

            case '-':
                state = ssi_error_end0_state;

                ctx->param->key.data[ctx->param->key.len++] = ch;
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "invalid \"%V\" parameter in \"%V\" SSI command",
                              &ctx->param->key, &ctx->command);
                break;

            default:
                if (ctx->param->key.len == NGX_HTTP_SSI_PARAM_LEN) {
                    state = ssi_error_state;
                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                                  "too long \"%V%c...\" parameter in "
                                  "\"%V\" SSI command",
                                  &ctx->param->key, ch, &ctx->command);
                    break;
                }

                ctx->param->key.data[ctx->param->key.len++] = ch;
            }
```

```
1273
1274            break;
1275
1276        case ssi_preequal_state:
1277            switch (ch) {
1278            case ' ':
1279            case CR:
1280            case LF:
1281            case '\t':
1282                break;
1283
1284            case '=':
1285                state = ssi_prevalue_state;
1286                break;
1287
1288            default:
1289                if (ch == '-') {
1290                    state = ssi_error_end0_state;
1291                } else {
1292                    state = ssi_error_state;
1293                }
1294
1295                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1296                              "unexpected \"%c\" symbol after \"%V\" "
1297                              "parameter in \"%V\" SSI command",
1298                              ch, &ctx->param->key, &ctx->command);
1299                break;
1300            }
1301
1302            break;
1303
1304        case ssi_prevalue_state:
1305            switch (ch) {
1306            case ' ':
1307            case CR:
1308            case LF:
1309            case '\t':
1310                break;
1311
1312            case '"':
1313                state = ssi_double_quoted_value_state;
1314                break;
1315
1316            case '\'':
1317                state = ssi_quoted_value_state;
1318                break;
1319
1320            default:
1321                if (ch == '-') {
1322                    state = ssi_error_end0_state;
1323                } else {
1324                    state = ssi_error_state;
1325                }
1326
1327                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1328                              "unexpected \"%c\" symbol before value of "
1329                              "\"%V\" parameter in \"%V\" SSI command",
1330                              ch, &ctx->param->key, &ctx->command);
1331                break;
1332            }
1333
1334            break;
1335
1336        case ssi_double_quoted_value_state:
1337            switch (ch) {
1338            case '"':
1339                state = ssi_postparam_state;
1340                break;
1341
1342            case '\\':
1343                ctx->saved_state = ssi_double_quoted_value_state;
1344                state = ssi_quoted_symbol_state;
1345
1346                /* fall through */
1347
1348            default:
```

```
1349                if (ctx->param->value.len == ctx->value_len) {
1350                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1351                                  "too long \"%V%c...\" value of \"%V\" "
1352                                  "parameter in \"%V\" SSI command",
1353                                  &ctx->param->value, ch, &ctx->param->key,
1354                                  &ctx->command);
1355                    state = ssi_error_state;
1356                    break;
1357                }
1358
1359                ctx->param->value.data[ctx->param->value.len++] = ch;
1360            }
1361
1362            break;
1363
1364        case ssi_quoted_value_state:
1365            switch (ch) {
1366            case '\'':
1367                state = ssi_postparam_state;
1368                break;
1369
1370            case '\\':
1371                ctx->saved_state = ssi_quoted_value_state;
1372                state = ssi_quoted_symbol_state;
1373
1374                /* fall through */
1375
1376            default:
1377                if (ctx->param->value.len == ctx->value_len) {
1378                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1379                                  "too long \"%V%c...\" value of \"%V\" "
1380                                  "parameter in \"%V\" SSI command",
1381                                  &ctx->param->value, ch, &ctx->param->key,
1382                                  &ctx->command);
1383                    state = ssi_error_state;
1384                    break;
1385                }
1386
1387                ctx->param->value.data[ctx->param->value.len++] = ch;
1388            }
1389
1390            break;
1391
1392        case ssi_quoted_symbol_state:
1393            state = ctx->saved_state;
1394
1395            if (ctx->param->value.len == ctx->value_len) {
1396                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1397                              "too long \"%V%c...\" value of \"%V\" "
1398                              "parameter in \"%V\" SSI command",
1399                              &ctx->param->value, ch, &ctx->param->key,
1400                              &ctx->command);
1401                state = ssi_error_state;
1402                break;
1403            }
1404
1405            ctx->param->value.data[ctx->param->value.len++] = ch;
1406
1407            break;
1408
1409        case ssi_postparam_state:
1410
1411            if (ctx->param->value.len + 1 < ctx->value_len / 2) {
1412                value = ngx_pnalloc(r->pool, ctx->param->value.len + 1);
1413                if (value == NULL) {
1414                    return NGX_ERROR;
1415                }
1416
1417                ngx_memcpy(value, ctx->param->value.data,
1418                           ctx->param->value.len);
1419
1420                ctx->value_buf = ctx->param->value.data;
1421                ctx->param->value.data = value;
1422
1423            } else {
1424                ctx->value_buf = NULL;
```

```
1425            }
1426
1427            switch (ch) {
1428            case ' ':
1429            case CR:
1430            case LF:
1431            case '\t':
1432                state = ssi_preparam_state;
1433                break;
1434
1435            case '-':
1436                state = ssi_comment_end0_state;
1437                break;
1438
1439            default:
1440                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1441                              "unexpected \"%c\" symbol after \"%V\" value "
1442                              "of \"%V\" parameter in \"%V\" SSI command",
1443                              ch, &ctx->param->value, &ctx->param->key,
1444                              &ctx->command);
1445                state = ssi_error_state;
1446                break;
1447            }
1448
1449            break;
1450
1451        case ssi_comment_end0_state:
1452            switch (ch) {
1453            case '-':
1454                state = ssi_comment_end1_state;
1455                break;
1456
1457            default:
1458                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1459                              "unexpected \"%c\" symbol in \"%V\" SSI command",
1460                              ch, &ctx->command);
1461                state = ssi_error_state;
1462                break;
1463            }
1464
1465            break;
1466
1467        case ssi_comment_end1_state:
1468            switch (ch) {
1469            case '>':
1470                ctx->state = ssi_start_state;
1471                ctx->pos = p + 1;
1472                ctx->looked = looked;
1473                ctx->copy_end = copy_end;
1474
1475                if (ctx->copy_start == NULL && copy_end) {
1476                    ctx->copy_start = ctx->buf->pos;
1477                }
1478
1479                return NGX_OK;
1480
1481            default:
1482                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1483                              "unexpected \"%c\" symbol in \"%V\" SSI command",
1484                              ch, &ctx->command);
1485                state = ssi_error_state;
1486                break;
1487            }
1488
1489            break;
1490
1491        case ssi_error_state:
1492            switch (ch) {
1493            case '-':
1494                state = ssi_error_end0_state;
1495                break;
1496
1497            default:
1498                break;
1499            }
1500
```

```
            break;

        case ssi_error_end0_state:
            switch (ch) {
            case '-':
                state = ssi_error_end1_state;
                break;

            default:
                state = ssi_error_state;
                break;
            }

            break;

        case ssi_error_end1_state:
            switch (ch) {
            case '>':
                ctx->state = ssi_start_state;
                ctx->pos = p + 1;
                ctx->looked = looked;
                ctx->copy_end = copy_end;

                if (ctx->copy_start == NULL && copy_end) {
                    ctx->copy_start = ctx->buf->pos;
                }

                return NGX_HTTP_SSI_ERROR;

            default:
                state = ssi_error_state;
                break;
            }

            break;
        }
    }

    ctx->state = state;
    ctx->pos = p;
    ctx->looked = looked;

    ctx->copy_end = (state == ssi_start_state) ? p : copy_end;

    if (ctx->copy_start == NULL && ctx->copy_end) {
        ctx->copy_start = ctx->buf->pos;
    }

    return NGX_AGAIN;
}


static ngx_str_t *
ngx_http_ssi_get_variable(ngx_http_request_t *r, ngx_str_t *name,
    ngx_uint_t key)
{
    ngx_uint_t           i;
    ngx_list_part_t     *part;
    ngx_http_ssi_var_t  *var;
    ngx_http_ssi_ctx_t  *ctx;

    ctx = ngx_http_get_module_ctx(r->main, ngx_http_ssi_filter_module);

#if (NGX_PCRE)
    {
    ngx_str_t  *value;

    if (key >= '0' && key <= '9') {
        i = key - '0';

        if (i < ctx->ncaptures) {
            value = ngx_palloc(r->pool, sizeof(ngx_str_t));
            if (value == NULL) {
                return NULL;
            }

```

```
            i *= 2;

            value->data = ctx->captures_data + ctx->captures[i];
            value->len = ctx->captures[i + 1] - ctx->captures[i];

            return value;
        }
    }
    }
#endif

    if (ctx->variables == NULL) {
        return NULL;
    }

    part = &ctx->variables->part;
    var = part->elts;

    for (i = 0; /* void */ ; i++) {

        if (i >= part->nelts) {
            if (part->next == NULL) {
                break;
            }

            part = part->next;
            var = part->elts;
            i = 0;
        }

        if (name->len != var[i].name.len) {
            continue;
        }

        if (key != var[i].key) {
            continue;
        }

        if (ngx_strncmp(name->data, var[i].name.data, name->len) == 0) {
            return &var[i].value;
        }
    }

    return NULL;
}


static ngx_int_t
ngx_http_ssi_evaluate_string(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t *text, ngx_uint_t flags)
{
    u_char                     ch, *p, **value, *data, *part_data;
    size_t                    *size, len, prefix, part_len;
    ngx_str_t                  var, *val;
    ngx_int_t                  key;
    ngx_uint_t                 i, n, bracket, quoted;
    ngx_array_t                lengths, values;
    ngx_http_variable_value_t  *vv;

    n = ngx_http_script_variables_count(text);

    if (n == 0) {

        data = text->data;
        p = data;

        if ((flags & NGX_HTTP_SSI_ADD_PREFIX) && text->data[0] != '/') {

            for (prefix = r->uri.len; prefix; prefix--) {
                if (r->uri.data[prefix - 1] == '/') {
                    break;
                }
            }

            if (prefix) {
                len = prefix + text->len;
```

```
1653
1654                data = ngx_pnalloc(r->pool, len);
1655                if (data == NULL) {
1656                    return NGX_ERROR;
1657                }
1658
1659                p = ngx_copy(data, r->uri.data, prefix);
1660            }
1661        }
1662
1663        quoted = 0;
1664
1665        for (i = 0; i < text->len; i++) {
1666            ch = text->data[i];
1667
1668            if (!quoted) {
1669
1670                if (ch == '\\') {
1671                    quoted = 1;
1672                    continue;
1673                }
1674
1675            } else {
1676                quoted = 0;
1677
1678                if (ch != '\\' && ch != '\'' && ch != '"' && ch != '$') {
1679                    *p++ = '\\';
1680                }
1681            }
1682
1683            *p++ = ch;
1684        }
1685
1686        text->len = p - data;
1687        text->data = data;
1688
1689        return NGX_OK;
1690    }
1691
1692    if (ngx_array_init(&lengths, r->pool, 8, sizeof(size_t *)) != NGX_OK) {
1693        return NGX_ERROR;
1694    }
1695
1696    if (ngx_array_init(&values, r->pool, 8, sizeof(u_char *)) != NGX_OK) {
1697        return NGX_ERROR;
1698    }
1699
1700    len = 0;
1701    i = 0;
1702
1703    while (i < text->len) {
1704
1705        if (text->data[i] == '$') {
1706
1707            var.len = 0;
1708
1709            if (++i == text->len) {
1710                goto invalid_variable;
1711            }
1712
1713            if (text->data[i] == '{') {
1714                bracket = 1;
1715
1716                if (++i == text->len) {
1717                    goto invalid_variable;
1718                }
1719
1720                var.data = &text->data[i];
1721
1722            } else {
1723                bracket = 0;
1724                var.data = &text->data[i];
1725            }
1726
1727            for ( /* void */ ; i < text->len; i++, var.len++) {
1728                ch = text->data[i];
```

```
1729
1730                if (ch == '}' && bracket) {
1731                    i++;
1732                    bracket = 0;
1733                    break;
1734                }
1735
1736                if ((ch >= 'A' && ch <= 'Z')
1737                    || (ch >= 'a' && ch <= 'z')
1738                    || (ch >= '0' && ch <= '9')
1739                    || ch == '_')
1740                {
1741                    continue;
1742                }
1743
1744                break;
1745            }
1746
1747            if (bracket) {
1748                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1749                              "the closing bracket in \"%V\" "
1750                              "variable is missing", &var);
1751                return NGX_HTTP_SSI_ERROR;
1752            }
1753
1754            if (var.len == 0) {
1755                goto invalid_variable;
1756            }
1757
1758            key = ngx_hash_strlow(var.data, var.data, var.len);
1759
1760            val = ngx_http_ssi_get_variable(r, &var, key);
1761
1762            if (val == NULL) {
1763                vv = ngx_http_get_variable(r, &var, key);
1764                if (vv == NULL) {
1765                    return NGX_ERROR;
1766                }
1767
1768                if (vv->not_found) {
1769                    continue;
1770                }
1771
1772                part_data = vv->data;
1773                part_len = vv->len;
1774
1775            } else {
1776                part_data = val->data;
1777                part_len = val->len;
1778            }
1779
1780        } else {
1781            part_data = &text->data[i];
1782            quoted = 0;
1783
1784            for (p = part_data; i < text->len; i++) {
1785                ch = text->data[i];
1786
1787                if (!quoted) {
1788
1789                    if (ch == '\\') {
1790                        quoted = 1;
1791                        continue;
1792                    }
1793
1794                    if (ch == '$') {
1795                        break;
1796                    }
1797
1798                } else {
1799                    quoted = 0;
1800
1801                    if (ch != '\\' && ch != '\'' && ch != '"' && ch != '$') {
1802                        *p++ = '\\';
1803                    }
1804                }
```

```
                *p++ = ch;
            }

            part_len = p - part_data;
        }

        len += part_len;

        size = ngx_array_push(&lengths);
        if (size == NULL) {
            return NGX_ERROR;
        }

        *size = part_len;

        value = ngx_array_push(&values);
        if (value == NULL) {
            return NGX_ERROR;
        }

        *value = part_data;
    }

    prefix = 0;

    size = lengths.elts;
    value = values.elts;

    if (flags & NGX_HTTP_SSI_ADD_PREFIX) {
        for (i = 0; i < values.nelts; i++) {
            if (size[i] != 0) {
                if (*value[i] != '/') {
                    for (prefix = r->uri.len; prefix; prefix--) {
                        if (r->uri.data[prefix - 1] == '/') {
                            len += prefix;
                            break;
                        }
                    }
                }

                break;
            }
        }
    }

    p = ngx_pnalloc(r->pool, len + ((flags & NGX_HTTP_SSI_ADD_ZERO) ? 1 : 0));
    if (p == NULL) {
        return NGX_ERROR;
    }

    text->len = len;
    text->data = p;

    p = ngx_copy(p, r->uri.data, prefix);

    for (i = 0; i < values.nelts; i++) {
        p = ngx_copy(p, value[i], size[i]);
    }

    return NGX_OK;

invalid_variable:

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "invalid variable name in \"%V\"", text);

    return NGX_HTTP_SSI_ERROR;
}


static ngx_int_t
ngx_http_ssi_regex_match(ngx_http_request_t *r, ngx_str_t *pattern,
    ngx_str_t *str)
{
#if (NGX_PCRE)
```

```
    int                  rc, *captures;
    u_char               *p, errstr[NGX_MAX_CONF_ERRSTR];
    size_t               size;
    ngx_int_t            key;
    ngx_str_t            *vv, name, value;
    ngx_uint_t           i, n;
    ngx_http_ssi_ctx_t   *ctx;
    ngx_http_ssi_var_t   *var;
    ngx_regex_compile_t  rgc;

    ngx_memzero(&rgc, sizeof(ngx_regex_compile_t));

    rgc.pattern = *pattern;
    rgc.pool = r->pool;
    rgc.err.len = NGX_MAX_CONF_ERRSTR;
    rgc.err.data = errstr;

    if (ngx_regex_compile(&rgc) != NGX_OK) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0, "%V", &rgc.err);
        return NGX_HTTP_SSI_ERROR;
    }

    n = (rgc.captures + 1) * 3;

    captures = ngx_palloc(r->pool, n * sizeof(int));
    if (captures == NULL) {
        return NGX_ERROR;
    }

    rc = ngx_regex_exec(rgc.regex, str, captures, n);

    if (rc < NGX_REGEX_NO_MATCHED) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      ngx_regex_exec_n " failed: %i on \"%V\" using \"%V\"",
                      rc, str, pattern);
        return NGX_HTTP_SSI_ERROR;
    }

    if (rc == NGX_REGEX_NO_MATCHED) {
        return NGX_DECLINED;
    }

    ctx = ngx_http_get_module_ctx(r->main, ngx_http_ssi_filter_module);

    ctx->ncaptures = rc;
    ctx->captures = captures;
    ctx->captures_data = str->data;

    if (rgc.named_captures > 0) {

        if (ctx->variables == NULL) {
            ctx->variables = ngx_list_create(r->pool, 4,
                                             sizeof(ngx_http_ssi_var_t));
            if (ctx->variables == NULL) {
                return NGX_ERROR;
            }
        }

        size = rgc.name_size;
        p = rgc.names;

        for (i = 0; i < (ngx_uint_t) rgc.named_captures; i++, p += size) {

            name.data = &p[2];
            name.len = ngx_strlen(name.data);

            n = 2 * ((p[0] << 8) + p[1]);

            value.data = &str->data[captures[n]];
            value.len = captures[n + 1] - captures[n];

            key = ngx_hash_strlow(name.data, name.data, name.len);

            vv = ngx_http_ssi_get_variable(r, &name, key);

            if (vv) {
```

```
                *vv = value;
                continue;
            }

            var = ngx_list_push(ctx->variables);
            if (var == NULL) {
                return NGX_ERROR;
            }

            var->name = name;
            var->key = key;
            var->value = value;
        }
    }

    return NGX_OK;

#else

    ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                  "the using of the regex \"%V\" in SSI requires PCRE library",
                  pattern);
    return NGX_HTTP_SSI_ERROR;

#endif
}


static ngx_int_t
ngx_http_ssi_include(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t **params)
{
    ngx_int_t                   rc, key;
    ngx_str_t                  *uri, *file, *wait, *set, *stub, args;
    ngx_buf_t                  *b;
    ngx_uint_t                  flags, i;
    ngx_chain_t                *cl, *tl, **ll, *out;
    ngx_http_request_t         *sr;
    ngx_http_ssi_var_t         *var;
    ngx_http_ssi_ctx_t         *mctx;
    ngx_http_ssi_block_t       *bl;
    ngx_http_post_subrequest_t  *psr;

    uri = params[NGX_HTTP_SSI_INCLUDE_VIRTUAL];
    file = params[NGX_HTTP_SSI_INCLUDE_FILE];
    wait = params[NGX_HTTP_SSI_INCLUDE_WAIT];
    set = params[NGX_HTTP_SSI_INCLUDE_SET];
    stub = params[NGX_HTTP_SSI_INCLUDE_STUB];

    if (uri && file) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "inlcusion may be either virtual=\"%V\" or file=\"%V\"",
                      uri, file);
        return NGX_HTTP_SSI_ERROR;
    }

    if (uri == NULL && file == NULL) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "no parameter in \"include\" SSI command");
        return NGX_HTTP_SSI_ERROR;
    }

    if (set && stub) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "\"set\" and \"stub\" cannot be used together "
                      "in \"include\" SSI command");
        return NGX_HTTP_SSI_ERROR;
    }

    if (wait) {
        if (uri == NULL) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "\"wait\" cannot be used with file=\"%V\"", file);
            return NGX_HTTP_SSI_ERROR;
        }

```

```
        if (wait->len == 2
            && ngx_strncasecmp(wait->data, (u_char *) "no", 2) == 0)
        {
            wait = NULL;

        } else if (wait->len != 3
                   || ngx_strncasecmp(wait->data, (u_char *) "yes", 3) != 0)
        {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "invalid value \"%V\" in the \"wait\" parameter",
                          wait);
            return NGX_HTTP_SSI_ERROR;
        }
    }

    if (uri == NULL) {
        uri = file;
        wait = (ngx_str_t *) -1;
    }

    rc = ngx_http_ssi_evaluate_string(r, ctx, uri, NGX_HTTP_SSI_ADD_PREFIX);

    if (rc != NGX_OK) {
        return rc;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi include: \"%V\"", uri);

    ngx_str_null(&args);
    flags = NGX_HTTP_LOG_UNSAFE;

    if (ngx_http_parse_unsafe_uri(r, uri, &args, &flags) != NGX_OK) {
        return NGX_HTTP_SSI_ERROR;
    }

    psr = NULL;

    mctx = ngx_http_get_module_ctx(r->main, ngx_http_ssi_filter_module);

    if (stub) {
        if (mctx->blocks) {
            bl = mctx->blocks->elts;
            for (i = 0; i < mctx->blocks->nelts; i++) {
                if (stub->len == bl[i].name.len
                    && ngx_strncmp(stub->data, bl[i].name.data, stub->len) == 0)
                {
                    goto found;
                }
            }
        }

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "\"stub\"=\"%V\" for \"include\" not found", stub);
        return NGX_HTTP_SSI_ERROR;

    found:

        psr = ngx_palloc(r->pool, sizeof(ngx_http_post_subrequest_t));
        if (psr == NULL) {
            return NGX_ERROR;
        }

        psr->handler = ngx_http_ssi_stub_output;

        if (bl[i].count++) {

            out = NULL;
            ll = &out;

            for (tl = bl[i].bufs; tl; tl = tl->next) {

                if (ctx->free) {
                    cl = ctx->free;
                    ctx->free = ctx->free->next;
                    b = cl->buf;
```

```
2109
2110                } else {
2111                    b = ngx_alloc_buf(r->pool);
2112                    if (b == NULL) {
2113                        return NGX_ERROR;
2114                    }
2115
2116                    cl = ngx_alloc_chain_link(r->pool);
2117                    if (cl == NULL) {
2118                        return NGX_ERROR;
2119                    }
2120
2121                    cl->buf = b;
2122                }
2123
2124                ngx_memcpy(b, tl->buf, sizeof(ngx_buf_t));
2125
2126                b->pos = b->start;
2127
2128                *ll = cl;
2129                cl->next = NULL;
2130                ll = &cl->next;
2131            }
2132
2133            psr->data = out;
2134
2135        } else {
2136            psr->data = bl[i].bufs;
2137        }
2138    }
2139
2140    if (wait) {
2141        flags |= NGX_HTTP_SUBREQUEST_WAITED;
2142    }
2143
2144    if (set) {
2145        key = ngx_hash_strlow(set->data, set->data, set->len);
2146
2147        psr = ngx_palloc(r->pool, sizeof(ngx_http_post_subrequest_t));
2148        if (psr == NULL) {
2149            return NGX_ERROR;
2150        }
2151
2152        psr->handler = ngx_http_ssi_set_variable;
2153        psr->data = ngx_http_ssi_get_variable(r, set, key);
2154
2155        if (psr->data == NULL) {
2156
2157            if (mctx->variables == NULL) {
2158                mctx->variables = ngx_list_create(r->pool, 4,
2159                                                  sizeof(ngx_http_ssi_var_t));
2160                if (mctx->variables == NULL) {
2161                    return NGX_ERROR;
2162                }
2163            }
2164
2165            var = ngx_list_push(mctx->variables);
2166            if (var == NULL) {
2167                return NGX_ERROR;
2168            }
2169
2170            var->name = *set;
2171            var->key = key;
2172            var->value = ngx_http_ssi_null_string;
2173            psr->data = &var->value;
2174        }
2175
2176        flags |= NGX_HTTP_SUBREQUEST_IN_MEMORY|NGX_HTTP_SUBREQUEST_WAITED;
2177    }
2178
2179    if (ngx_http_subrequest(r, uri, &args, &sr, psr, flags) != NGX_OK) {
2180        return NGX_HTTP_SSI_ERROR;
2181    }
2182
2183    if (wait == NULL && set == NULL) {
2184        return NGX_OK;
```

```
2185     }
2186
2187     if (ctx->wait == NULL) {
2188         ctx->wait = sr;
2189
2190         return NGX_AGAIN;
2191
2192     } else {
2193         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2194                       "can only wait for one subrequest at a time");
2195     }
2196
2197     return NGX_OK;
2198 }
2199
2200
2201 static ngx_int_t
2202 ngx_http_ssi_stub_output(ngx_http_request_t *r, void *data, ngx_int_t rc)
2203 {
2204     ngx_chain_t  *out;
2205
2206     if (rc == NGX_ERROR || r->connection->error || r->request_output) {
2207         return rc;
2208     }
2209
2210     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2211                    "ssi stub output: \"%V?%V\"", &r->uri, &r->args);
2212
2213     out = data;
2214
2215     if (!r->header_sent) {
2216         r->headers_out.content_type_len =
2217                                       r->parent->headers_out.content_type_len;
2218         r->headers_out.content_type = r->parent->headers_out.content_type;
2219
2220         if (ngx_http_send_header(r) == NGX_ERROR) {
2221             return NGX_ERROR;
2222         }
2223     }
2224
2225     return ngx_http_output_filter(r, out);
2226 }
2227
2228
2229 static ngx_int_t
2230 ngx_http_ssi_set_variable(ngx_http_request_t *r, void *data, ngx_int_t rc)
2231 {
2232     ngx_str_t  *value = data;
2233
2234     if (r->upstream) {
2235         value->len = r->upstream->buffer.last - r->upstream->buffer.pos;
2236         value->data = r->upstream->buffer.pos;
2237     }
2238
2239     return rc;
2240 }
2241
2242
2243 static ngx_int_t
2244 ngx_http_ssi_echo(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
2245     ngx_str_t **params)
2246 {
2247     u_char                     *p;
2248     uintptr_t                   len;
2249     ngx_int_t                   key;
2250     ngx_buf_t                  *b;
2251     ngx_str_t                  *var, *value, *enc, text;
2252     ngx_chain_t                *cl;
2253     ngx_http_variable_value_t  *vv;
2254
2255     var = params[NGX_HTTP_SSI_ECHO_VAR];
2256
2257     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2258                    "ssi echo \"%V\"", var);
2259
2260     key = ngx_hash_strlow(var->data, var->data, var->len);
```

```
2261
2262      value = ngx_http_ssi_get_variable(r, var, key);
2263
2264      if (value == NULL) {
2265          vv = ngx_http_get_variable(r, var, key);
2266
2267          if (vv == NULL) {
2268              return NGX_HTTP_SSI_ERROR;
2269          }
2270
2271          if (!vv->not_found) {
2272              text.data = vv->data;
2273              text.len = vv->len;
2274              value = &text;
2275          }
2276      }
2277
2278      if (value == NULL) {
2279          value = params[NGX_HTTP_SSI_ECHO_DEFAULT];
2280
2281          if (value == NULL) {
2282              value = &ngx_http_ssi_none;
2283
2284          } else if (value->len == 0) {
2285              return NGX_OK;
2286          }
2287
2288      } else {
2289          if (value->len == 0) {
2290              return NGX_OK;
2291          }
2292      }
2293
2294      enc = params[NGX_HTTP_SSI_ECHO_ENCODING];
2295
2296      if (enc) {
2297          if (enc->len == 4 && ngx_strncmp(enc->data, "none", 4) == 0) {
2298
2299              ctx->encoding = NGX_HTTP_SSI_NO_ENCODING;
2300
2301          } else if (enc->len == 3 && ngx_strncmp(enc->data, "url", 3) == 0) {
2302
2303              ctx->encoding = NGX_HTTP_SSI_URL_ENCODING;
2304
2305          } else if (enc->len == 6 && ngx_strncmp(enc->data, "entity", 6) == 0) {
2306
2307              ctx->encoding = NGX_HTTP_SSI_ENTITY_ENCODING;
2308
2309          } else {
2310              ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2311                            "unknown encoding \"%V\" in the \"echo\" command",
2312                            enc);
2313          }
2314      }
2315
2316      p = value->data;
2317
2318      switch (ctx->encoding) {
2319
2320      case NGX_HTTP_SSI_URL_ENCODING:
2321          len = 2 * ngx_escape_uri(NULL, value->data, value->len,
2322                                   NGX_ESCAPE_HTML);
2323
2324          if (len) {
2325              p = ngx_pnalloc(r->pool, value->len + len);
2326              if (p == NULL) {
2327                  return NGX_HTTP_SSI_ERROR;
2328              }
2329
2330              (void) ngx_escape_uri(p, value->data, value->len, NGX_ESCAPE_HTML);
2331          }
2332
2333          len += value->len;
2334          break;
2335
2336      case NGX_HTTP_SSI_ENTITY_ENCODING:
```

```
2337            len = ngx_escape_html(NULL, value->data, value->len);
2338
2339            if (len) {
2340                p = ngx_pnalloc(r->pool, value->len + len);
2341                if (p == NULL) {
2342                    return NGX_HTTP_SSI_ERROR;
2343                }
2344
2345                (void) ngx_escape_html(p, value->data, value->len);
2346            }
2347
2348            len += value->len;
2349            break;
2350
2351        default: /* NGX_HTTP_SSI_NO_ENCODING */
2352            len = value->len;
2353            break;
2354        }
2355
2356        b = ngx_calloc_buf(r->pool);
2357        if (b == NULL) {
2358            return NGX_HTTP_SSI_ERROR;
2359        }
2360
2361        cl = ngx_alloc_chain_link(r->pool);
2362        if (cl == NULL) {
2363            return NGX_HTTP_SSI_ERROR;
2364        }
2365
2366        b->memory = 1;
2367        b->pos = p;
2368        b->last = p + len;
2369
2370        cl->buf = b;
2371        cl->next = NULL;
2372        *ctx->last_out = cl;
2373        ctx->last_out = &cl->next;
2374
2375        return NGX_OK;
2376    }
2377
2378
2379    static ngx_int_t
2380    ngx_http_ssi_config(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
2381        ngx_str_t **params)
2382    {
2383        ngx_str_t  *value;
2384
2385        value = params[NGX_HTTP_SSI_CONFIG_TIMEFMT];
2386
2387        if (value) {
2388            ctx->timefmt.len = value->len;
2389            ctx->timefmt.data = ngx_pnalloc(r->pool, value->len + 1);
2390            if (ctx->timefmt.data == NULL) {
2391                return NGX_HTTP_SSI_ERROR;
2392            }
2393
2394            ngx_cpystrn(ctx->timefmt.data, value->data, value->len + 1);
2395        }
2396
2397        value = params[NGX_HTTP_SSI_CONFIG_ERRMSG];
2398
2399        if (value) {
2400            ctx->errmsg = *value;
2401        }
2402
2403        return NGX_OK;
2404    }
2405
2406
2407    static ngx_int_t
2408    ngx_http_ssi_set(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
2409        ngx_str_t **params)
2410    {
2411        ngx_int_t             key, rc;
2412        ngx_str_t            *name, *value, *vv;
```

```
    ngx_http_ssi_var_t  *var;
    ngx_http_ssi_ctx_t  *mctx;

    mctx = ngx_http_get_module_ctx(r->main, ngx_http_ssi_filter_module);

    if (mctx->variables == NULL) {
        mctx->variables = ngx_list_create(r->pool, 4,
                                          sizeof(ngx_http_ssi_var_t));
        if (mctx->variables == NULL) {
            return NGX_ERROR;
        }
    }

    name = params[NGX_HTTP_SSI_SET_VAR];
    value = params[NGX_HTTP_SSI_SET_VALUE];

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi set \"%V\" \"%V\"", name, value);

    rc = ngx_http_ssi_evaluate_string(r, ctx, value, 0);

    if (rc != NGX_OK) {
        return rc;
    }

    key = ngx_hash_strlow(name->data, name->data, name->len);

    vv = ngx_http_ssi_get_variable(r, name, key);

    if (vv) {
        *vv = *value;
        return NGX_OK;
    }

    var = ngx_list_push(mctx->variables);
    if (var == NULL) {
        return NGX_ERROR;
    }

    var->name = *name;
    var->key = key;
    var->value = *value;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "set: \"%V\"=\"%V\"", name, value);

    return NGX_OK;
}


static ngx_int_t
ngx_http_ssi_if(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t **params)
{
    u_char       *p, *last;
    ngx_str_t    *expr, left, right;
    ngx_int_t     rc;
    ngx_uint_t    negative, noregex, flags;

    if (ctx->command.len == 2) {
        if (ctx->conditional) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "the \"if\" command inside the \"if\" command");
            return NGX_HTTP_SSI_ERROR;
        }
    }

    if (ctx->output_chosen) {
        ctx->output = 0;
        return NGX_OK;
    }

    expr = params[NGX_HTTP_SSI_IF_EXPR];

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi if expr=\"%V\"", expr);
```

```
2489
2490        left.data = expr->data;
2491        last = expr->data + expr->len;
2492
2493        for (p = left.data; p < last; p++) {
2494            if (*p >= 'A' && *p <= 'Z') {
2495                *p |= 0x20;
2496                continue;
2497            }
2498
2499            if ((*p >= 'a' && *p <= 'z')
2500                 || (*p >= '0' && *p <= '9')
2501                 || *p == '$' || *p == '{' || *p == '}' || *p == '_'
2502                 || *p == '"' || *p == '\'')
2503            {
2504                continue;
2505            }
2506
2507            break;
2508        }
2509
2510        left.len = p - left.data;
2511
2512        while (p < last && *p == ' ') {
2513            p++;
2514        }
2515
2516        flags = 0;
2517
2518        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2519                       "left: \"%V\"", &left);
2520
2521        rc = ngx_http_ssi_evaluate_string(r, ctx, &left, flags);
2522
2523        if (rc != NGX_OK) {
2524            return rc;
2525        }
2526
2527        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2528                       "evaluted left: \"%V\"", &left);
2529
2530        if (p == last) {
2531            if (left.len) {
2532                ctx->output = 1;
2533                ctx->output_chosen = 1;
2534
2535            } else {
2536                ctx->output = 0;
2537            }
2538
2539            ctx->conditional = NGX_HTTP_SSI_COND_IF;
2540
2541            return NGX_OK;
2542        }
2543
2544        if (p < last && *p == '=') {
2545            negative = 0;
2546            p++;
2547
2548        } else if (p + 1 < last && *p == '!' && *(p + 1) == '=') {
2549            negative = 1;
2550            p += 2;
2551
2552        } else {
2553            goto invalid_expression;
2554        }
2555
2556        while (p < last && *p == ' ') {
2557            p++;
2558        }
2559
2560        if (p < last - 1 && *p == '/') {
2561            if (*(last - 1) != '/') {
2562                goto invalid_expression;
2563            }
2564
```

```
        noregex = 0;
        flags = NGX_HTTP_SSI_ADD_ZERO;
        last--;
        p++;

    } else {
        noregex = 1;
        flags = 0;

        if (p < last - 1 && p[0] == '\\' && p[1] == '/') {
            p++;
        }
    }

    right.len = last - p;
    right.data = p;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "right: \"%V\"", &right);

    rc = ngx_http_ssi_evaluate_string(r, ctx, &right, flags);

    if (rc != NGX_OK) {
        return rc;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "evaluted right: \"%V\"", &right);

    if (noregex) {
        if (left.len != right.len) {
            rc = -1;

        } else {
            rc = ngx_strncmp(left.data, right.data, right.len);
        }

    } else {
        right.data[right.len] = '\0';

        rc = ngx_http_ssi_regex_match(r, &right, &left);

        if (rc == NGX_OK) {
            rc = 0;
        } else if (rc == NGX_DECLINED) {
            rc = -1;
        } else {
            return rc;
        }
    }

    if ((rc == 0 && !negative) || (rc != 0 && negative)) {
        ctx->output = 1;
        ctx->output_chosen = 1;

    } else {
        ctx->output = 0;
    }

    ctx->conditional = NGX_HTTP_SSI_COND_IF;

    return NGX_OK;

invalid_expression:

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "invalid expression in \"%V\"", expr);

    return NGX_HTTP_SSI_ERROR;
}


static ngx_int_t
ngx_http_ssi_else(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t **params)
{
```

```
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi else");

    if (ctx->output_chosen) {
        ctx->output = 0;
    } else {
        ctx->output = 1;
    }

    ctx->conditional = NGX_HTTP_SSI_COND_ELSE;

    return NGX_OK;
}


static ngx_int_t
ngx_http_ssi_endif(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t **params)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi endif");

    ctx->output = 1;
    ctx->output_chosen = 0;
    ctx->conditional = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_http_ssi_block(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t **params)
{
    ngx_http_ssi_ctx_t    *mctx;
    ngx_http_ssi_block_t  *bl;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi block");

    mctx = ngx_http_get_module_ctx(r->main, ngx_http_ssi_filter_module);

    if (mctx->blocks == NULL) {
        mctx->blocks = ngx_array_create(r->pool, 4,
                                        sizeof(ngx_http_ssi_block_t));
        if (mctx->blocks == NULL) {
            return NGX_HTTP_SSI_ERROR;
        }
    }

    bl = ngx_array_push(mctx->blocks);
    if (bl == NULL) {
        return NGX_HTTP_SSI_ERROR;
    }

    bl->name = *params[NGX_HTTP_SSI_BLOCK_NAME];
    bl->bufs = NULL;
    bl->count = 0;

    ctx->output = 0;
    ctx->block = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_ssi_endblock(ngx_http_request_t *r, ngx_http_ssi_ctx_t *ctx,
    ngx_str_t **params)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "ssi endblock");

    ctx->output = 1;
    ctx->block = 0;

```

```
    return NGX_OK;
}


static ngx_int_t
ngx_http_ssi_date_gmt_local_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t gmt)
{
    ngx_http_ssi_ctx_t  *ctx;
    ngx_time_t          *tp;
    ngx_str_t           *timefmt;
    struct tm            tm;
    char                 buf[NGX_HTTP_SSI_DATE_LEN];

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    tp = ngx_timeofday();

    ctx = ngx_http_get_module_ctx(r, ngx_http_ssi_filter_module);

    timefmt = ctx ? &ctx->timefmt : &ngx_http_ssi_timefmt;

    if (timefmt->len == sizeof("%s") - 1
        && timefmt->data[0] == '%' && timefmt->data[1] == 's')
    {
        v->data = ngx_pnalloc(r->pool, NGX_TIME_T_LEN);
        if (v->data == NULL) {
            return NGX_ERROR;
        }

        v->len = ngx_sprintf(v->data, "%T", tp->sec) - v->data;

        return NGX_OK;
    }

    if (gmt) {
        ngx_libc_gmtime(tp->sec, &tm);
    } else {
        ngx_libc_localtime(tp->sec, &tm);
    }

    v->len = strftime(buf, NGX_HTTP_SSI_DATE_LEN,
                      (char *) timefmt->data, &tm);
    if (v->len == 0) {
        return NGX_ERROR;
    }

    v->data = ngx_pnalloc(r->pool, v->len);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    ngx_memcpy(v->data, buf, v->len);

    return NGX_OK;
}


static ngx_int_t
ngx_http_ssi_preconfiguration(ngx_conf_t *cf)
{
    ngx_int_t                  rc;
    ngx_http_variable_t       *var, *v;
    ngx_http_ssi_command_t    *cmd;
    ngx_http_ssi_main_conf_t  *smcf;

    for (v = ngx_http_ssi_vars; v->name.len; v++) {
        var = ngx_http_add_variable(cf, &v->name, v->flags);
        if (var == NULL) {
            return NGX_ERROR;
        }

        var->get_handler = v->get_handler;
        var->data = v->data;
```

```
    }

    smcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_ssi_filter_module);

    for (cmd = ngx_http_ssi_commands; cmd->name.len; cmd++) {
        rc = ngx_hash_add_key(&smcf->commands, &cmd->name, cmd,
                              NGX_HASH_READONLY_KEY);

        if (rc == NGX_OK) {
            continue;
        }

        if (rc == NGX_BUSY) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "conflicting SSI command \"%V\"", &cmd->name);
        }

        return NGX_ERROR;
    }

    return NGX_OK;
}


static void *
ngx_http_ssi_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_ssi_main_conf_t  *smcf;

    smcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_ssi_main_conf_t));
    if (smcf == NULL) {
        return NULL;
    }

    smcf->commands.pool = cf->pool;
    smcf->commands.temp_pool = cf->temp_pool;

    if (ngx_hash_keys_array_init(&smcf->commands, NGX_HASH_SMALL) != NGX_OK) {
        return NULL;
    }

    return smcf;
}


static char *
ngx_http_ssi_init_main_conf(ngx_conf_t *cf, void *conf)
{
    ngx_http_ssi_main_conf_t *smcf = conf;

    ngx_hash_init_t  hash;

    hash.hash = &smcf->hash;
    hash.key = ngx_hash_key;
    hash.max_size = 1024;
    hash.bucket_size = ngx_cacheline_size;
    hash.name = "ssi_command_hash";
    hash.pool = cf->pool;
    hash.temp_pool = NULL;

    if (ngx_hash_init(&hash, smcf->commands.keys.elts,
                      smcf->commands.keys.nelts)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static void *
ngx_http_ssi_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_ssi_loc_conf_t  *slcf;

```

```
2869     slcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_ssi_loc_conf_t));
2870     if (slcf == NULL) {
2871         return NULL;
2872     }
2873
2874     /*
2875      * set by ngx_pcalloc():
2876      *
2877      *     conf->types = { NULL };
2878      *     conf->types_keys = NULL;
2879      */
2880
2881     slcf->enable = NGX_CONF_UNSET;
2882     slcf->silent_errors = NGX_CONF_UNSET;
2883     slcf->ignore_recycled_buffers = NGX_CONF_UNSET;
2884     slcf->last_modified = NGX_CONF_UNSET;
2885
2886     slcf->min_file_chunk = NGX_CONF_UNSET_SIZE;
2887     slcf->value_len = NGX_CONF_UNSET_SIZE;
2888
2889     return slcf;
2890 }
2891
2892
2893 static char *
2894 ngx_http_ssi_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
2895 {
2896     ngx_http_ssi_loc_conf_t *prev = parent;
2897     ngx_http_ssi_loc_conf_t *conf = child;
2898
2899     ngx_conf_merge_value(conf->enable, prev->enable, 0);
2900     ngx_conf_merge_value(conf->silent_errors, prev->silent_errors, 0);
2901     ngx_conf_merge_value(conf->ignore_recycled_buffers,
2902                          prev->ignore_recycled_buffers, 0);
2903     ngx_conf_merge_value(conf->last_modified, prev->last_modified, 0);
2904
2905     ngx_conf_merge_size_value(conf->min_file_chunk, prev->min_file_chunk, 1024);
2906     ngx_conf_merge_size_value(conf->value_len, prev->value_len, 255);
2907
2908     if (ngx_http_merge_types(cf, &conf->types_keys, &conf->types,
2909                              &prev->types_keys, &prev->types,
2910                              ngx_http_html_default_types)
2911         != NGX_OK)
2912     {
2913         return NGX_CONF_ERROR;
2914     }
2915
2916     return NGX_CONF_OK;
2917 }
2918
2919
2920 static ngx_int_t
2921 ngx_http_ssi_filter_init(ngx_conf_t *cf)
2922 {
2923     ngx_http_next_header_filter = ngx_http_top_header_filter;
2924     ngx_http_top_header_filter = ngx_http_ssi_header_filter;
2925
2926     ngx_http_next_body_filter = ngx_http_top_body_filter;
2927     ngx_http_top_body_filter = ngx_http_ssi_body_filter;
2928
2929     return NGX_OK;
2930 }
```

One Level Up Top Level

src/http/modules/ngx_http_ssi_filter_module.h - nginx-1.7.10

Data types defined

- ngx_http_ssi_command_pt
- ngx_http_ssi_command_t
- ngx_http_ssi_ctx_t
- ngx_http_ssi_main_conf_t
- ngx_http_ssi_param_t

Macros defined

- NGX_HTTP_SSI_COMMAND_LEN
- NGX_HTTP_SSI_COND_ELSE
- NGX_HTTP_SSI_COND_IF
- NGX_HTTP_SSI_ENTITY_ENCODING
- NGX_HTTP_SSI_MAX_PARAMS
- NGX_HTTP_SSI_NO_ENCODING
- NGX_HTTP_SSI_PARAMS_N
- NGX_HTTP_SSI_PARAM_LEN
- NGX_HTTP_SSI_URL_ENCODING
- _NGX_HTTP_SSI_FILTER_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_HTTP_SSI_FILTER_H_INCLUDED_
9  #define _NGX_HTTP_SSI_FILTER_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_http.h>
15
16
17  #define NGX_HTTP_SSI_MAX_PARAMS       16
18
19  #define NGX_HTTP_SSI_COMMAND_LEN      32
20  #define NGX_HTTP_SSI_PARAM_LEN        32
21  #define NGX_HTTP_SSI_PARAMS_N         4
22
23
24  #define NGX_HTTP_SSI_COND_IF          1
25  #define NGX_HTTP_SSI_COND_ELSE        2
26
27
```

```
#define NGX_HTTP_SSI_NO_ENCODING      0
#define NGX_HTTP_SSI_URL_ENCODING     1
#define NGX_HTTP_SSI_ENTITY_ENCODING  2


typedef struct {
    ngx_hash_t                hash;
    ngx_hash_keys_arrays_t    commands;
} ngx_http_ssi_main_conf_t;


typedef struct {
    ngx_buf_t                *buf;

    u_char                   *pos;
    u_char                   *copy_start;
    u_char                   *copy_end;

    ngx_uint_t                key;
    ngx_str_t                 command;
    ngx_array_t               params;
    ngx_table_elt_t          *param;
    ngx_table_elt_t           params_array[NGX_HTTP_SSI_PARAMS_N];

    ngx_chain_t              *in;
    ngx_chain_t              *out;
    ngx_chain_t             **last_out;
    ngx_chain_t              *busy;
    ngx_chain_t              *free;

    ngx_uint_t                state;
    ngx_uint_t                saved_state;
    size_t                    saved;
    size_t                    looked;

    size_t                    value_len;

    ngx_list_t               *variables;
    ngx_array_t              *blocks;

#if (NGX_PCRE)
    ngx_uint_t                ncaptures;
    int                      *captures;
    u_char                   *captures_data;
#endif

    unsigned                  conditional:2;
    unsigned                  encoding:2;
    unsigned                  block:1;
    unsigned                  output:1;
    unsigned                  output_chosen:1;

    ngx_http_request_t       *wait;
    void                     *value_buf;
    ngx_str_t                 timefmt;
    ngx_str_t                 errmsg;
} ngx_http_ssi_ctx_t;


typedef ngx_int_t (*ngx_http_ssi_command_pt) (ngx_http_request_t *r,
    ngx_http_ssi_ctx_t *ctx, ngx_str_t **);


typedef struct {
    ngx_str_t                 name;
    ngx_uint_t                index;

    unsigned                  mandatory:1;
    unsigned                  multiple:1;
} ngx_http_ssi_param_t;


 typedef struct {
     ngx_str_t                 name;
     ngx_http_ssi_command_pt   handler;
     ngx_http_ssi_param_t     *params;
```

```
104
105     unsigned                  conditional:2;
106     unsigned                  block:1;
107     unsigned                  flush:1;
108 } ngx_http_ssi_command_t;
109
110
111 extern ngx_module_t  ngx_http_ssi_filter_module;
112
113
114 #endif /* _NGX_HTTP_SSI_FILTER_H_INCLUDED_ */
```

One Level Up Top Level

src/http/ngx_http_postpone_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_body_filter
- ngx_http_postpone_filter_module
- ngx_http_postpone_filter_module_ctx

Functions defined

- ngx_http_postpone_filter
- ngx_http_postpone_filter_add
- ngx_http_postpone_filter_init

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static ngx_int_t ngx_http_postpone_filter_add(ngx_http_request_t *r,
14      ngx_chain_t *in);
15  static ngx_int_t ngx_http_postpone_filter_init(ngx_conf_t *cf);
16
17
18  static ngx_http_module_t  ngx_http_postpone_filter_module_ctx = {
19      NULL,                                  /* preconfiguration */
20      ngx_http_postpone_filter_init,         /* postconfiguration */
21
22      NULL,                                  /* create main configuration */
23      NULL,                                  /* init main configuration */
24
25      NULL,                                  /* create server configuration */
26      NULL,                                  /* merge server configuration */
27
28      NULL,                                  /* create location configuration */
29      NULL                                   /* merge location configuration */
30  };
31
32
33  ngx_module_t  ngx_http_postpone_filter_module = {
34      NGX_MODULE_V1,
35      &ngx_http_postpone_filter_module_ctx,  /* module context */
36      NULL,                                  /* module directives */
37      NGX_HTTP_MODULE,                       /* module type */
38      NULL,                                  /* init master */
39      NULL,                                  /* init module */
40      NULL,                                  /* init process */
41      NULL,                                  /* init thread */
42      NULL,                                  /* exit thread */
43      NULL,                                  /* exit process */
44      NULL,                                  /* exit master */
45      NGX_MODULE_V1_PADDING
46  };
47
48
```

```
49  static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;
50
51
52  static ngx_int_t
53  ngx_http_postpone_filter(ngx_http_request_t *r, ngx_chain_t *in)
54  {
55      ngx_connection_t              *c;
56      ngx_http_postponed_request_t  *pr;
57
58      c = r->connection;
59
60      ngx_log_debug3(NGX_LOG_DEBUG_HTTP, c->log, 0,
61                     "http postpone filter \"%V?%V\" %p", &r->uri, &r->args, in);
62
63      if (r != c->data) {
64
65          if (in) {
66              ngx_http_postpone_filter_add(r, in);
67              return NGX_OK;
68          }
69
70  #if 0
71          /* TODO: SSI may pass NULL */
72          ngx_log_error(NGX_LOG_ALERT, c->log, 0,
73                        "http postpone filter NULL inactive request");
74  #endif
75
76          return NGX_OK;
77      }
78
79      if (r->postponed == NULL) {
80
81          if (in || c->buffered) {
82              return ngx_http_next_body_filter(r->main, in);
83          }
84
85          return NGX_OK;
86      }
87
88      if (in) {
89          ngx_http_postpone_filter_add(r, in);
90      }
91
92      do {
93          pr = r->postponed;
94
95          if (pr->request) {
96
97              ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
98                             "http postpone filter wake \"%V?%V\"",
99                             &pr->request->uri, &pr->request->args);
100
101             r->postponed = pr->next;
102
103             c->data = pr->request;
104
105             return ngx_http_post_request(pr->request, NULL);
106         }
107
108         if (pr->out == NULL) {
109             ngx_log_error(NGX_LOG_ALERT, c->log, 0,
110                           "http postpone filter NULL output");
111
112         } else {
113             ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
114                            "http postpone filter output \"%V?%V\"",
115                            &r->uri, &r->args);
116
117             if (ngx_http_next_body_filter(r->main, pr->out) == NGX_ERROR) {
118                 return NGX_ERROR;
119             }
120         }
121
122         r->postponed = pr->next;
123
124     } while (r->postponed);
```

```
125
126     return NGX_OK;
127 }
128
129
130 static ngx_int_t
131 ngx_http_postpone_filter_add(ngx_http_request_t *r, ngx_chain_t *in)
132 {
133     ngx_http_postponed_request_t  *pr, **ppr;
134
135     if (r->postponed) {
136         for (pr = r->postponed; pr->next; pr = pr->next) { /* void */ }
137
138         if (pr->request == NULL) {
139             goto found;
140         }
141
142         ppr = &pr->next;
143
144     } else {
145         ppr = &r->postponed;
146     }
147
148     pr = ngx_palloc(r->pool, sizeof(ngx_http_postponed_request_t));
149     if (pr == NULL) {
150         return NGX_ERROR;
151     }
152
153     *ppr = pr;
154
155     pr->request = NULL;
156     pr->out = NULL;
157     pr->next = NULL;
158
159 found:
160
161     if (ngx_chain_add_copy(r->pool, &pr->out, in) == NGX_OK) {
162         return NGX_OK;
163     }
164
165     return NGX_ERROR;
166 }
167
168
169 static ngx_int_t
170 ngx_http_postpone_filter_init(ngx_conf_t *cf)
171 {
172     ngx_http_next_body_filter = ngx_http_top_body_filter;
173     ngx_http_top_body_filter = ngx_http_postpone_filter;
174
175     return NGX_OK;
176 }
```

One Level Up Top Level

src/http/modules/ngx_http_degradation_module.c - nginx-1.7.10

Global variables defined

- ngx_http_degradation_commands
- ngx_http_degradation_module
- ngx_http_degradation_module_ctx
- ngx_http_degrade

Data types defined

- ngx_http_degradation_loc_conf_t
- ngx_http_degradation_main_conf_t

Functions defined

- ngx_http_degradation
- ngx_http_degradation_create_loc_conf
- ngx_http_degradation_create_main_conf
- ngx_http_degradation_handler
- ngx_http_degradation_init
- ngx_http_degradation_merge_loc_conf
- ngx_http_degraded

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      size_t      sbrk_size;
15  } ngx_http_degradation_main_conf_t;
16
17
18  typedef struct {
19      ngx_uint_t  degrade;
20  } ngx_http_degradation_loc_conf_t;
21
22
23  static ngx_conf_enum_t  ngx_http_degrade[] = {
24      { ngx_string("204"), 204 },
25      { ngx_string("444"), 444 },
26      { ngx_null_string, 0 }
27  };
28
```

```
29
30  static void *ngx_http_degradation_create_main_conf(ngx_conf_t *cf);
31  static void *ngx_http_degradation_create_loc_conf(ngx_conf_t *cf);
32  static char *ngx_http_degradation_merge_loc_conf(ngx_conf_t *cf, void *parent,
33      void *child);
34  static char *ngx_http_degradation(ngx_conf_t *cf, ngx_command_t *cmd,
35      void *conf);
36  static ngx_int_t ngx_http_degradation_init(ngx_conf_t *cf);
37
38
39  static ngx_command_t  ngx_http_degradation_commands[] = {
40
41      { ngx_string("degradation"),
42        NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
43        ngx_http_degradation,
44        NGX_HTTP_MAIN_CONF_OFFSET,
45        0,
46        NULL },
47
48      { ngx_string("degrade"),
49        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
50        ngx_conf_set_enum_slot,
51        NGX_HTTP_LOC_CONF_OFFSET,
52        offsetof(ngx_http_degradation_loc_conf_t, degrade),
53        &ngx_http_degrade },
54
55        ngx_null_command
56  };
57
58
59  static ngx_http_module_t  ngx_http_degradation_module_ctx = {
60      NULL,                                  /* preconfiguration */
61      ngx_http_degradation_init,             /* postconfiguration */
62
63      ngx_http_degradation_create_main_conf, /* create main configuration */
64      NULL,                                  /* init main configuration */
65
66      NULL,                                  /* create server configuration */
67      NULL,                                  /* merge server configuration */
68
69      ngx_http_degradation_create_loc_conf,  /* create location configuration */
70      ngx_http_degradation_merge_loc_conf    /* merge location configuration */
71  };
72
73
74  ngx_module_t  ngx_http_degradation_module = {
75      NGX_MODULE_V1,
76      &ngx_http_degradation_module_ctx,      /* module context */
77      ngx_http_degradation_commands,         /* module directives */
78      NGX_HTTP_MODULE,                       /* module type */
79      NULL,                                  /* init master */
80      NULL,                                  /* init module */
81      NULL,                                  /* init process */
82      NULL,                                  /* init thread */
83      NULL,                                  /* exit thread */
84      NULL,                                  /* exit process */
85      NULL,                                  /* exit master */
86      NGX_MODULE_V1_PADDING
87  };
88
89
90  static ngx_int_t
91  ngx_http_degradation_handler(ngx_http_request_t *r)
92  {
93      ngx_http_degradation_loc_conf_t  *dlcf;
94
95      dlcf = ngx_http_get_module_loc_conf(r, ngx_http_degradation_module);
96
97      if (dlcf->degrade && ngx_http_degraded(r)) {
98          return dlcf->degrade;
99      }
100
101      return NGX_DECLINED;
102  }
103
104
```

```c
ngx_uint_t
ngx_http_degraded(ngx_http_request_t *r)
{
    time_t                          now;
    ngx_uint_t                      log;
    static size_t                   sbrk_size;
    static time_t                   sbrk_time;
    ngx_http_degradation_main_conf_t  *dmcf;

    dmcf = ngx_http_get_module_main_conf(r, ngx_http_degradation_module);

    if (dmcf->sbrk_size) {

        log = 0;
        now = ngx_time();

        /* lock mutex */

        if (now != sbrk_time) {

            /*
             * ELF/i386 is loaded at 0x08000000, 128M
             * ELF/amd64 is loaded at 0x00400000, 4M
             *
             * use a function address to subtract the loading address
             */

            sbrk_size = (size_t) sbrk(0) - ((uintptr_t) ngx_palloc & ~0x3FFFFF);
            sbrk_time = now;
            log = 1;
        }

        /* unlock mutex */

        if (sbrk_size >= dmcf->sbrk_size) {
            if (log) {
                ngx_log_error(NGX_LOG_NOTICE, r->connection->log, 0,
                              "degradation sbrk:%uzM",
                              sbrk_size / (1024 * 1024));
            }

            return 1;
        }
    }

    return 0;
}


static void *
ngx_http_degradation_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_degradation_main_conf_t  *dmcf;

    dmcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_degradation_main_conf_t));
    if (dmcf == NULL) {
        return NULL;
    }

    return dmcf;
}


static void *
ngx_http_degradation_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_degradation_loc_conf_t  *conf;

    conf = ngx_palloc(cf->pool, sizeof(ngx_http_degradation_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->degrade = NGX_CONF_UNSET_UINT;

    return conf;
```

```
}


static char *
ngx_http_degradation_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_degradation_loc_conf_t  *prev = parent;
    ngx_http_degradation_loc_conf_t  *conf = child;

    ngx_conf_merge_uint_value(conf->degrade, prev->degrade, 0);

    return NGX_CONF_OK;
}


static char *
ngx_http_degradation(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_degradation_main_conf_t  *dmcf = conf;

    ngx_str_t  *value, s;

    value = cf->args->elts;

    if (ngx_strncmp(value[1].data, "sbrk=", 5) == 0) {

        s.len = value[1].len - 5;
        s.data = value[1].data + 5;

        dmcf->sbrk_size = ngx_parse_size(&s);
        if (dmcf->sbrk_size == (size_t) NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid sbrk size \"%V\"", &value[1]);
            return NGX_CONF_ERROR;
        }

        return NGX_CONF_OK;
    }

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "invalid parameter \"%V\"", &value[1]);

    return NGX_CONF_ERROR;
}


static ngx_int_t
ngx_http_degradation_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt        *h;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_PREACCESS_PHASE].handlers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    *h = ngx_http_degradation_handler;

    return NGX_OK;
}
```

One Level Up Top Level

src/core/ngx_spinlock.c - nginx-1.7.10

Functions defined

- ngx_spinlock

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  void
13  ngx_spinlock(ngx_atomic_t *lock, ngx_atomic_int_t value, ngx_uint_t spin)
14  {
15
16  #if (NGX_HAVE_ATOMIC_OPS)
17
18      ngx_uint_t  i, n;
19
20      for ( ;; ) {
21
22          if (*lock == 0 && ngx_atomic_cmp_set(lock, 0, value)) {
23              return;
24          }
25
26          if (ngx_ncpu > 1) {
27
28              for (n = 1; n < spin; n <<= 1) {
29
30                  for (i = 0; i < n; i++) {
31                      ngx_cpu_pause();
32                  }
33
34                  if (*lock == 0 && ngx_atomic_cmp_set(lock, 0, value)) {
35                      return;
36                  }
37              }
38          }
39
40          ngx_sched_yield();
41      }
42
43  #else
44
45  #if (NGX_THREADS)
46
47  #error ngx_spinlock() or ngx_atomic_cmp_set() are not defined !
48
49  #endif
50
51  #endif
52
53  }
```

One Level Up Top Level

src/core/ngx_syslog.h - nginx-1.7.10

Data types defined

- ngx_syslog_peer_t

Macros defined

- _NGX_SYSLOG_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  */
5
6
7  #ifndef _NGX_SYSLOG_H_INCLUDED_
8  #define _NGX_SYSLOG_H_INCLUDED_
9
10
11 typedef struct {
12     ngx_pool_t       *pool;
13     ngx_uint_t        facility;
14     ngx_uint_t        severity;
15     ngx_str_t         tag;
16
17     ngx_addr_t        server;
18     ngx_connection_t  conn;
19     ngx_uint_t        busy;  /* unsigned busy:1; */
20 } ngx_syslog_peer_t;
21
22
23 char *ngx_syslog_process_conf(ngx_conf_t *cf, ngx_syslog_peer_t *peer);
24 u_char *ngx_syslog_add_header(ngx_syslog_peer_t *peer, u_char *buf);
25 void ngx_syslog_writer(ngx_log_t *log, ngx_uint_t level, u_char *buf,
26     size_t len);
27 ssize_t ngx_syslog_send(ngx_syslog_peer_t *peer, u_char *buf, size_t len);
28
29
30 #endif /* _NGX_SYSLOG_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_syslog.c - nginx-1.7.10

Global variables defined

- facilities
- ngx_syslog_dummy_event
- ngx_syslog_dummy_log
- severities

Functions defined

- ngx_syslog_add_header
- ngx_syslog_cleanup
- ngx_syslog_init_peer
- ngx_syslog_parse_args
- ngx_syslog_process_conf
- ngx_syslog_send
- ngx_syslog_writer

Macros defined

- NGX_SYSLOG_MAX_STR

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  */
5
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9  #include <ngx_event.h>
10
11
12 #define NGX_SYSLOG_MAX_STR                                                   \
13      NGX_MAX_ERROR_STR + sizeof("<255>Jan 01 00:00:00 ") - 1                 \
14      + (NGX_MAXHOSTNAMELEN - 1) + 1 /* space */                             \
15      + 32 /* tag */ + 2 /* colon, space */
16
17
18 static char *ngx_syslog_parse_args(ngx_conf_t *cf, ngx_syslog_peer_t *peer);
19 static ngx_int_t ngx_syslog_init_peer(ngx_syslog_peer_t *peer);
20 static void ngx_syslog_cleanup(void *data);
21
22
23 static char  *facilities[] = {
24     "kern", "user", "mail", "daemon", "auth", "intern", "lpr", "news", "uucp",
25     "clock", "authpriv", "ftp", "ntp", "audit", "alert", "cron", "local0",
26     "local1", "local2", "local3", "local4", "local5", "local6", "local7",
27     NULL
28 };
29
30 /* note 'error/warn' like in nginx.conf, not 'err/warning' */
```

```c
static char  *severities[] = {
    "emerg", "alert", "crit", "error", "warn", "notice", "info", "debug", NULL
};

static ngx_log_t    ngx_syslog_dummy_log;
static ngx_event_t  ngx_syslog_dummy_event;


char *
ngx_syslog_process_conf(ngx_conf_t *cf, ngx_syslog_peer_t *peer)
{
    peer->pool = cf->pool;
    peer->facility = NGX_CONF_UNSET_UINT;
    peer->severity = NGX_CONF_UNSET_UINT;

    if (ngx_syslog_parse_args(cf, peer) != NGX_CONF_OK) {
        return NGX_CONF_ERROR;
    }

    if (peer->server.sockaddr == NULL) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "no syslog server specified");
        return NGX_CONF_ERROR;
    }

    if (peer->facility == NGX_CONF_UNSET_UINT) {
        peer->facility = 23; /* local7 */
    }

    if (peer->severity == NGX_CONF_UNSET_UINT) {
        peer->severity = 6; /* info */
    }

    if (peer->tag.data == NULL) {
        ngx_str_set(&peer->tag, "nginx");
    }

    peer->conn.fd = (ngx_socket_t) -1;

    return NGX_CONF_OK;
}


static char *
ngx_syslog_parse_args(ngx_conf_t *cf, ngx_syslog_peer_t *peer)
{
    u_char      *p, *comma, c;
    size_t       len;
    ngx_str_t   *value;
    ngx_url_t    u;
    ngx_uint_t   i;

    value = cf->args->elts;

    p = value[1].data + sizeof("syslog:") - 1;

    for ( ;; ) {
        comma = (u_char *) ngx_strchr(p, ',');

        if (comma != NULL) {
            len = comma - p;
            *comma = '\0';

        } else {
            len = value[1].data + value[1].len - p;
        }

        if (ngx_strncmp(p, "server=", 7) == 0) {

             if (peer->server.sockaddr != NULL) {
                 ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                    "duplicate syslog \"server\"");
                 return NGX_CONF_ERROR;
             }

             ngx_memzero(&u, sizeof(ngx_url_t));
```

```
            u.url.data = p + 7;
            u.url.len = len - 7;
            u.default_port = 514;

            if (ngx_parse_url(cf->pool, &u) != NGX_OK) {
                if (u.err) {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "%s in syslog server \"%V\"",
                                       u.err, &u.url);
                }

                return NGX_CONF_ERROR;
            }

            peer->server = u.addrs[0];

        } else if (ngx_strncmp(p, "facility=", 9) == 0) {

            if (peer->facility != NGX_CONF_UNSET_UINT) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "duplicate syslog \"facility\"");
                return NGX_CONF_ERROR;
            }

            for (i = 0; facilities[i] != NULL; i++) {

                if (ngx_strcmp(p + 9, facilities[i]) == 0) {
                    peer->facility = i;
                    goto next;
                }
            }

            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "unknown syslog facility \"%s\"", p + 9);
            return NGX_CONF_ERROR;

        } else if (ngx_strncmp(p, "severity=", 9) == 0) {

            if (peer->severity != NGX_CONF_UNSET_UINT) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "duplicate syslog \"severity\"");
                return NGX_CONF_ERROR;
            }

            for (i = 0; severities[i] != NULL; i++) {

                if (ngx_strcmp(p + 9, severities[i]) == 0) {
                    peer->severity = i;
                    goto next;
                }
            }

            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "unknown syslog severity \"%s\"", p + 9);
            return NGX_CONF_ERROR;

        } else if (ngx_strncmp(p, "tag=", 4) == 0) {

            if (peer->tag.data != NULL) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "duplicate syslog \"tag\"");
                return NGX_CONF_ERROR;
            }

            /*
             * RFC 3164: the TAG is a string of ABNF alphanumeric characters
             * that MUST NOT exceed 32 characters.
             */
            if (len - 4 > 32) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "syslog tag length exceeds 32");
                return NGX_CONF_ERROR;
            }

            for (i = 4; i < len; i++) {
```

```
                c = ngx_tolower(p[i]);

                if (c < '0' || (c > '9' && c < 'a' && c != '_') || c > 'z') {
                    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                       "syslog \"tag\" only allows "
                                       "alphanumeric characters "
                                       "and underscore");
                    return NGX_CONF_ERROR;
                }
            }

            peer->tag.data = p + 4;
            peer->tag.len = len - 4;

        } else {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "unknown syslog parameter \"%s\"", p);
            return NGX_CONF_ERROR;
        }

    next:

        if (comma == NULL) {
            break;
        }

        p = comma + 1;
    }

    return NGX_CONF_OK;
}


u_char *
ngx_syslog_add_header(ngx_syslog_peer_t *peer, u_char *buf)
{
    ngx_uint_t  pri;

    pri = peer->facility * 8 + peer->severity;

    return ngx_sprintf(buf, "<%ui>%V %V %V: ", pri, &ngx_cached_syslog_time,
                       &ngx_cycle->hostname, &peer->tag);
}


void
ngx_syslog_writer(ngx_log_t *log, ngx_uint_t level, u_char *buf,
    size_t len)
{
    u_char             *p, msg[NGX_SYSLOG_MAX_STR];
    ngx_uint_t          head_len;
    ngx_syslog_peer_t  *peer;

    peer = log->wdata;

    if (peer->busy) {
        return;
    }

    peer->busy = 1;
    peer->severity = level - 1;

    p = ngx_syslog_add_header(peer, msg);
    head_len = p - msg;

    len -= NGX_LINEFEED_SIZE;

    if (len > NGX_SYSLOG_MAX_STR - head_len) {
        len = NGX_SYSLOG_MAX_STR - head_len;
    }

    p = ngx_snprintf(p, len, "%s", buf);

    (void) ngx_syslog_send(peer, msg, p - msg);

    peer->busy = 0;
```

```
}


ssize_t
ngx_syslog_send(ngx_syslog_peer_t *peer, u_char *buf, size_t len)
{
    ssize_t  n;

    if (peer->conn.fd == (ngx_socket_t) -1) {
        if (ngx_syslog_init_peer(peer) != NGX_OK) {
            return NGX_ERROR;
        }
    }

    /* log syslog socket events with valid log */
    peer->conn.log = ngx_cycle->log;

    if (ngx_send) {
        n = ngx_send(&peer->conn, buf, len);

    } else {
        /* event module has not yet set ngx_io */
        n = ngx_os_io.send(&peer->conn, buf, len);
    }

#if (NGX_HAVE_UNIX_DOMAIN)

    if (n == NGX_ERROR && peer->server.sockaddr->sa_family == AF_UNIX) {

        if (ngx_close_socket(peer->conn.fd) == -1) {
            ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_socket_errno,
                          ngx_close_socket_n " failed");
        }

        peer->conn.fd = (ngx_socket_t) -1;
    }

#endif

    return n;
}


static ngx_int_t
ngx_syslog_init_peer(ngx_syslog_peer_t *peer)
{
    ngx_socket_t         fd;
    ngx_pool_cleanup_t  *cln;

    peer->conn.read = &ngx_syslog_dummy_event;
    peer->conn.write = &ngx_syslog_dummy_event;

    ngx_syslog_dummy_event.log = &ngx_syslog_dummy_log;

    fd = ngx_socket(peer->server.sockaddr->sa_family, SOCK_DGRAM, 0);
    if (fd == (ngx_socket_t) -1) {
        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_socket_errno,
                      ngx_socket_n " failed");
        return NGX_ERROR;
    }

    if (ngx_nonblocking(fd) == -1) {
        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_socket_errno,
                      ngx_nonblocking_n " failed");
        goto failed;
    }

    if (connect(fd, peer->server.sockaddr, peer->server.socklen) == -1) {
        ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_socket_errno,
                      "connect() failed");
        goto failed;
    }

    cln = ngx_pool_cleanup_add(peer->pool, 0);
    if (cln == NULL) {
        goto failed;
```

```
335     }
336
337     cln->data = peer;
338     cln->handler = ngx_syslog_cleanup;
339
340     peer->conn.fd = fd;
341
342     /* UDP sockets are always ready to write */
343     peer->conn.write->ready = 1;
344
345     return NGX_OK;
346
347 failed:
348
349     if (ngx_close_socket(fd) == -1) {
350         ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_socket_errno,
351                       ngx_close_socket_n " failed");
352     }
353
354     return NGX_ERROR;
355 }
356
357
358 static void
359 ngx_syslog_cleanup(void *data)
360 {
361     ngx_syslog_peer_t  *peer = data;
362
363     /* prevents further use of this peer */
364     peer->busy = 1;
365
366     if (peer->conn.fd == (ngx_socket_t) -1) {
367         return;
368     }
369
370     if (ngx_close_socket(peer->conn.fd) == -1) {
371         ngx_log_error(NGX_LOG_ALERT, ngx_cycle->log, ngx_socket_errno,
372                       ngx_close_socket_n " failed");
373     }
374 }
```

One Level Up Top Level

src/event/modules/ngx_devpoll_module.c - nginx-1.7.10

Global variables defined

- change_index
- change_list
- devpoll_name
- dp
- event_list
- max_changes
- nchanges
- nevents
- ngx_devpoll_commands
- ngx_devpoll_module
- ngx_devpoll_module_ctx

Data types defined

- dvpoll
- ngx_devpoll_conf_t

Functions defined

- ngx_devpoll_add_event
- ngx_devpoll_create_conf
- ngx_devpoll_del_event
- ngx_devpoll_done
- ngx_devpoll_init
- ngx_devpoll_init_conf
- ngx_devpoll_process_events
- ngx_devpoll_set_event

Macros defined

- DP_ISPOLLED
- DP_POLL
- POLLREMOVE

Source code

```c

/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_event.h>


 #if (NGX_TEST_BUILD_DEVPOLL)

 /* Solaris declarations */

 #define POLLREMOVE   0x0800
 #define DP_POLL      0xD001
 #define DP_ISPOLLED  0xD002

 struct dvpoll {
     struct pollfd  *dp_fds;
     int             dp_nfds;
     int             dp_timeout;
 };

 #endif


 typedef struct {
     ngx_uint_t      changes;
     ngx_uint_t      events;
 } ngx_devpoll_conf_t;


 static ngx_int_t ngx_devpoll_init(ngx_cycle_t *cycle, ngx_msec_t timer);
 static void ngx_devpoll_done(ngx_cycle_t *cycle);
 static ngx_int_t ngx_devpoll_add_event(ngx_event_t *ev, ngx_int_t event,
     ngx_uint_t flags);
 static ngx_int_t ngx_devpoll_del_event(ngx_event_t *ev, ngx_int_t event,
     ngx_uint_t flags);
 static ngx_int_t ngx_devpoll_set_event(ngx_event_t *ev, ngx_int_t event,
     ngx_uint_t flags);
 static ngx_int_t ngx_devpoll_process_events(ngx_cycle_t *cycle,
     ngx_msec_t timer, ngx_uint_t flags);

 static void *ngx_devpoll_create_conf(ngx_cycle_t *cycle);
 static char *ngx_devpoll_init_conf(ngx_cycle_t *cycle, void *conf);

 static int              dp = -1;
 static struct pollfd   *change_list, *event_list;
 static ngx_uint_t       nchanges, max_changes, nevents;

 static ngx_event_t    **change_index;


 static ngx_str_t      devpoll_name = ngx_string("/dev/poll");

 static ngx_command_t  ngx_devpoll_commands[] = {

     { ngx_string("devpoll_changes"),
       NGX_EVENT_CONF|NGX_CONF_TAKE1,
       ngx_conf_set_num_slot,
       0,
       offsetof(ngx_devpoll_conf_t, changes),
       NULL },

     { ngx_string("devpoll_events"),
       NGX_EVENT_CONF|NGX_CONF_TAKE1,
       ngx_conf_set_num_slot,
       0,
       offsetof(ngx_devpoll_conf_t, events),
       NULL },

       ngx_null_command
 };
```

```
77
78
79  ngx_event_module_t  ngx_devpoll_module_ctx = {
80      &devpoll_name,
81      ngx_devpoll_create_conf,               /* create configuration */
82      ngx_devpoll_init_conf,                 /* init configuration */
83
84      {
85          ngx_devpoll_add_event,             /* add an event */
86          ngx_devpoll_del_event,             /* delete an event */
87          ngx_devpoll_add_event,             /* enable an event */
88          ngx_devpoll_del_event,             /* disable an event */
89          NULL,                              /* add an connection */
90          NULL,                              /* delete an connection */
91          NULL,                              /* process the changes */
92          ngx_devpoll_process_events,        /* process the events */
93          ngx_devpoll_init,                  /* init the events */
94          ngx_devpoll_done,                  /* done the events */
95      }
96
97  };
98
99  ngx_module_t  ngx_devpoll_module = {
100      NGX_MODULE_V1,
101      &ngx_devpoll_module_ctx,               /* module context */
102      ngx_devpoll_commands,                  /* module directives */
103      NGX_EVENT_MODULE,                      /* module type */
104      NULL,                                  /* init master */
105      NULL,                                  /* init module */
106      NULL,                                  /* init process */
107      NULL,                                  /* init thread */
108      NULL,                                  /* exit thread */
109      NULL,                                  /* exit process */
110      NULL,                                  /* exit master */
111      NGX_MODULE_V1_PADDING
112  };
113
114
115  static ngx_int_t
116  ngx_devpoll_init(ngx_cycle_t *cycle, ngx_msec_t timer)
117  {
118      size_t               n;
119      ngx_devpoll_conf_t  *dpcf;
120
121      dpcf = ngx_event_get_conf(cycle->conf_ctx, ngx_devpoll_module);
122
123      if (dp == -1) {
124          dp = open("/dev/poll", O_RDWR);
125
126          if (dp == -1) {
127              ngx_log_error(NGX_LOG_EMERG, cycle->log, ngx_errno,
128                            "open(/dev/poll) failed");
129              return NGX_ERROR;
130          }
131      }
132
133      if (max_changes < dpcf->changes) {
134          if (nchanges) {
135              n = nchanges * sizeof(struct pollfd);
136              if (write(dp, change_list, n) != (ssize_t) n) {
137                  ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
138                                "write(/dev/poll) failed");
139                  return NGX_ERROR;
140              }
141
142              nchanges = 0;
143          }
144
145          if (change_list) {
146              ngx_free(change_list);
147          }
148
149          change_list = ngx_alloc(sizeof(struct pollfd) * dpcf->changes,
150                                  cycle->log);
151          if (change_list == NULL) {
152              return NGX_ERROR;
```

```
        }

        if (change_index) {
            ngx_free(change_index);
        }

        change_index = ngx_alloc(sizeof(ngx_event_t *) * dpcf->changes,
                                 cycle->log);
        if (change_index == NULL) {
            return NGX_ERROR;
        }
    }

    max_changes = dpcf->changes;

    if (nevents < dpcf->events) {
        if (event_list) {
            ngx_free(event_list);
        }

        event_list = ngx_alloc(sizeof(struct pollfd) * dpcf->events,
                               cycle->log);
        if (event_list == NULL) {
            return NGX_ERROR;
        }
    }

    nevents = dpcf->events;

    ngx_io = ngx_os_io;

    ngx_event_actions = ngx_devpoll_module_ctx.actions;

    ngx_event_flags = NGX_USE_LEVEL_EVENT|NGX_USE_FD_EVENT;

    return NGX_OK;
}


static void
ngx_devpoll_done(ngx_cycle_t *cycle)
{
    if (close(dp) == -1) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                      "close(/dev/poll) failed");
    }

    dp = -1;

    ngx_free(change_list);
    ngx_free(event_list);
    ngx_free(change_index);

    change_list = NULL;
    event_list = NULL;
    change_index = NULL;
    max_changes = 0;
    nchanges = 0;
    nevents = 0;
}


static ngx_int_t
ngx_devpoll_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
#if (NGX_DEBUG)
    ngx_connection_t *c;
#endif

#if (NGX_READ_EVENT != POLLIN)
    event = (event == NGX_READ_EVENT) ? POLLIN : POLLOUT;
#endif

#if (NGX_DEBUG)
    c = ev->data;
    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
```

```
                   "devpoll add event: fd:%d ev:%04Xi", c->fd, event);
#endif

    ev->active = 1;

    return ngx_devpoll_set_event(ev, event, 0);
}


static ngx_int_t
ngx_devpoll_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_event_t       *e;
    ngx_connection_t  *c;

    c = ev->data;

#if (NGX_READ_EVENT != POLLIN)
    event = (event == NGX_READ_EVENT) ? POLLIN : POLLOUT;
#endif

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "devpoll del event: fd:%d ev:%04Xi", c->fd, event);

    if (ngx_devpoll_set_event(ev, POLLREMOVE, flags) == NGX_ERROR) {
        return NGX_ERROR;
    }

    ev->active = 0;

    if (flags & NGX_CLOSE_EVENT) {
        e = (event == POLLIN) ? c->write : c->read;

        if (e) {
            e->active = 0;
        }

        return NGX_OK;
    }

    /* restore the pair event if it exists */

    if (event == POLLIN) {
        e = c->write;
        event = POLLOUT;

    } else {
        e = c->read;
        event = POLLIN;
    }

    if (e && e->active) {
        return ngx_devpoll_set_event(e, event, 0);
    }

    return NGX_OK;
}


static ngx_int_t
ngx_devpoll_set_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    size_t             n;
    ngx_connection_t  *c;

    c = ev->data;

    ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "devpoll fd:%d ev:%04Xi fl:%04Xi", c->fd, event, flags);

    if (nchanges >= max_changes) {
        ngx_log_error(NGX_LOG_WARN, ev->log, 0,
                      "/dev/pool change list is filled up");

        n = nchanges * sizeof(struct pollfd);
        if (write(dp, change_list, n) != (ssize_t) n) {
```

```
            ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
                          "write(/dev/poll) failed");
            return NGX_ERROR;
        }

        nchanges = 0;
    }

    change_list[nchanges].fd = c->fd;
    change_list[nchanges].events = (short) event;
    change_list[nchanges].revents = 0;

    change_index[nchanges] = ev;
    ev->index = nchanges;

    nchanges++;

    if (flags & NGX_CLOSE_EVENT) {
        n = nchanges * sizeof(struct pollfd);
        if (write(dp, change_list, n) != (ssize_t) n) {
            ngx_log_error(NGX_LOG_ALERT, ev->log, ngx_errno,
                          "write(/dev/poll) failed");
            return NGX_ERROR;
        }

        nchanges = 0;
    }

    return NGX_OK;
}


ngx_int_t
ngx_devpoll_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
    ngx_uint_t flags)
{
    int                 events, revents, rc;
    size_t              n;
    ngx_fd_t            fd;
    ngx_err_t           err;
    ngx_int_t           i;
    ngx_uint_t          level, instance;
    ngx_event_t        *rev, *wev;
    ngx_queue_t        *queue;
    ngx_connection_t   *c;
    struct pollfd       pfd;
    struct dvpoll       dvp;

    /* NGX_TIMER_INFINITE == INFTIM */

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "devpoll timer: %M", timer);

    if (nchanges) {
        n = nchanges * sizeof(struct pollfd);
        if (write(dp, change_list, n) != (ssize_t) n) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                          "write(/dev/poll) failed");
            return NGX_ERROR;
        }

        nchanges = 0;
    }

    dvp.dp_fds = event_list;
    dvp.dp_nfds = (int) nevents;
    dvp.dp_timeout = timer;
    events = ioctl(dp, DP_POLL, &dvp);

    err = (events == -1) ? ngx_errno : 0;

    if (flags & NGX_UPDATE_TIME || ngx_event_timer_alarm) {
        ngx_time_update();
    }

    if (err) {
```

```
        if (err == NGX_EINTR) {

            if (ngx_event_timer_alarm) {
                ngx_event_timer_alarm = 0;
                return NGX_OK;
            }

            level = NGX_LOG_INFO;

        } else {
            level = NGX_LOG_ALERT;
        }

        ngx_log_error(level, cycle->log, err, "ioctl(DP_POLL) failed");
        return NGX_ERROR;
    }

    if (events == 0) {
        if (timer != NGX_TIMER_INFINITE) {
            return NGX_OK;
        }

        ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                      "ioctl(DP_POLL) returned no events without timeout");
        return NGX_ERROR;
    }

    for (i = 0; i < events; i++) {

        fd = event_list[i].fd;
        revents = event_list[i].revents;

        c = ngx_cycle->files[fd];

        if (c == NULL || c->fd == -1) {

            pfd.fd = fd;
            pfd.events = 0;
            pfd.revents = 0;

            rc = ioctl(dp, DP_ISPOLLED, &pfd);

            switch (rc) {

            case -1:
                ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                    "ioctl(DP_ISPOLLED) failed for socket %d, event %04Xd",
                    fd, revents);
                break;

            case 0:
                ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                    "phantom event %04Xd for closed and removed socket %d",
                    revents, fd);
                break;

            default:
                ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                    "unexpected event %04Xd for closed and removed socket %d, ",
                    "ioctl(DP_ISPOLLED) returned rc:%d, fd:%d, event %04Xd",
                    revents, fd, rc, pfd.fd, pfd.revents);

                pfd.fd = fd;
                pfd.events = POLLREMOVE;
                pfd.revents = 0;

                if (write(dp, &pfd, sizeof(struct pollfd))
                    != (ssize_t) sizeof(struct pollfd))
                {
                    ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                                  "write(/dev/poll) for %d failed", fd);
                }

                if (close(fd) == -1) {
                    ngx_log_error(NGX_LOG_ALERT, cycle->log, ngx_errno,
                                  "close(%d) failed", fd);
```

```
                }

                break;
            }

            continue;
        }

        ngx_log_debug3(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "devpoll: fd:%d, ev:%04Xd, rev:%04Xd",
                       fd, event_list[i].events, revents);

        if (revents & (POLLERR|POLLHUP|POLLNVAL)) {
            ngx_log_debug3(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "ioctl(DP_POLL) error fd:%d ev:%04Xd rev:%04Xd",
                           fd, event_list[i].events, revents);
        }

        if (revents & ~(POLLIN|POLLOUT|POLLERR|POLLHUP|POLLNVAL)) {
            ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                           "strange ioctl(DP_POLL) events "
                           "fd:%d ev:%04Xd rev:%04Xd",
                           fd, event_list[i].events, revents);
        }

        if ((revents & (POLLERR|POLLHUP|POLLNVAL))
             && (revents & (POLLIN|POLLOUT)) == 0)
        {
            /*
             * if the error events were returned without POLLIN or POLLOUT,
             * then add these flags to handle the events at least in one
             * active handler
             */

            revents |= POLLIN|POLLOUT;
        }

        rev = c->read;

        if ((revents & POLLIN) && rev->active) {
            rev->ready = 1;

            if (flags & NGX_POST_EVENTS) {
                queue = rev->accept ? &ngx_posted_accept_events
                                    : &ngx_posted_events;

                ngx_post_event(rev, queue);

            } else {
                instance = rev->instance;

                rev->handler(rev);

                if (c->fd == -1 || rev->instance != instance) {
                    continue;
                }
            }
        }

        wev = c->write;

        if ((revents & POLLOUT) && wev->active) {
            wev->ready = 1;

            if (flags & NGX_POST_EVENTS) {
                ngx_post_event(wev, &ngx_posted_events);

            } else {
                wev->handler(wev);
            }
        }
    }

    return NGX_OK;
}
```

```
533
534 static void *
535 ngx_devpoll_create_conf(ngx_cycle_t *cycle)
536 {
537     ngx_devpoll_conf_t  *dpcf;
538
539     dpcf = ngx_palloc(cycle->pool, sizeof(ngx_devpoll_conf_t));
540     if (dpcf == NULL) {
541         return NULL;
542     }
543
544     dpcf->changes = NGX_CONF_UNSET;
545     dpcf->events = NGX_CONF_UNSET;
546
547     return dpcf;
548 }
549
550
551 static char *
552 ngx_devpoll_init_conf(ngx_cycle_t *cycle, void *conf)
553 {
554     ngx_devpoll_conf_t *dpcf = conf;
555
556     ngx_conf_init_uint_value(dpcf->changes, 32);
557     ngx_conf_init_uint_value(dpcf->events, 32);
558
559     return NGX_CONF_OK;
560 }
```

One Level Up Top Level

src/event/ngx_event_busy_lock.c - nginx-1.7.10

Functions defined

- ngx_event_busy_lock
- ngx_event_busy_lock_cacheable
- ngx_event_busy_lock_cancel
- ngx_event_busy_lock_handler
- ngx_event_busy_lock_look_cacheable
- ngx_event_busy_lock_posted_handler
- ngx_event_busy_unlock

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  static ngx_int_t ngx_event_busy_lock_look_cacheable(ngx_event_busy_lock_t *bl,
14      ngx_event_busy_lock_ctx_t *ctx);
15  static void ngx_event_busy_lock_handler(ngx_event_t *ev);
16  static void ngx_event_busy_lock_posted_handler(ngx_event_t *ev);
17
18
19  /*
20  * NGX_OK:      the busy lock is held
21  * NGX_AGAIN:   the all busy locks are held but we will wait the specified time
22  * NGX_BUSY:    ctx->timer == 0: there are many the busy locks
23  *              ctx->timer != 0: there are many the waiting locks
24  */
25
26  ngx_int_t
27  ngx_event_busy_lock(ngx_event_busy_lock_t *bl, ngx_event_busy_lock_ctx_t *ctx)
28  {
29      ngx_int_t  rc;
30
31      ngx_mutex_lock(bl->mutex);
32
33      ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ctx->event->log, 0,
34                     "event busy lock: b:%d mb:%d",
35                     bl->busy, bl->max_busy);
36
37      if (bl->busy < bl->max_busy) {
38          bl->busy++;
39
40          rc = NGX_OK;
41
42      } else if (ctx->timer && bl->waiting < bl->max_waiting) {
43          bl->waiting++;
44          ngx_add_timer(ctx->event, ctx->timer);
45          ctx->event->handler = ngx_event_busy_lock_handler;
46
47          if (bl->events) {
48              bl->last->next = ctx;
49
```

```
        } else {
            bl->events = ctx;
        }

        bl->last = ctx;

        rc = NGX_AGAIN;

    } else {
        rc = NGX_BUSY;
    }

    ngx_mutex_unlock(bl->mutex);

    return rc;
}


ngx_int_t
ngx_event_busy_lock_cacheable(ngx_event_busy_lock_t *bl,
    ngx_event_busy_lock_ctx_t *ctx)
{
    ngx_int_t  rc;

    ngx_mutex_lock(bl->mutex);

    rc = ngx_event_busy_lock_look_cacheable(bl, ctx);

    ngx_log_debug3(NGX_LOG_DEBUG_EVENT, ctx->event->log, 0,
                   "event busy lock: %d w:%d mw:%d",
                   rc, bl->waiting, bl->max_waiting);

    /*
     * NGX_OK:     no the same request, there is free slot and we locked it
     * NGX_BUSY:   no the same request and there is no free slot
     * NGX_AGAIN:  the same request is processing
     */

    if (rc == NGX_AGAIN) {

        if (ctx->timer && bl->waiting < bl->max_waiting) {
            bl->waiting++;
            ngx_add_timer(ctx->event, ctx->timer);
            ctx->event->handler = ngx_event_busy_lock_handler;

            if (bl->events == NULL) {
                bl->events = ctx;
            } else {
                bl->last->next = ctx;
            }
             bl->last = ctx;

         } else {
             rc = NGX_BUSY;
        }
     }

     ngx_mutex_unlock(bl->mutex);

     return rc;
}


 void
 ngx_event_busy_unlock(ngx_event_busy_lock_t *bl,
     ngx_event_busy_lock_ctx_t *ctx)
 {
     ngx_event_t                *ev;
     ngx_event_busy_lock_ctx_t  *wakeup;

     ngx_mutex_lock(bl->mutex);

     if (bl->events) {
         wakeup = bl->events;
         bl->events = bl->events->next;

```

```
    } else {
        wakeup = NULL;
        bl->busy--;
    }

    /*
     * MP: all ctx's and their queue must be in shared memory,
     *     each ctx has pid to wake up
     */

    if (wakeup == NULL) {
        ngx_mutex_unlock(bl->mutex);
        return;
    }

    if (ctx->md5) {
        for (wakeup = bl->events; wakeup; wakeup = wakeup->next) {
            if (wakeup->md5 == NULL || wakeup->slot != ctx->slot) {
                continue;
            }

            wakeup->handler = ngx_event_busy_lock_posted_handler;
            wakeup->cache_updated = 1;

            ev = wakeup->event;

            ngx_post_event(ev, &ngx_posted_events);
        }

        ngx_mutex_unlock(bl->mutex);

    } else {
        bl->waiting--;

        ngx_mutex_unlock(bl->mutex);

        wakeup->handler = ngx_event_busy_lock_posted_handler;
        wakeup->locked = 1;

        ev = wakeup->event;

        if (ev->timer_set) {
            ngx_del_timer(ev);
        }

        ngx_post_event(ev, &ngx_posted_events);
    }
}


void
ngx_event_busy_lock_cancel(ngx_event_busy_lock_t *bl,
    ngx_event_busy_lock_ctx_t *ctx)
{
    ngx_event_busy_lock_ctx_t  *c, *p;

    ngx_mutex_lock(bl->mutex);

    bl->waiting--;

    if (ctx == bl->events) {
        bl->events = ctx->next;

    } else {
        p = bl->events;
        for (c = bl->events->next; c; c = c->next) {
            if (c == ctx) {
                p->next = ctx->next;
                break;
            }
            p = c;
        }
    }

    ngx_mutex_unlock(bl->mutex);
}
```

```
202
203
204 static ngx_int_t
205 ngx_event_busy_lock_look_cacheable(ngx_event_busy_lock_t *bl,
206     ngx_event_busy_lock_ctx_t *ctx)
207 {
208     ngx_int_t    free;
209     ngx_uint_t   i, bit, cacheable, mask;
210
211     bit = 0;
212     cacheable = 0;
213     free = -1;
214
215 #if (NGX_SUPPRESS_WARN)
216     mask = 0;
217 #endif
218
219     for (i = 0; i < bl->max_busy; i++) {
220
221         if ((bit & 7) == 0) {
222             mask = bl->md5_mask[i / 8];
223         }
224
225         if (mask & 1) {
226             if (ngx_memcmp(&bl->md5[i * 16], ctx->md5, 16) == 0) {
227                 ctx->waiting = 1;
228                 ctx->slot = i;
229                 return NGX_AGAIN;
230             }
231             cacheable++;
232
233         } else if (free == -1) {
234             free = i;
235         }
236
237         if (cacheable == bl->cacheable) {
238             if (free == -1 && cacheable < bl->max_busy) {
239                 free = i + 1;
240             }
241
242             break;
243         }
244
245         mask >>= 1;
246         bit++;
247     }
248
249     if (free == -1) {
250         return NGX_BUSY;
251     }
252
253 #if 0
254     if (bl->busy == bl->max_busy) {
255         return NGX_BUSY;
256     }
257 #endif
258
259     ngx_memcpy(&bl->md5[free * 16], ctx->md5, 16);
260     bl->md5_mask[free / 8] |= 1 << (free & 7);
261     ctx->slot = free;
262
263     bl->cacheable++;
264     bl->busy++;
265
266     return NGX_OK;
267 }
268
269
270 static void
271 ngx_event_busy_lock_handler(ngx_event_t *ev)
272 {
273     ev->handler = ngx_event_busy_lock_posted_handler;
274
275     ngx_post_event(ev, &ngx_posted_events);
276 }
277
```

```
278
279 static void
280 ngx_event_busy_lock_posted_handler(ngx_event_t *ev)
281 {
282     ngx_event_busy_lock_ctx_t  *ctx;
283
284     ctx = ev->data;
285     ctx->handler(ev);
286 }
```

One Level Up Top Level

src/event/ngx_event_busy_lock.h - nginx-1.7.10

Data types defined

- ngx_event_busy_lock_ctx_s
- ngx_event_busy_lock_ctx_t
- ngx_event_busy_lock_t

Macros defined

- _NGX_EVENT_BUSY_LOCK_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_EVENT_BUSY_LOCK_H_INCLUDED_
9  #define _NGX_EVENT_BUSY_LOCK_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14  #include <ngx_event.h>
15
16  typedef struct ngx_event_busy_lock_ctx_s  ngx_event_busy_lock_ctx_t;
17
18  struct ngx_event_busy_lock_ctx_s {
19      ngx_event_t                *event;
20      ngx_event_handler_pt        handler;
21      void                       *data;
22      ngx_msec_t                  timer;
23
24      unsigned                    locked:1;
25      unsigned                    waiting:1;
26      unsigned                    cache_updated:1;
27
28      char                       *md5;
29      ngx_int_t                   slot;
30
31      ngx_event_busy_lock_ctx_t  *next;
32  };
33
34
35  typedef struct {
36      u_char                     *md5_mask;
37      char                       *md5;
38      ngx_uint_t                  cacheable;
39
40      ngx_uint_t                  busy;
41      ngx_uint_t                  max_busy;
42
43      ngx_uint_t                  waiting;
44      ngx_uint_t                  max_waiting;
45
46      ngx_event_busy_lock_ctx_t  *events;
47      ngx_event_busy_lock_ctx_t  *last;
48
49  #if (NGX_THREADS)
50      ngx_mutex_t                *mutex;
51  #endif
52  } ngx_event_busy_lock_t;
```

```
53
54
55  ngx_int_t ngx_event_busy_lock(ngx_event_busy_lock_t *bl,
56      ngx_event_busy_lock_ctx_t *ctx);
57  ngx_int_t ngx_event_busy_lock_cacheable(ngx_event_busy_lock_t *bl,
58      ngx_event_busy_lock_ctx_t *ctx);
59  void ngx_event_busy_unlock(ngx_event_busy_lock_t *bl,
60      ngx_event_busy_lock_ctx_t *ctx);
61  void ngx_event_busy_lock_cancel(ngx_event_busy_lock_t *bl,
62      ngx_event_busy_lock_ctx_t *ctx);
63
64
65  #endif /* _NGX_EVENT_BUSY_LOCK_H_INCLUDED_ */
```

One Level Up Top Level

src/event/modules/ngx_poll_module.c - nginx-1.7.10

Global variables defined

- event_list
- nevents
- ngx_poll_module
- ngx_poll_module_ctx
- poll_name

Functions defined

- ngx_poll_add_event
- ngx_poll_del_event
- ngx_poll_done
- ngx_poll_init
- ngx_poll_init_conf
- ngx_poll_process_events

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  static ngx_int_t ngx_poll_init(ngx_cycle_t *cycle, ngx_msec_t timer);
14  static void ngx_poll_done(ngx_cycle_t *cycle);
15  static ngx_int_t ngx_poll_add_event(ngx_event_t *ev, ngx_int_t event,
16      ngx_uint_t flags);
17  static ngx_int_t ngx_poll_del_event(ngx_event_t *ev, ngx_int_t event,
18      ngx_uint_t flags);
19  static ngx_int_t ngx_poll_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
20      ngx_uint_t flags);
21  static char *ngx_poll_init_conf(ngx_cycle_t *cycle, void *conf);
22
23
24  static struct pollfd  *event_list;
25  static ngx_uint_t      nevents;
26
27
28  static ngx_str_t    poll_name = ngx_string("poll");
29
30  ngx_event_module_t  ngx_poll_module_ctx = {
31      &poll_name,
32      NULL,                                  /* create configuration */
33      ngx_poll_init_conf,                    /* init configuration */
34
35      {
36          ngx_poll_add_event,                /* add an event */
```

```
        ngx_poll_del_event,              /* delete an event */
        ngx_poll_add_event,              /* enable an event */
        ngx_poll_del_event,              /* disable an event */
        NULL,                            /* add an connection */
        NULL,                            /* delete an connection */
        NULL,                            /* process the changes */
        ngx_poll_process_events,         /* process the events */
        ngx_poll_init,                   /* init the events */
        ngx_poll_done                    /* done the events */
    }

};

ngx_module_t  ngx_poll_module = {
    NGX_MODULE_V1,
    &ngx_poll_module_ctx,                /* module context */
    NULL,                                /* module directives */
    NGX_EVENT_MODULE,                    /* module type */
    NULL,                                /* init master */
    NULL,                                /* init module */
    NULL,                                /* init process */
    NULL,                                /* init thread */
    NULL,                                /* exit thread */
    NULL,                                /* exit process */
    NULL,                                /* exit master */
    NGX_MODULE_V1_PADDING
};



static ngx_int_t
ngx_poll_init(ngx_cycle_t *cycle, ngx_msec_t timer)
{
    struct pollfd   *list;

    if (event_list == NULL) {
        nevents = 0;
    }

    if (ngx_process >= NGX_PROCESS_WORKER
        || cycle->old_cycle == NULL
        || cycle->old_cycle->connection_n < cycle->connection_n)
    {
        list = ngx_alloc(sizeof(struct pollfd) * cycle->connection_n,
                         cycle->log);
        if (list == NULL) {
            return NGX_ERROR;
        }

        if (event_list) {
            ngx_memcpy(list, event_list, sizeof(ngx_event_t *) * nevents);
            ngx_free(event_list);
        }

        event_list = list;
    }

    ngx_io = ngx_os_io;

    ngx_event_actions = ngx_poll_module_ctx.actions;

    ngx_event_flags = NGX_USE_LEVEL_EVENT|NGX_USE_FD_EVENT;

     return NGX_OK;
}


static void
ngx_poll_done(ngx_cycle_t *cycle)
{
    ngx_free(event_list);

    event_list = NULL;
}
```

```c
static ngx_int_t
ngx_poll_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_event_t       *e;
    ngx_connection_t  *c;

    c = ev->data;

    ev->active = 1;

    if (ev->index != NGX_INVALID_INDEX) {
        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "poll event fd:%d ev:%i is already set", c->fd, event);
        return NGX_OK;
    }

    if (event == NGX_READ_EVENT) {
        e = c->write;
#if (NGX_READ_EVENT != POLLIN)
        event = POLLIN;
#endif

    } else {
        e = c->read;
#if (NGX_WRITE_EVENT != POLLOUT)
        event = POLLOUT;
#endif
    }

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "poll add event: fd:%d ev:%i", c->fd, event);

    if (e == NULL || e->index == NGX_INVALID_INDEX) {
        event_list[nevents].fd = c->fd;
        event_list[nevents].events = (short) event;
        event_list[nevents].revents = 0;

        ev->index = nevents;
        nevents++;

    } else {
        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                       "poll add index: %i", e->index);

        event_list[e->index].events |= (short) event;
        ev->index = e->index;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_poll_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_event_t       *e;
    ngx_connection_t  *c;

    c = ev->data;

    ev->active = 0;

    if (ev->index == NGX_INVALID_INDEX) {
        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "poll event fd:%d ev:%i is already deleted",
                      c->fd, event);
        return NGX_OK;
    }

    if (event == NGX_READ_EVENT) {
        e = c->write;
#if (NGX_READ_EVENT != POLLIN)
        event = POLLIN;
#endif

    } else {
```

```
        e = c->read;
#if (NGX_WRITE_EVENT != POLLOUT)
        event = POLLOUT;
#endif
    }

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "poll del event: fd:%d ev:%i", c->fd, event);

    if (e == NULL || e->index == NGX_INVALID_INDEX) {
        nevents--;

        if (ev->index < nevents) {

            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                           "index: copy event %ui to %i", nevents, ev->index);

            event_list[ev->index] = event_list[nevents];

            c = ngx_cycle->files[event_list[nevents].fd];

            if (c->fd == -1) {
                ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                              "unexpected last event");

            } else {
                if (c->read->index == nevents) {
                    c->read->index = ev->index;
                }

                if (c->write->index == nevents) {
                    c->write->index = ev->index;
                }
            }
        }

    } else {
        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                       "poll del index: %i", e->index);

        event_list[e->index].events &= (short) ~event;
    }

    ev->index = NGX_INVALID_INDEX;

    return NGX_OK;
}


static ngx_int_t
ngx_poll_process_events(ngx_cycle_t *cycle, ngx_msec_t timer, ngx_uint_t flags)
{
    int                ready, revents;
    ngx_err_t          err;
    ngx_uint_t         i, found, level;
    ngx_event_t       *ev;
    ngx_queue_t       *queue;
    ngx_connection_t  *c;

    /* NGX_TIMER_INFINITE == INFTIM */

#if (NGX_DEBUG0)
    if (cycle->log->log_level & NGX_LOG_DEBUG_ALL) {
        for (i = 0; i < nevents; i++) {
            ngx_log_debug3(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "poll: %ui: fd:%d ev:%04Xd",
                           i, event_list[i].fd, event_list[i].events);
        }
    }
#endif

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0, "poll timer: %M", timer);

    ready = poll(event_list, (u_int) nevents, (int) timer);

    err = (ready == -1) ? ngx_errno : 0;
```

```
265
266     if (flags & NGX_UPDATE_TIME || ngx_event_timer_alarm) {
267         ngx_time_update();
268     }
269
270     ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
271                    "poll ready %d of %ui", ready, nevents);
272
273     if (err) {
274         if (err == NGX_EINTR) {
275
276             if (ngx_event_timer_alarm) {
277                 ngx_event_timer_alarm = 0;
278                 return NGX_OK;
279             }
280
281             level = NGX_LOG_INFO;
282
283         } else {
284             level = NGX_LOG_ALERT;
285         }
286
287         ngx_log_error(level, cycle->log, err, "poll() failed");
288         return NGX_ERROR;
289     }
290
291     if (ready == 0) {
292         if (timer != NGX_TIMER_INFINITE) {
293             return NGX_OK;
294         }
295
296         ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
297                       "poll() returned no events without timeout");
298         return NGX_ERROR;
299     }
300
301     for (i = 0; i < nevents && ready; i++) {
302
303         revents = event_list[i].revents;
304
305 #if 1
306         ngx_log_debug4(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
307                        "poll: %ui: fd:%d ev:%04Xd rev:%04Xd",
308                        i, event_list[i].fd, event_list[i].events, revents);
309 #else
310         if (revents) {
311             ngx_log_debug4(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
312                            "poll: %ui: fd:%d ev:%04Xd rev:%04Xd",
313                            i, event_list[i].fd, event_list[i].events, revents);
314         }
315 #endif
316
317         if (revents & POLLNVAL) {
318             ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
319                           "poll() error fd:%d ev:%04Xd rev:%04Xd",
320                           event_list[i].fd, event_list[i].events, revents);
321         }
322
323         if (revents & ~(POLLIN|POLLOUT|POLLERR|POLLHUP|POLLNVAL)) {
324             ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
325                           "strange poll() events fd:%d ev:%04Xd rev:%04Xd",
326                           event_list[i].fd, event_list[i].events, revents);
327         }
328
329         if (event_list[i].fd == -1) {
330             /*
331              * the disabled event, a workaround for our possible bug,
332              * see the comment below
333              */
334             continue;
335         }
336
337         c = ngx_cycle->files[event_list[i].fd];
338
339         if (c->fd == -1) {
340             ngx_log_error(NGX_LOG_ALERT, cycle->log, 0, "unexpected event");
```

```
341
342             /*
343              * it is certainly our fault and it should be investigated,
344              * in the meantime we disable this event to avoid a CPU spinning
345              */
346
347             if (i == nevents - 1) {
348                 nevents--;
349             } else {
350                 event_list[i].fd = -1;
351             }
352
353             continue;
354         }
355
356         if ((revents & (POLLERR|POLLHUP|POLLNVAL))
357              && (revents & (POLLIN|POLLOUT)) == 0)
358         {
359             /*
360              * if the error events were returned without POLLIN or POLLOUT,
361              * then add these flags to handle the events at least in one
362              * active handler
363              */
364
365             revents |= POLLIN|POLLOUT;
366         }
367
368         found = 0;
369
370         if ((revents & POLLIN) && c->read->active) {
371             found = 1;
372
373             ev = c->read;
374             ev->ready = 1;
375
376             queue = ev->accept ? &ngx_posted_accept_events
377                                : &ngx_posted_events;
378
379             ngx_post_event(ev, queue);
380         }
381
382         if ((revents & POLLOUT) && c->write->active) {
383             found = 1;
384
385             ev = c->write;
386             ev->ready = 1;
387
388             ngx_post_event(ev, &ngx_posted_events);
389         }
390
391         if (found) {
392             ready--;
393             continue;
394         }
395     }
396
397     if (ready != 0) {
398         ngx_log_error(NGX_LOG_ALERT, cycle->log, 0, "poll ready != events");
399     }
400
401     return NGX_OK;
402 }
403
404
405 static char *
406 ngx_poll_init_conf(ngx_cycle_t *cycle, void *conf)
407 {
408     ngx_event_conf_t  *ecf;
409
410     ecf = ngx_event_get_conf(cycle->conf_ctx, ngx_event_core_module);
411
412     if (ecf->use != ngx_poll_module.ctx_index) {
413         return NGX_CONF_OK;
414     }
415
416 #if (NGX_THREADS)
```

```
417
418     ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
419                   "poll() is not supported in the threaded mode");
420     return NGX_CONF_ERROR;
421
422 #else
423
424     return NGX_CONF_OK;
425
426 #endif
427 }
```

One Level Up Top Level

src/os/unix/ngx_linux_init.c - nginx-1.7.10

Global variables defined

- ngx_linux_io
- ngx_linux_kern_osrelease
- ngx_linux_kern_ostype
- ngx_linux_rtsig_max

Functions defined

- ngx_os_specific_init
- ngx_os_specific_status

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 u_char  ngx_linux_kern_ostype[50];
13 u_char  ngx_linux_kern_osrelease[50];
14
15 int     ngx_linux_rtsig_max;
16
17
18 static ngx_os_io_t ngx_linux_io = {
19     ngx_unix_recv,
20     ngx_readv_chain,
21     ngx_udp_unix_recv,
22     ngx_unix_send,
23 #if (NGX_HAVE_SENDFILE)
24     ngx_linux_sendfile_chain,
25     NGX_IO_SENDFILE
26 #else
27     ngx_writev_chain,
28     0
29 #endif
30 };
31
32
33 ngx_int_t
34 ngx_os_specific_init(ngx_log_t *log)
35 {
36     struct utsname  u;
37
38     if (uname(&u) == -1) {
39         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno, "uname() failed");
40         return NGX_ERROR;
41     }
42
43     (void) ngx_cpystrn(ngx_linux_kern_ostype, (u_char *) u.sysname,
44                        sizeof(ngx_linux_kern_ostype));
45
46     (void) ngx_cpystrn(ngx_linux_kern_osrelease, (u_char *) u.release,
47                        sizeof(ngx_linux_kern_osrelease));
48
```

```
#if (NGX_HAVE_RTSIG)
    {
    int        name[2];
    size_t     len;
    ngx_err_t  err;

    name[0] = CTL_KERN;
    name[1] = KERN_RTSIGMAX;
    len = sizeof(ngx_linux_rtsig_max);

    if (sysctl(name, 2, &ngx_linux_rtsig_max, &len, NULL, 0) == -1) {
        err = ngx_errno;

        if (err != NGX_ENOTDIR && err != NGX_ENOSYS) {
            ngx_log_error(NGX_LOG_ALERT, log, err,
                          "sysctl(KERN_RTSIGMAX) failed");

            return NGX_ERROR;
        }

        ngx_linux_rtsig_max = 0;
    }

    }
#endif

    ngx_os_io = ngx_linux_io;

    return NGX_OK;
}


void
ngx_os_specific_status(ngx_log_t *log)
{
    ngx_log_error(NGX_LOG_NOTICE, log, 0, "OS: %s %s",
                  ngx_linux_kern_ostype, ngx_linux_kern_osrelease);

#if (NGX_HAVE_RTSIG)
    ngx_log_error(NGX_LOG_NOTICE, log, 0, "sysctl(KERN_RTSIGMAX): %d",
                  ngx_linux_rtsig_max);
#endif
}
```

One Level Up Top Level

src/os/unix/ngx_linux_sendfile_chain.c - nginx-1.7.10

Functions defined

- ngx_linux_sendfile_chain

Macros defined

- NGX_SENDFILE_MAXSIZE

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  /*
14  * On Linux up to 2.4.21 sendfile() (syscall #187) works with 32-bit
15  * offsets only, and the including <sys/sendfile.h> breaks the compiling,
16  * if off_t is 64 bit wide.  So we use own sendfile() definition, where offset
17  * parameter is int32_t, and use sendfile() for the file parts below 2G only,
18  * see src/os/unix/ngx_linux_config.h
19  *
20  * Linux 2.4.21 has the new sendfile64() syscall #239.
21  *
22  * On Linux up to 2.6.16 sendfile() does not allow to pass the count parameter
23  * more than 2G-1 bytes even on 64-bit platforms: it returns EINVAL,
24  * so we limit it to 2G-1 bytes.
25  */
26
27  #define NGX_SENDFILE_MAXSIZE  2147483647L
28
29
30  ngx_chain_t *
31  ngx_linux_sendfile_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
32  {
33      int            tcp_nodelay;
34      off_t          send, prev_send, sent;
35      size_t         file_size;
36      ssize_t        n;
37      ngx_err_t      err;
38      ngx_buf_t     *file;
39      ngx_uint_t     eintr;
40      ngx_event_t   *wev;
41      ngx_chain_t   *cl;
42      ngx_iovec_t    header;
43      struct iovec   headers[NGX_IOVS_PREALLOCATE];
44  #if (NGX_HAVE_SENDFILE64)
45      off_t          offset;
46  #else
47      int32_t        offset;
48  #endif
49
50      wev = c->write;
51
52      if (!wev->ready) {
53          return in;
54      }
55
56
57      /* the maximum limit size is 2G-1 - the page size */
```

```
58
59      if (limit == 0 || limit > (off_t) (NGX_SENDFILE_MAXSIZE - ngx_pagesize)) {
60          limit = NGX_SENDFILE_MAXSIZE - ngx_pagesize;
61      }
62
63
64      send = 0;
65
66      header.iovs = headers;
67      header.nalloc = NGX_IOVS_PREALLOCATE;
68
69      for ( ;; ) {
70          eintr = 0;
71          prev_send = send;
72
73          /* create the iovec and coalesce the neighbouring bufs */
74
75          cl = ngx_output_chain_to_iovec(&header, in, limit - send, c->log);
76
77          if (cl == NGX_CHAIN_ERROR) {
78              return NGX_CHAIN_ERROR;
79          }
80
81          send += header.size;
82
83          /* set TCP_CORK if there is a header before a file */
84
85          if (c->tcp_nopush == NGX_TCP_NOPUSH_UNSET
86              && header.count != 0
87              && cl
88              && cl->buf->in_file)
89          {
90              /* the TCP_CORK and TCP_NODELAY are mutually exclusive */
91
92              if (c->tcp_nodelay == NGX_TCP_NODELAY_SET) {
93
94                  tcp_nodelay = 0;
95
96                  if (setsockopt(c->fd, IPPROTO_TCP, TCP_NODELAY,
97                                 (const void *) &tcp_nodelay, sizeof(int)) == -1)
98                  {
99                      err = ngx_socket_errno;
100
101                     /*
102                      * there is a tiny chance to be interrupted, however,
103                      * we continue a processing with the TCP_NODELAY
104                      * and without the TCP_CORK
105                      */
106
107                     if (err != NGX_EINTR) {
108                         wev->error = 1;
109                         ngx_connection_error(c, err,
110                                              "setsockopt(TCP_NODELAY) failed");
111                         return NGX_CHAIN_ERROR;
112                     }
113
114                 } else {
115                     c->tcp_nodelay = NGX_TCP_NODELAY_UNSET;
116
117                     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
118                                    "no tcp_nodelay");
119                 }
120             }
121
122             if (c->tcp_nodelay == NGX_TCP_NODELAY_UNSET) {
123
124                 if (ngx_tcp_nopush(c->fd) == NGX_ERROR) {
125                     err = ngx_socket_errno;
126
127                     /*
128                      * there is a tiny chance to be interrupted, however,
129                      * we continue a processing without the TCP_CORK
130                      */
131
132                     if (err != NGX_EINTR) {
133                         wev->error = 1;
```

```
                        ngx_connection_error(c, err,
                                             ngx_tcp_nopush_n " failed");
                        return NGX_CHAIN_ERROR;
                    }

                } else {
                    c->tcp_nopush = NGX_TCP_NOPUSH_SET;

                    ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, 0,
                                   "tcp_nopush");
                }
            }
        }

        /* get the file buf */

        if (header.count == 0 && cl && cl->buf->in_file && send < limit) {
            file = cl->buf;

            /* coalesce the neighbouring file bufs */

            file_size = (size_t) ngx_chain_coalesce_file(&cl, limit - send);

            send += file_size;
#if 1
            if (file_size == 0) {
                ngx_debug_point();
                return NGX_CHAIN_ERROR;
            }
#endif
#if (NGX_HAVE_SENDFILE64)
            offset = file->file_pos;
#else
            offset = (int32_t) file->file_pos;
#endif

            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, c->log, 0,
                           "sendfile: @%O %uz", file->file_pos, file_size);

            n = sendfile(c->fd, file->file->fd, &offset, file_size);

            if (n == -1) {
                err = ngx_errno;

                switch (err) {
                case NGX_EAGAIN:
                    break;

                case NGX_EINTR:
                    eintr = 1;
                    break;

                default:
                    wev->error = 1;
                    ngx_connection_error(c, err, "sendfile() failed");
                    return NGX_CHAIN_ERROR;
                }

                ngx_log_debug0(NGX_LOG_DEBUG_EVENT, c->log, err,
                               "sendfile() is not ready");
            }

            sent = n > 0 ? n : 0;

            ngx_log_debug4(NGX_LOG_DEBUG_EVENT, c->log, 0,
                           "sendfile: %z, @%O %O:%uz",
                           n, file->file_pos, sent, file_size);

        } else {
            n = ngx_writev(c, &header);

            if (n == NGX_ERROR) {
                return NGX_CHAIN_ERROR;
            }

            sent = (n == NGX_AGAIN) ? 0 : n;
```

```
210         }
211
212         c->sent += sent;
213
214         in = ngx_chain_update_sent(in, sent);
215
216         if (eintr) {
217             send = prev_send;
218             continue;
219         }
220
221         if (send - prev_send != sent) {
222             wev->ready = 0;
223             return in;
224         }
225
226         if (send >= limit || in == NULL) {
227             return in;
228         }
229     }
230 }
```

One Level Up Top Level

src/event/modules/ngx_select_module.c - nginx-1.7.10

Global variables defined

- event_index
- master_read_fd_set
- master_write_fd_set
- max_fd
- nevents
- ngx_select_module
- ngx_select_module_ctx
- select_name
- work_read_fd_set
- work_write_fd_set

Functions defined

- ngx_select_add_event
- ngx_select_del_event
- ngx_select_done
- ngx_select_init
- ngx_select_init_conf
- ngx_select_process_events
- ngx_select_repair_fd_sets

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  static ngx_int_t ngx_select_init(ngx_cycle_t *cycle, ngx_msec_t timer);
14  static void ngx_select_done(ngx_cycle_t *cycle);
15  static ngx_int_t ngx_select_add_event(ngx_event_t *ev, ngx_int_t event,
16      ngx_uint_t flags);
17  static ngx_int_t ngx_select_del_event(ngx_event_t *ev, ngx_int_t event,
18      ngx_uint_t flags);
19  static ngx_int_t ngx_select_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
20      ngx_uint_t flags);
21  static void ngx_select_repair_fd_sets(ngx_cycle_t *cycle);
22  static char *ngx_select_init_conf(ngx_cycle_t *cycle, void *conf);
23
```

```
24
25  static fd_set         master_read_fd_set;
26  static fd_set         master_write_fd_set;
27  static fd_set         work_read_fd_set;
28  static fd_set         work_write_fd_set;
29
30  static ngx_int_t      max_fd;
31  static ngx_uint_t     nevents;
32
33  static ngx_event_t  **event_index;
34
35
36  static ngx_str_t    select_name = ngx_string("select");
37
38  ngx_event_module_t  ngx_select_module_ctx = {
39      &select_name,
40      NULL,                                  /* create configuration */
41      ngx_select_init_conf,                  /* init configuration */
42
43      {
44          ngx_select_add_event,              /* add an event */
45          ngx_select_del_event,              /* delete an event */
46          ngx_select_add_event,              /* enable an event */
47          ngx_select_del_event,              /* disable an event */
48          NULL,                              /* add an connection */
49          NULL,                              /* delete an connection */
50          NULL,                              /* process the changes */
51          ngx_select_process_events,         /* process the events */
52          ngx_select_init,                   /* init the events */
53          ngx_select_done                    /* done the events */
54      }
55
56  };
57
58  ngx_module_t  ngx_select_module = {
59      NGX_MODULE_V1,
60      &ngx_select_module_ctx,                /* module context */
61      NULL,                                  /* module directives */
62      NGX_EVENT_MODULE,                      /* module type */
63      NULL,                                  /* init master */
64      NULL,                                  /* init module */
65      NULL,                                  /* init process */
66      NULL,                                  /* init thread */
67      NULL,                                  /* exit thread */
68      NULL,                                  /* exit process */
69      NULL,                                  /* exit master */
70      NGX_MODULE_V1_PADDING
71  };
72
73
74  static ngx_int_t
75  ngx_select_init(ngx_cycle_t *cycle, ngx_msec_t timer)
76  {
77      ngx_event_t  **index;
78
79      if (event_index == NULL) {
80          FD_ZERO(&master_read_fd_set);
81          FD_ZERO(&master_write_fd_set);
82          nevents = 0;
83      }
84
85      if (ngx_process >= NGX_PROCESS_WORKER
86          || cycle->old_cycle == NULL
87          || cycle->old_cycle->connection_n < cycle->connection_n)
88      {
89          index = ngx_alloc(sizeof(ngx_event_t *) * 2 * cycle->connection_n,
90                            cycle->log);
91          if (index == NULL) {
92              return NGX_ERROR;
93          }
94
95          if (event_index) {
96              ngx_memcpy(index, event_index, sizeof(ngx_event_t *) * nevents);
97              ngx_free(event_index);
98          }
99
```

```
        event_index = index;
    }

    ngx_io = ngx_os_io;

    ngx_event_actions = ngx_select_module_ctx.actions;

    ngx_event_flags = NGX_USE_LEVEL_EVENT;

    max_fd = -1;

    return NGX_OK;
}


static void
ngx_select_done(ngx_cycle_t *cycle)
{
    ngx_free(event_index);

    event_index = NULL;
}


static ngx_int_t
ngx_select_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_connection_t  *c;

    c = ev->data;

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "select add event fd:%d ev:%i", c->fd, event);

    if (ev->index != NGX_INVALID_INDEX) {
        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "select event fd:%d ev:%i is already set", c->fd, event);
        return NGX_OK;
    }

    if ((event == NGX_READ_EVENT && ev->write)
        || (event == NGX_WRITE_EVENT && !ev->write))
    {
        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "invalid select %s event fd:%d ev:%i",
                      ev->write ? "write" : "read", c->fd, event);
        return NGX_ERROR;
    }

    if (event == NGX_READ_EVENT) {
        FD_SET(c->fd, &master_read_fd_set);

    } else if (event == NGX_WRITE_EVENT) {
        FD_SET(c->fd, &master_write_fd_set);
    }

    if (max_fd != -1 && max_fd < c->fd) {
        max_fd = c->fd;
    }

    ev->active = 1;

    event_index[nevents] = ev;
    ev->index = nevents;
    nevents++;

    return NGX_OK;
}


static ngx_int_t
ngx_select_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_event_t       *e;
    ngx_connection_t  *c;

```

```
    c = ev->data;

    ev->active = 0;

    if (ev->index == NGX_INVALID_INDEX) {
        return NGX_OK;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "select del event fd:%d ev:%i", c->fd, event);

    if (event == NGX_READ_EVENT) {
        FD_CLR(c->fd, &master_read_fd_set);

    } else if (event == NGX_WRITE_EVENT) {
        FD_CLR(c->fd, &master_write_fd_set);
    }

    if (max_fd == c->fd) {
        max_fd = -1;
    }

    if (ev->index < --nevents) {
        e = event_index[nevents];
        event_index[ev->index] = e;
        e->index = ev->index;
    }

    ev->index = NGX_INVALID_INDEX;

    return NGX_OK;
}


static ngx_int_t
ngx_select_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
    ngx_uint_t flags)
{
    int               ready, nready;
    ngx_err_t         err;
    ngx_uint_t        i, found;
    ngx_event_t       *ev;
    ngx_queue_t       *queue;
    struct timeval    tv, *tp;
    ngx_connection_t  *c;

    if (max_fd == -1) {
        for (i = 0; i < nevents; i++) {
            c = event_index[i]->data;
            if (max_fd < c->fd) {
                max_fd = c->fd;
            }
        }

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "change max_fd: %i", max_fd);
    }

#if (NGX_DEBUG)
    if (cycle->log->log_level & NGX_LOG_DEBUG_ALL) {
        for (i = 0; i < nevents; i++) {
            ev = event_index[i];
            c = ev->data;
            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "select event: fd:%d wr:%d", c->fd, ev->write);
        }

        ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                       "max_fd: %i", max_fd);
    }
#endif

    if (timer == NGX_TIMER_INFINITE) {
        tp = NULL;

    } else {
```

```
252         tv.tv_sec = (long) (timer / 1000);
253         tv.tv_usec = (long) ((timer % 1000) * 1000);
254         tp = &tv;
255     }
256
257     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
258                    "select timer: %M", timer);
259
260     work_read_fd_set = master_read_fd_set;
261     work_write_fd_set = master_write_fd_set;
262
263     ready = select(max_fd + 1, &work_read_fd_set, &work_write_fd_set, NULL, tp);
264
265     err = (ready == -1) ? ngx_errno : 0;
266
267     if (flags & NGX_UPDATE_TIME || ngx_event_timer_alarm) {
268         ngx_time_update();
269     }
270
271     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
272                    "select ready %d", ready);
273
274     if (err) {
275         ngx_uint_t  level;
276
277         if (err == NGX_EINTR) {
278
279             if (ngx_event_timer_alarm) {
280                 ngx_event_timer_alarm = 0;
281                 return NGX_OK;
282             }
283
284             level = NGX_LOG_INFO;
285
286         } else {
287             level = NGX_LOG_ALERT;
288         }
289
290         ngx_log_error(level, cycle->log, err, "select() failed");
291
292         if (err == NGX_EBADF) {
293             ngx_select_repair_fd_sets(cycle);
294         }
295
296         return NGX_ERROR;
297     }
298
299     if (ready == 0) {
300         if (timer != NGX_TIMER_INFINITE) {
301             return NGX_OK;
302         }
303
304         ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
305                       "select() returned no events without timeout");
306         return NGX_ERROR;
307     }
308
309     nready = 0;
310
311     for (i = 0; i < nevents; i++) {
312         ev = event_index[i];
313         c = ev->data;
314         found = 0;
315
316         if (ev->write) {
317             if (FD_ISSET(c->fd, &work_write_fd_set)) {
318                 found = 1;
319                 ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
320                                "select write %d", c->fd);
321             }
322
323         } else {
324             if (FD_ISSET(c->fd, &work_read_fd_set)) {
325                 found = 1;
326                 ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
327                                "select read %d", c->fd);
```

```
            }
        }

        if (found) {
            ev->ready = 1;

            queue = ev->accept ? &ngx_posted_accept_events
                               : &ngx_posted_events;

            ngx_post_event(ev, queue);

            nready++;
        }
    }

    if (ready != nready) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                      "select ready != events: %d:%d", ready, nready);

        ngx_select_repair_fd_sets(cycle);
    }

    return NGX_OK;
}


static void
ngx_select_repair_fd_sets(ngx_cycle_t *cycle)
{
    int           n;
    socklen_t     len;
    ngx_err_t     err;
    ngx_socket_t  s;

    for (s = 0; s <= max_fd; s++) {

        if (FD_ISSET(s, &master_read_fd_set) == 0) {
            continue;
        }

        len = sizeof(int);

        if (getsockopt(s, SOL_SOCKET, SO_TYPE, &n, &len) == -1) {
            err = ngx_socket_errno;

            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          "invalid descriptor #%d in read fd_set", s);

            FD_CLR(s, &master_read_fd_set);
        }
    }

    for (s = 0; s <= max_fd; s++) {

        if (FD_ISSET(s, &master_write_fd_set) == 0) {
            continue;
        }

        len = sizeof(int);

        if (getsockopt(s, SOL_SOCKET, SO_TYPE, &n, &len) == -1) {
            err = ngx_socket_errno;

            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          "invalid descriptor #%d in write fd_set", s);

            FD_CLR(s, &master_write_fd_set);
        }
    }

    max_fd = -1;
}


static char *
ngx_select_init_conf(ngx_cycle_t *cycle, void *conf)
```

```
{
    ngx_event_conf_t  *ecf;

    ecf = ngx_event_get_conf(cycle->conf_ctx, ngx_event_core_module);

    if (ecf->use != ngx_select_module.ctx_index) {
        return NGX_CONF_OK;
    }

    /* disable warning: the default FD_SETSIZE is 1024U in FreeBSD 5.x */

    if (cycle->connection_n > FD_SETSIZE) {
        ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
                      "the maximum number of files "
                      "supported by select() is %ud", FD_SETSIZE);
        return NGX_CONF_ERROR;
    }

#if (NGX_THREADS)

    ngx_log_error(NGX_LOG_EMERG, cycle->log, 0,
                  "select() is not supported in the threaded mode");
    return NGX_CONF_ERROR;

#else

    return NGX_CONF_OK;

#endif
}
```

One Level Up Top Level

src/event/ngx_event_mutex.c - nginx-1.7.10

Functions defined

- ngx_event_mutex_timedlock
- ngx_event_mutex_unlock

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  ngx_int_t ngx_event_mutex_timedlock(ngx_event_mutex_t *m, ngx_msec_t timer,
14                                      ngx_event_t *ev)
15  {
16      ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
17                     "lock event mutex %p lock:%XD", m, m->lock);
18
19      if (m->lock) {
20
21          if (m->events == NULL) {
22              m->events = ev;
23
24          } else {
25              m->last->next = ev;
26          }
27
28          m->last = ev;
29          ev->next = NULL;
30
31  #if (NGX_THREADS0)
32          ev->light = 1;
33  #endif
34
35          ngx_add_timer(ev, timer);
36
37          return NGX_AGAIN;
38      }
39
40      m->lock = 1;
41
42      return NGX_OK;
43  }
44
45
46  ngx_int_t ngx_event_mutex_unlock(ngx_event_mutex_t *m, ngx_log_t *log)
47  {
48      ngx_event_t  *ev;
49
50      if (m->lock == 0) {
51          ngx_log_error(NGX_LOG_ALERT, log, 0,
52                        "tring to unlock the free event mutex %p", m);
53          return NGX_ERROR;
54      }
55
56      ngx_log_debug2(NGX_LOG_DEBUG_EVENT, log, 0,
57                     "unlock event mutex %p, next event: %p", m, m->events);
58
59      m->lock = 0;
60
```

```
61      if (m->events) {
62          ev = m->events;
63          m->events = ev->next;
64
65          ev->next = ngx_posted_events;
66          ngx_posted_events = ev;
67      }
68
69      return NGX_OK;
70  }
```

One Level Up Top Level

src/event/modules/ngx_aio_module.c - nginx-1.7.10

Global variables defined

- aio_name
- ngx_aio_module
- ngx_aio_module_ctx
- ngx_os_aio

Functions defined

- ngx_aio_add_event
- ngx_aio_del_connection
- ngx_aio_del_event
- ngx_aio_done
- ngx_aio_init
- ngx_aio_process_events

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  extern ngx_event_module_t  ngx_kqueue_module_ctx;
14
15
16  static ngx_int_t ngx_aio_init(ngx_cycle_t *cycle, ngx_msec_t timer);
17  static void ngx_aio_done(ngx_cycle_t *cycle);
18  static ngx_int_t ngx_aio_add_event(ngx_event_t *ev, ngx_int_t event,
19      ngx_uint_t flags);
20  static ngx_int_t ngx_aio_del_event(ngx_event_t *ev, ngx_int_t event,
21      ngx_uint_t flags);
22  static ngx_int_t ngx_aio_del_connection(ngx_connection_t *c, ngx_uint_t flags);
23  static ngx_int_t ngx_aio_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
24      ngx_uint_t flags);
25
26
27  ngx_os_io_t ngx_os_aio = {
28      ngx_aio_read,
29      ngx_aio_read_chain,
30      NULL,
31      ngx_aio_write,
32      ngx_aio_write_chain,
33      0
34  };
35
36
37  static ngx_str_t      aio_name = ngx_string("aio");
38
```

```
ngx_event_module_t  ngx_aio_module_ctx = {
    &aio_name,
    NULL,                                  /* create configuration */
    NULL,                                  /* init configuration */

    {
        ngx_aio_add_event,                 /* add an event */
        ngx_aio_del_event,                 /* delete an event */
        NULL,                              /* enable an event */
        NULL,                              /* disable an event */
        NULL,                              /* add an connection */
        ngx_aio_del_connection,            /* delete an connection */
        NULL,                              /* process the changes */
        ngx_aio_process_events,            /* process the events */
        ngx_aio_init,                      /* init the events */
        ngx_aio_done                       /* done the events */
    }

};

ngx_module_t  ngx_aio_module = {
    NGX_MODULE_V1,
    &ngx_aio_module_ctx,                   /* module context */
    NULL,                                  /* module directives */
    NGX_EVENT_MODULE,                      /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


#if (NGX_HAVE_KQUEUE)

static ngx_int_t
ngx_aio_init(ngx_cycle_t *cycle, ngx_msec_t timer)
{
    if (ngx_kqueue_module_ctx.actions.init(cycle, timer) == NGX_ERROR) {
        return NGX_ERROR;
    }

    ngx_io = ngx_os_aio;

    ngx_event_flags = NGX_USE_AIO_EVENT;
    ngx_event_actions = ngx_aio_module_ctx.actions;


    return NGX_OK;
}


static void
ngx_aio_done(ngx_cycle_t *cycle)
{
    ngx_kqueue_module_ctx.actions.done(cycle);
}


 /* the event adding and deleting are needed for the listening sockets */

 static ngx_int_t
 ngx_aio_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
 {
     return ngx_kqueue_module_ctx.actions.add(ev, event, flags);
 }


 static ngx_int_t
 ngx_aio_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
 {
     return ngx_kqueue_module_ctx.actions.del(ev, event, flags);
 }
```

```
115
116
117 static ngx_int_t
118 ngx_aio_del_connection(ngx_connection_t *c, ngx_uint_t flags)
119 {
120     int  rc;
121
122     if (c->read->active == 0 && c->write->active == 0) {
123         return NGX_OK;
124     }
125
126     if (flags & NGX_CLOSE_EVENT) {
127         return NGX_OK;
128     }
129
130     rc = aio_cancel(c->fd, NULL);
131
132     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, 0, "aio_cancel: %d", rc);
133
134     if (rc == AIO_CANCELED) {
135         c->read->active = 0;
136         c->write->active = 0;
137         return NGX_OK;
138     }
139
140     if (rc == AIO_ALLDONE) {
141         c->read->active = 0;
142         c->write->active = 0;
143         ngx_log_error(NGX_LOG_ALERT, c->log, 0,
144                       "aio_cancel() returned AIO_ALLDONE");
145         return NGX_OK;
146     }
147
148     if (rc == -1) {
149         ngx_log_error(NGX_LOG_ALERT, c->log, ngx_errno,
150                       "aio_cancel() failed");
151         return NGX_ERROR;
152     }
153
154     if (rc == AIO_NOTCANCELED) {
155         ngx_log_error(NGX_LOG_ALERT, c->log, 0,
156                       "aio_cancel() returned AIO_NOTCANCELED");
157
158         return NGX_ERROR;
159     }
160
161     return NGX_OK;
162 }
163
164
165 static ngx_int_t
166 ngx_aio_process_events(ngx_cycle_t *cycle, ngx_msec_t timer, ngx_uint_t flags)
167 {
168     return ngx_kqueue_module_ctx.actions.process_events(cycle, timer, flags);
169 }
170
171 #endif /* NGX_HAVE_KQUEUE */
```

One Level Up Top Level

src/event/modules/ngx_win32_select_module.c - nginx-1.7.10

Global variables defined

- event_index
- master_read_fd_set
- master_write_fd_set
- max_read
- max_write
- nevents
- ngx_select_module
- ngx_select_module_ctx
- select_name
- work_read_fd_set
- work_write_fd_set

Functions defined

- ngx_select_add_event
- ngx_select_del_event
- ngx_select_done
- ngx_select_init
- ngx_select_init_conf
- ngx_select_process_events
- ngx_select_repair_fd_sets

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  static ngx_int_t ngx_select_init(ngx_cycle_t *cycle, ngx_msec_t timer);
14  static void ngx_select_done(ngx_cycle_t *cycle);
15  static ngx_int_t ngx_select_add_event(ngx_event_t *ev, ngx_int_t event,
16      ngx_uint_t flags);
17  static ngx_int_t ngx_select_del_event(ngx_event_t *ev, ngx_int_t event,
18      ngx_uint_t flags);
19  static ngx_int_t ngx_select_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
20      ngx_uint_t flags);
```

```c
static void ngx_select_repair_fd_sets(ngx_cycle_t *cycle);
static char *ngx_select_init_conf(ngx_cycle_t *cycle, void *conf);


static fd_set         master_read_fd_set;
static fd_set         master_write_fd_set;
static fd_set         work_read_fd_set;
static fd_set         work_write_fd_set;

static ngx_uint_t     max_read;
static ngx_uint_t     max_write;
static ngx_uint_t     nevents;

static ngx_event_t  **event_index;


static ngx_str_t    select_name = ngx_string("select");

ngx_event_module_t  ngx_select_module_ctx = {
    &select_name,
    NULL,                                  /* create configuration */
    ngx_select_init_conf,                  /* init configuration */

    {
        ngx_select_add_event,              /* add an event */
        ngx_select_del_event,              /* delete an event */
        ngx_select_add_event,              /* enable an event */
        ngx_select_del_event,              /* disable an event */
        NULL,                              /* add an connection */
        NULL,                              /* delete an connection */
        NULL,                              /* process the changes */
        ngx_select_process_events,         /* process the events */
        ngx_select_init,                   /* init the events */
        ngx_select_done                    /* done the events */
    }

};

ngx_module_t  ngx_select_module = {
    NGX_MODULE_V1,
    &ngx_select_module_ctx,                /* module context */
    NULL,                                  /* module directives */
    NGX_EVENT_MODULE,                      /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_select_init(ngx_cycle_t *cycle, ngx_msec_t timer)
{
    ngx_event_t  **index;

    if (event_index == NULL) {
        FD_ZERO(&master_read_fd_set);
        FD_ZERO(&master_write_fd_set);
        nevents = 0;
    }

    if (ngx_process >= NGX_PROCESS_WORKER
        || cycle->old_cycle == NULL
        || cycle->old_cycle->connection_n < cycle->connection_n)
    {
        index = ngx_alloc(sizeof(ngx_event_t *) * 2 * cycle->connection_n,
                          cycle->log);
        if (index == NULL) {
            return NGX_ERROR;
        }

        if (event_index) {
```

```
            ngx_memcpy(index, event_index, sizeof(ngx_event_t *) * nevents);
            ngx_free(event_index);
        }

        event_index = index;
    }

    ngx_io = ngx_os_io;

    ngx_event_actions = ngx_select_module_ctx.actions;

    ngx_event_flags = NGX_USE_LEVEL_EVENT;

    max_read = 0;
    max_write = 0;

    return NGX_OK;
}


static void
ngx_select_done(ngx_cycle_t *cycle)
{
    ngx_free(event_index);

    event_index = NULL;
}


static ngx_int_t
ngx_select_add_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_connection_t  *c;

    c = ev->data;

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "select add event fd:%d ev:%i", c->fd, event);

    if (ev->index != NGX_INVALID_INDEX) {
        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "select event fd:%d ev:%i is already set", c->fd, event);
        return NGX_OK;
    }

    if ((event == NGX_READ_EVENT && ev->write)
        || (event == NGX_WRITE_EVENT && !ev->write))
    {
        ngx_log_error(NGX_LOG_ALERT, ev->log, 0,
                      "invalid select %s event fd:%d ev:%i",
                      ev->write ? "write" : "read", c->fd, event);
        return NGX_ERROR;
    }

    if ((event == NGX_READ_EVENT && max_read >= FD_SETSIZE)
        || (event == NGX_WRITE_EVENT && max_write >= FD_SETSIZE))
    {
        ngx_log_error(NGX_LOG_ERR, ev->log, 0,
                      "maximum number of descriptors "
                      "supported by select() is %d", FD_SETSIZE);
        return NGX_ERROR;
    }

    if (event == NGX_READ_EVENT) {
        FD_SET(c->fd, &master_read_fd_set);
        max_read++;

    } else if (event == NGX_WRITE_EVENT) {
        FD_SET(c->fd, &master_write_fd_set);
        max_write++;
    }

    ev->active = 1;

    event_index[nevents] = ev;
    ev->index = nevents;
```

```
    nevents++;

    return NGX_OK;
}


static ngx_int_t
ngx_select_del_event(ngx_event_t *ev, ngx_int_t event, ngx_uint_t flags)
{
    ngx_event_t       *e;
    ngx_connection_t  *c;

    c = ev->data;

    ev->active = 0;

    if (ev->index == NGX_INVALID_INDEX) {
        return NGX_OK;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_EVENT, ev->log, 0,
                   "select del event fd:%d ev:%i", c->fd, event);

    if (event == NGX_READ_EVENT) {
        FD_CLR(c->fd, &master_read_fd_set);
        max_read--;

    } else if (event == NGX_WRITE_EVENT) {
        FD_CLR(c->fd, &master_write_fd_set);
        max_write--;
    }

    if (ev->index < --nevents) {
        e = event_index[nevents];
        event_index[ev->index] = e;
        e->index = ev->index;
    }

    ev->index = NGX_INVALID_INDEX;

    return NGX_OK;
}


static ngx_int_t
ngx_select_process_events(ngx_cycle_t *cycle, ngx_msec_t timer,
    ngx_uint_t flags)
{
    int                ready, nready;
    ngx_err_t          err;
    ngx_uint_t         i, found;
    ngx_event_t       *ev;
    ngx_queue_t       *queue;
    struct timeval     tv, *tp;
    ngx_connection_t  *c;

#if (NGX_DEBUG)
    if (cycle->log->log_level & NGX_LOG_DEBUG_ALL) {
        for (i = 0; i < nevents; i++) {
            ev = event_index[i];
            c = ev->data;
            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                           "select event: fd:%d wr:%d", c->fd, ev->write);
        }
    }
#endif

    if (timer == NGX_TIMER_INFINITE) {
        tp = NULL;

    } else {
        tv.tv_sec = (long) (timer / 1000);
        tv.tv_usec = (long) ((timer % 1000) * 1000);
        tp = &tv;
    }

```

```
    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "select timer: %M", timer);

    work_read_fd_set = master_read_fd_set;
    work_write_fd_set = master_write_fd_set;

    if (max_read || max_write) {
        ready = select(0, &work_read_fd_set, &work_write_fd_set, NULL, tp);

    } else {

        /*
         * Winsock select() requires that at least one descriptor set must be
         * be non-null, and any non-null descriptor set must contain at least
         * one handle to a socket.  Otherwise select() returns WSAEINVAL.
         */

        ngx_msleep(timer);

        ready = 0;
    }

    err = (ready == -1) ? ngx_socket_errno : 0;

    if (flags & NGX_UPDATE_TIME) {
        ngx_time_update();
    }

    ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                   "select ready %d", ready);

    if (err) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, err, "select() failed");

        if (err == WSAENOTSOCK) {
            ngx_select_repair_fd_sets(cycle);
        }

        return NGX_ERROR;
    }

    if (ready == 0) {
        if (timer != NGX_TIMER_INFINITE) {
            return NGX_OK;
        }

        ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                      "select() returned no events without timeout");
        return NGX_ERROR;
    }

    nready = 0;

    for (i = 0; i < nevents; i++) {
        ev = event_index[i];
        c = ev->data;
        found = 0;

        if (ev->write) {
            if (FD_ISSET(c->fd, &work_write_fd_set)) {
                found = 1;
                ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                               "select write %d", c->fd);
            }

        } else {
            if (FD_ISSET(c->fd, &work_read_fd_set)) {
                found = 1;
                ngx_log_debug1(NGX_LOG_DEBUG_EVENT, cycle->log, 0,
                               "select read %d", c->fd);
            }
        }

        if (found) {
            ev->ready = 1;

```

```
            queue = ev->accept ? &ngx_posted_accept_events
                               : &ngx_posted_events;

            ngx_post_event(ev, queue);

            nready++;
        }
    }

    if (ready != nready) {
        ngx_log_error(NGX_LOG_ALERT, cycle->log, 0,
                      "select ready != events: %d:%d", ready, nready);

        ngx_select_repair_fd_sets(cycle);
    }

    return NGX_OK;
}


static void
ngx_select_repair_fd_sets(ngx_cycle_t *cycle)
{
    int           n;
    u_int         i;
    socklen_t     len;
    ngx_err_t     err;
    ngx_socket_t  s;

    for (i = 0; i < master_read_fd_set.fd_count; i++) {

        s = master_read_fd_set.fd_array[i];
        len = sizeof(int);

        if (getsockopt(s, SOL_SOCKET, SO_TYPE, (char *) &n, &len) == -1) {
            err = ngx_socket_errno;

            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          "invalid descriptor #%d in read fd_set", s);

            FD_CLR(s, &master_read_fd_set);
        }
    }

    for (i = 0; i < master_write_fd_set.fd_count; i++) {

        s = master_write_fd_set.fd_array[i];
        len = sizeof(int);

        if (getsockopt(s, SOL_SOCKET, SO_TYPE, (char *) &n, &len) == -1) {
            err = ngx_socket_errno;

            ngx_log_error(NGX_LOG_ALERT, cycle->log, err,
                          "invalid descriptor #%d in write fd_set", s);

            FD_CLR(s, &master_write_fd_set);
        }
    }
}


static char *
ngx_select_init_conf(ngx_cycle_t *cycle, void *conf)
{
    ngx_event_conf_t  *ecf;

    ecf = ngx_event_get_conf(cycle->conf_ctx, ngx_event_core_module);

    if (ecf->use != ngx_select_module.ctx_index) {
        return NGX_CONF_OK;
    }

    return NGX_CONF_OK;
}
```

One Level Up Top Level

src/os/unix/ngx_darwin.h - nginx-1.7.10

Macros defined

- _NGX_DARWIN_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_DARWIN_H_INCLUDED_
9  #define _NGX_DARWIN_H_INCLUDED_
10
11
12  void ngx_debug_init(void);
13  ngx_chain_t *ngx_darwin_sendfile_chain(ngx_connection_t *c, ngx_chain_t *in,
14      off_t limit);
15
16  extern int       ngx_darwin_kern_osreldate;
17  extern int       ngx_darwin_hw_ncpu;
18  extern u_long    ngx_darwin_net_inet_tcp_sendspace;
19
20  extern ngx_uint_t  ngx_debug_malloc;
21
22
23  #endif /* _NGX_DARWIN_H_INCLUDED_ */
```

src/os/unix/ngx_darwin_sendfile_chain.c - nginx-1.7.10

Functions defined

- ngx_darwin_sendfile_chain

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_event.h>
11
12
13  /*
14  * It seems that Darwin 9.4 (Mac OS X 1.5) sendfile() has the same
15  * old bug as early FreeBSD sendfile() syscall:
16  * http://bugs.freebsd.org/33771
17  *
18  * Besides sendfile() has another bug: if one calls sendfile()
19  * with both a header and a trailer, then sendfile() ignores a file part
20  * at all and sends only the header and the trailer together.
21  * For this reason we send a trailer only if there is no a header.
22  *
23  * Although sendfile() allows to pass a header or a trailer,
24  * it may send the header or the trailer and a part of the file
25  * in different packets.  And FreeBSD workaround (TCP_NOPUSH option)
26  * does not help.
27  */
28
29
30  ngx_chain_t *
31  ngx_darwin_sendfile_chain(ngx_connection_t *c, ngx_chain_t *in, off_t limit)
32  {
33      int              rc;
34      off_t            send, prev_send, sent;
35      off_t            file_size;
36      ssize_t          n;
37      ngx_uint_t       eintr;
38      ngx_err_t        err;
39      ngx_buf_t       *file;
40      ngx_event_t     *wev;
41      ngx_chain_t     *cl;
42      ngx_iovec_t      header, trailer;
43      struct sf_hdtr   hdtr;
44      struct iovec     headers[NGX_IOVS_PREALLOCATE];
45      struct iovec     trailers[NGX_IOVS_PREALLOCATE];
46
47      wev = c->write;
48
49      if (!wev->ready) {
50          return in;
51      }
52
53  #if (NGX_HAVE_KQUEUE)
54
55      if ((ngx_event_flags & NGX_USE_KQUEUE_EVENT) && wev->pending_eof) {
56          (void) ngx_connection_error(c, wev->kq_errno,
57                                      "kevent() reported about an closed connection");
58          wev->error = 1;
59          return NGX_CHAIN_ERROR;
60      }
61
62  #endif
```

```
63
64      /* the maximum limit size is the maximum size_t value - the page size */
65
66      if (limit == 0 || limit > (off_t) (NGX_MAX_SIZE_T_VALUE - ngx_pagesize)) {
67          limit = NGX_MAX_SIZE_T_VALUE - ngx_pagesize;
68      }
69
70      send = 0;
71
72      header.iovs = headers;
73      header.nalloc = NGX_IOVS_PREALLOCATE;
74
75      trailer.iovs = trailers;
76      trailer.nalloc = NGX_IOVS_PREALLOCATE;
77
78      for ( ;; ) {
79          eintr = 0;
80          prev_send = send;
81
82          /* create the header iovec and coalesce the neighbouring bufs */
83
84          cl = ngx_output_chain_to_iovec(&header, in, limit - send, c->log);
85
86          if (cl == NGX_CHAIN_ERROR) {
87              return NGX_CHAIN_ERROR;
88          }
89
90          send += header.size;
91
92          if (cl && cl->buf->in_file && send < limit) {
93              file = cl->buf;
94
95              /* coalesce the neighbouring file bufs */
96
97              file_size = ngx_chain_coalesce_file(&cl, limit - send);
98
99              send += file_size;
100
101             if (header.count == 0) {
102
103                 /*
104                  * create the trailer iovec and coalesce the neighbouring bufs
105                  */
106
107                 cl = ngx_output_chain_to_iovec(&trailer, cl, limit - send,
108                                                c->log);
109                 if (cl == NGX_CHAIN_ERROR) {
110                     return NGX_CHAIN_ERROR;
111                 }
112
113                 send += trailer.size;
114
115             } else {
116                 trailer.count = 0;
117             }
118
119             /*
120              * sendfile() returns EINVAL if sf_hdtr's count is 0,
121              * but corresponding pointer is not NULL
122              */
123
124             hdtr.headers = header.count ? header.iovs : NULL;
125             hdtr.hdr_cnt = header.count;
126             hdtr.trailers = trailer.count ? trailer.iovs : NULL;
127             hdtr.trl_cnt = trailer.count;
128
129             sent = header.size + file_size;
130
131             ngx_log_debug3(NGX_LOG_DEBUG_EVENT, c->log, 0,
132                            "sendfile: @%O %O h:%uz",
133                            file->file_pos, sent, header.size);
134
135             rc = sendfile(file->file->fd, c->fd, file->file_pos,
136                           &sent, &hdtr, 0);
137
138             if (rc == -1) {
```

```
                err = ngx_errno;

                switch (err) {
                case NGX_EAGAIN:
                    break;

                case NGX_EINTR:
                    eintr = 1;
                    break;

                default:
                    wev->error = 1;
                    (void) ngx_connection_error(c, err, "sendfile() failed");
                    return NGX_CHAIN_ERROR;
                }

                ngx_log_debug1(NGX_LOG_DEBUG_EVENT, c->log, err,
                               "sendfile() sent only %O bytes", sent);
            }

            if (rc == 0 && sent == 0) {

                /*
                 * if rc and sent equal to zero, then someone
                 * has truncated the file, so the offset became beyond
                 * the end of the file
                 */

                ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                              "sendfile() reported that \"%s\" was truncated",
                              file->file->name.data);

                return NGX_CHAIN_ERROR;
            }

            ngx_log_debug4(NGX_LOG_DEBUG_EVENT, c->log, 0,
                           "sendfile: %d, @%O %O:%O",
                           rc, file->file_pos, sent, file_size + header.size);

        } else {
            n = ngx_writev(c, &header);

            if (n == NGX_ERROR) {
                return NGX_CHAIN_ERROR;
            }

            sent = (n == NGX_AGAIN) ? 0 : n;
        }

        c->sent += sent;

        in = ngx_chain_update_sent(in, sent);

        if (eintr) {
            send = prev_send + sent;
            continue;
        }

        if (send - prev_send != sent) {
            wev->ready = 0;
            return in;
        }

        if (send >= limit || in == NULL) {
            return in;
        }
    }
}
```

One Level Up Top Level

src/os/unix/ngx_darwin_init.c - nginx-1.7.10

Global variables defined

- ngx_darwin_hw_ncpu
- ngx_darwin_io
- ngx_darwin_kern_ipc_somaxconn
- ngx_darwin_kern_osrelease
- ngx_darwin_kern_ostype
- ngx_darwin_net_inet_tcp_sendspace
- ngx_debug_malloc
- sysctls

Data types defined

- sysctl_t

Functions defined

- ngx_debug_init
- ngx_os_specific_init
- ngx_os_specific_status

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  char     ngx_darwin_kern_ostype[16];
13  char     ngx_darwin_kern_osrelease[128];
14  int      ngx_darwin_hw_ncpu;
15  int      ngx_darwin_kern_ipc_somaxconn;
16  u_long   ngx_darwin_net_inet_tcp_sendspace;
17
18  ngx_uint_t  ngx_debug_malloc;
19
20
21  static ngx_os_io_t ngx_darwin_io = {
22      ngx_unix_recv,
23      ngx_readv_chain,
24      ngx_udp_unix_recv,
25      ngx_unix_send,
26  #if (NGX_HAVE_SENDFILE)
27      ngx_darwin_sendfile_chain,
28      NGX_IO_SENDFILE
29  #else
30      ngx_writev_chain,
```

```
    0
#endif
};


typedef struct {
    char        *name;
    void        *value;
    size_t       size;
    ngx_uint_t   exists;
} sysctl_t;


sysctl_t sysctls[] = {
    { "hw.ncpu",
      &ngx_darwin_hw_ncpu,
      sizeof(ngx_darwin_hw_ncpu), 0 },

    { "net.inet.tcp.sendspace",
      &ngx_darwin_net_inet_tcp_sendspace,
      sizeof(ngx_darwin_net_inet_tcp_sendspace), 0 },

    { "kern.ipc.somaxconn",
      &ngx_darwin_kern_ipc_somaxconn,
      sizeof(ngx_darwin_kern_ipc_somaxconn), 0 },

    { NULL, NULL, 0, 0 }
};


void
ngx_debug_init(void)
{
#if (NGX_DEBUG_MALLOC)

    /*
     * MacOSX 10.6, 10.7:  MallocScribble fills freed memory with 0x55
     *                     and fills allocated memory with 0xAA.
     * MacOSX 10.4, 10.5:  MallocScribble fills freed memory with 0x55,
     *                     MallocPreScribble fills allocated memory with 0xAA.
     * MacOSX 10.3:        MallocScribble fills freed memory with 0x55,
     *                     and no way to fill allocated memory.
     */

    setenv("MallocScribble", "1", 0);

    ngx_debug_malloc = 1;

#else

    if (getenv("MallocScribble")) {
        ngx_debug_malloc = 1;
    }

#endif
}


ngx_int_t
ngx_os_specific_init(ngx_log_t *log)
{
    size_t      size;
    ngx_err_t   err;
    ngx_uint_t  i;

    size = sizeof(ngx_darwin_kern_ostype);
    if (sysctlbyname("kern.ostype", ngx_darwin_kern_ostype, &size, NULL, 0)
        == -1)
    {
         err = ngx_errno;

         if (err != NGX_ENOENT) {

             ngx_log_error(NGX_LOG_ALERT, log, err,
                           "sysctlbyname(kern.ostype) failed");

```

```
            if (err != NGX_ENOMEM) {
                return NGX_ERROR;
            }

            ngx_darwin_kern_ostype[size - 1] = '\0';
        }
    }

    size = sizeof(ngx_darwin_kern_osrelease);
    if (sysctlbyname("kern.osrelease", ngx_darwin_kern_osrelease, &size,
                     NULL, 0)
        == -1)
    {
        err = ngx_errno;

        if (err != NGX_ENOENT) {

            ngx_log_error(NGX_LOG_ALERT, log, err,
                          "sysctlbyname(kern.osrelease) failed");

            if (err != NGX_ENOMEM) {
                return NGX_ERROR;
            }

            ngx_darwin_kern_osrelease[size - 1] = '\0';
        }
    }

    for (i = 0; sysctls[i].name; i++) {
        size = sysctls[i].size;

        if (sysctlbyname(sysctls[i].name, sysctls[i].value, &size, NULL, 0)
            == 0)
        {
            sysctls[i].exists = 1;
            continue;
        }

        err = ngx_errno;

        if (err == NGX_ENOENT) {
            continue;
        }

        ngx_log_error(NGX_LOG_ALERT, log, err,
                      "sysctlbyname(%s) failed", sysctls[i].name);
        return NGX_ERROR;
    }

    ngx_ncpu = ngx_darwin_hw_ncpu;

    if (ngx_darwin_kern_ipc_somaxconn > 32767) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "sysctl kern.ipc.somaxconn must be less than 32768");
        return NGX_ERROR;
    }

    ngx_tcp_nodelay_and_tcp_nopush = 1;

    ngx_os_io = ngx_darwin_io;

    return NGX_OK;
}


void
ngx_os_specific_status(ngx_log_t *log)
{
    u_long      value;
    ngx_uint_t  i;

    if (ngx_darwin_kern_ostype[0]) {
        ngx_log_error(NGX_LOG_NOTICE, log, 0, "OS: %s %s",
                      ngx_darwin_kern_ostype, ngx_darwin_kern_osrelease);
    }

```

```
    for (i = 0; sysctls[i].name; i++) {
        if (sysctls[i].exists) {
            if (sysctls[i].size == sizeof(long)) {
                value = *(long *) sysctls[i].value;

            } else {
                value = *(int *) sysctls[i].value;
            }

            ngx_log_error(NGX_LOG_NOTICE, log, 0, "%s: %l",
                          sysctls[i].name, value);
        }
    }
}
```

One Level Up Top Level

src/os/unix/ngx_freebsd.h - nginx-1.7.10

Macros defined

- _NGX_FREEBSD_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_FREEBSD_H_INCLUDED_
9  #define _NGX_FREEBSD_H_INCLUDED_
10
11
12  void ngx_debug_init(void);
13  ngx_chain_t *ngx_freebsd_sendfile_chain(ngx_connection_t *c, ngx_chain_t *in,
14      off_t limit);
15
16  extern int          ngx_freebsd_kern_osreldate;
17  extern int          ngx_freebsd_hw_ncpu;
18  extern u_long       ngx_freebsd_net_inet_tcp_sendspace;
19
20  extern ngx_uint_t   ngx_freebsd_sendfile_nbytes_bug;
21  extern ngx_uint_t   ngx_freebsd_use_tcp_nopush;
22  extern ngx_uint_t   ngx_debug_malloc;
23
24
25  #endif /* _NGX_FREEBSD_H_INCLUDED_ */
```

src/os/unix/ngx_gcc_atomic_amd64.h - nginx-1.7.10

Functions defined

- ngx_atomic_cmp_set
- ngx_atomic_fetch_add

Macros defined

- NGX_SMP_LOCK
- NGX_SMP_LOCK
- ngx_cpu_pause
- ngx_memory_barrier

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #if (NGX_SMP)
9  #define NGX_SMP_LOCK  "lock;"
10  #else
11  #define NGX_SMP_LOCK
12  #endif
13
14
15  /*
16  * "cmpxchgq  r, [m]":
17  *
18  *     if (rax == [m]) {
19  *         zf = 1;
20  *         [m] = r;
21  *     } else {
22  *         zf = 0;
23  *         rax = [m];
24  *     }
25  *
26  *
27  * The "r" is any register, %rax (%r0) - %r16.
28  * The "=a" and "a" are the %rax register.
29  * Although we can return result in any register, we use "a" because it is
30  * used in cmpxchgq anyway.  The result is actually in %al but not in $rax,
31  * however as the code is inlined gcc can test %al as well as %rax.
32  *
33  * The "cc" means that flags were changed.
34  */
35
36  static ngx_inline ngx_atomic_uint_t
37  ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
38      ngx_atomic_uint_t set)
39  {
40      u_char  res;
41
42      __asm__ volatile (
43
44           NGX_SMP_LOCK
45      "    cmpxchgq  %3, %1;   "
46      "    sete      %0;       "
47
48      : "=a" (res) : "m" (*lock), "a" (old), "r" (set) : "cc", "memory");
```

```
49
50      return res;
51  }
52
53
54  /*
55  * "xaddq  r, [m]":
56  *
57  *     temp = [m];
58  *     [m] += r;
59  *     r = temp;
60  *
61  *
62  * The "+r" is any register, %rax (%r0) - %r16.
63  * The "cc" means that flags were changed.
64  */
65
66  static ngx_inline ngx_atomic_int_t
67  ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
68  {
69      __asm__ volatile (
70
71           NGX_SMP_LOCK
72      "    xaddq  %0, %1;   "
73
74      : "+r" (add) : "m" (*value) : "cc", "memory");
75
76      return add;
77  }
78
79
80  #define ngx_memory_barrier()    __asm__ volatile ("" ::: "memory")
81
82  #define ngx_cpu_pause()         __asm__ ("pause")
```

One Level Up Top Level

src/os/unix/ngx_gcc_atomic_x86.h - nginx-1.7.10

Functions defined

- ngx_atomic_cmp_set
- ngx_atomic_fetch_add
- ngx_atomic_fetch_add

Macros defined

- NGX_SMP_LOCK
- NGX_SMP_LOCK
- ngx_cpu_pause
- ngx_memory_barrier

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #if (NGX_SMP)
9  #define NGX_SMP_LOCK  "lock;"
10  #else
11  #define NGX_SMP_LOCK
12  #endif
13
14
15  /*
16  * "cmpxchgl  r, [m]":
17  *
18  *     if (eax == [m]) {
19  *         zf = 1;
20  *         [m] = r;
21  *     } else {
22  *         zf = 0;
23  *         eax = [m];
24  *     }
25  *
26  *
27  * The "r" means the general register.
28  * The "=a" and "a" are the %eax register.
29  * Although we can return result in any register, we use "a" because it is
30  * used in cmpxchgl anyway.  The result is actually in %al but not in %eax,
31  * however, as the code is inlined gcc can test %al as well as %eax,
32  * and icc adds "movzbl %al, %eax" by itself.
33  *
34  * The "cc" means that flags were changed.
35  */
36
37  static ngx_inline ngx_atomic_uint_t
38  ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
39      ngx_atomic_uint_t set)
40  {
41      u_char  res;
42
43      __asm__ volatile (
44
45           NGX_SMP_LOCK
```

```
    "    cmpxchgl  %3, %1;   "
    "    sete      %0;       "

    : "=a" (res) : "m" (*lock), "a" (old), "r" (set) : "cc", "memory");

    return res;
}


/*
* "xaddl  r, [m]":
*
*     temp = [m];
*     [m] += r;
*     r = temp;
*
*
* The "+r" means the general register.
* The "cc" means that flags were changed.
*/


#if !(( __GNUC__ == 2 && __GNUC_MINOR__ <= 7 ) || ( __INTEL_COMPILER >= 800 ))

/*
* icc 8.1 and 9.0 compile broken code with -march=pentium4 option:
* ngx_atomic_fetch_add() always return the input "add" value,
* so we use the gcc 2.7 version.
*
* icc 8.1 and 9.0 with -march=pentiumpro option or icc 7.1 compile
* correct code.
*/

static ngx_inline ngx_atomic_int_t
ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
{
    __asm__ volatile (

         NGX_SMP_LOCK
    "    xaddl  %0, %1;   "

    : "+r" (add) : "m" (*value) : "cc", "memory");

    return add;
}


#else

/*
* gcc 2.7 does not support "+r", so we have to use the fixed
* %eax ("=a" and "a") and this adds two superfluous instructions in the end
* of code, something like this: "mov %eax, %edx / mov %edx, %eax".
*/

 static ngx_inline ngx_atomic_int_t
 ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
 {
     ngx_atomic_uint_t  old;

     __asm__ volatile (

          NGX_SMP_LOCK
     "    xaddl  %2, %1;   "

     : "=a" (old) : "m" (*value), "a" (add) : "cc", "memory");

     return old;
 }

 #endif


 /*
 * on x86 the write operations go in a program order, so we need only
 * to disable the gcc reorder optimizations
```

```
122 */
123
124 #define ngx_memory_barrier()    __asm__ volatile ("" ::: "memory")
125
126 /* old "as" does not support "pause" opcode */
127 #define ngx_cpu_pause()         __asm__ (".byte 0xf3, 0x90")
```

One Level Up Top Level

src/os/unix/ngx_gcc_atomic_ppc.h - nginx-1.7.10

Functions defined

- ngx_atomic_cmp_set
- ngx_atomic_cmp_set
- ngx_atomic_fetch_add
- ngx_atomic_fetch_add

Macros defined

- ngx_cpu_pause
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_memory_barrier
- ngx_memory_barrier

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  /*
9  * The ppc assembler treats ";" as comment, so we have to use "\n".
10 * The minus in "bne-" is a hint for the branch prediction unit that
11 * this branch is unlikely to be taken.
12 * The "1b" means the nearest backward label "1" and the "1f" means
13 * the nearest forward label "1".
14 *
15 * The "b" means that the base registers can be used only, i.e.
16 * any register except r0.  The r0 register always has a zero value and
17 * could not be used in "addi  r0, r0, 1".
18 * The "=&b" means that no input registers can be used.
19 *
20 * "sync"    read and write barriers
21 * "isync"   read barrier, is faster than "sync"
22 * "eieio"   write barrier, is faster than "sync"
23 * "lwsync"  write barrier, is faster than "eieio" on ppc64
24 */
25
26 #if (NGX_PTR_SIZE == 8)
27
28 static ngx_inline ngx_atomic_uint_t
29 ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
30     ngx_atomic_uint_t set)
31 {
32     ngx_atomic_uint_t  res, temp;
33
34     __asm__ volatile (
35
36     "    li      %0, 0       \n" /* preset "0" to "res"                    */
37     "    lwsync              \n" /* write barrier                          */
38     "1:                      \n"
39     "    ldarx   %1, 0, %2   \n" /* load from [lock] into "temp"           */
40                                  /*   and store reservation                */
41     "    cmpd    %1, %3      \n" /* compare "temp" and "old"               */
```

```
    "    bne-    2f          \n" /* not equal                                */
    "    stdcx.  %4, 0, %2   \n" /* store "set" into [lock] if reservation   */
                                 /*   is not cleared                         */
    "    bne-    1b          \n" /* the reservation was cleared              */
    "    isync               \n" /* read barrier                             */
    "    li      %0, 1       \n" /* set "1" to "res"                         */
    "2:                      \n"

    : "=&b" (res), "=&b" (temp)
    : "b" (lock), "b" (old), "b" (set)
    : "cc", "memory");

    return res;
}


static ngx_inline ngx_atomic_int_t
ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
{
    ngx_atomic_uint_t  res, temp;

    __asm__ volatile (

    "    lwsync              \n" /* write barrier                            */
    "1:  ldarx   %0, 0, %2   \n" /* load from [value] into "res"             */
                                 /*   and store reservation                  */
    "    add     %1, %0, %3  \n" /* "res" + "add" store in "temp"            */
    "    stdcx.  %1, 0, %2   \n" /* store "temp" into [value] if reservation */
                                 /*   is not cleared                         */
    "    bne-    1b          \n" /* try again if reservation was cleared     */
    "    isync               \n" /* read barrier                             */

    : "=&b" (res), "=&b" (temp)
    : "b" (value), "b" (add)
    : "cc", "memory");

    return res;
}


#if (NGX_SMP)
#define ngx_memory_barrier()                                                 \
    __asm__ volatile ("isync  \n  lwsync  \n" ::: "memory")
#else
#define ngx_memory_barrier()   __asm__ volatile ("" ::: "memory")
#endif

#else

static ngx_inline ngx_atomic_uint_t
ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
    ngx_atomic_uint_t set)
{
    ngx_atomic_uint_t  res, temp;

    __asm__ volatile (

    "    li      %0, 0       \n" /* preset "0" to "res"                      */
     "    eieio               \n" /* write barrier                            */
     "1:                      \n"
     "    lwarx   %1, 0, %2   \n" /* load from [lock] into "temp"             */
                                  /*   and store reservation                  */
     "    cmpw    %1, %3      \n" /* compare "temp" and "old"                 */
     "    bne-    2f          \n" /* not equal                                */
     "    stwcx.  %4, 0, %2   \n" /* store "set" into [lock] if reservation   */
                                  /*   is not cleared                         */
     "    bne-    1b          \n" /* the reservation was cleared              */
     "    isync               \n" /* read barrier                             */
     "    li      %0, 1       \n" /* set "1" to "res"                         */
     "2:                      \n"

     : "=&b" (res), "=&b" (temp)
     : "b" (lock), "b" (old), "b" (set)
     : "cc", "memory");

     return res;
```

```
}


static ngx_inline ngx_atomic_int_t
ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
{
    ngx_atomic_uint_t  res, temp;

    __asm__ volatile (

    "    eieio                \n" /* write barrier                              */
    "1:  lwarx   %0, 0, %2    \n" /* load from [value] into "res"               */
                                  /*   and store reservation                    */
    "    add     %1, %0, %3   \n" /* "res" + "add" store in "temp"              */
    "    stwcx.  %1, 0, %2    \n" /* store "temp" into [value] if reservation */
                                  /*   is not cleared                           */
    "    bne-    1b           \n" /* try again if reservation was cleared      */
    "    isync                \n" /* read barrier                               */

    : "=&b" (res), "=&b" (temp)
    : "b" (value), "b" (add)
    : "cc", "memory");

    return res;
}


#if (NGX_SMP)
#define ngx_memory_barrier()                                                 \
    __asm__ volatile ("isync  \n  eieio  \n" ::: "memory")
#else
#define ngx_memory_barrier()   __asm__ volatile ("" ::: "memory")
#endif

#endif


#define ngx_cpu_pause()
```

One Level Up Top Level

src/os/unix/ngx_linux.h - nginx-1.7.10

Macros defined

- _NGX_LINUX_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_LINUX_H_INCLUDED_
9  #define _NGX_LINUX_H_INCLUDED_
10
11
12  ngx_chain_t *ngx_linux_sendfile_chain(ngx_connection_t *c, ngx_chain_t *in,
13      off_t limit);
14
15  extern int ngx_linux_rtsig_max;
16
17
18  #endif /* _NGX_LINUX_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_setaffinity.h - nginx-1.7.10

Macros defined

- NGX_HAVE_CPU_AFFINITY
- _NGX_SETAFFINITY_H_INCLUDED_
- ngx_setaffinity

Source code

```
1
2  /*
3  * Copyright (C) Nginx, Inc.
4  */
5
6  #ifndef _NGX_SETAFFINITY_H_INCLUDED_
7  #define _NGX_SETAFFINITY_H_INCLUDED_
8
9
10 #if (NGX_HAVE_SCHED_SETAFFINITY || NGX_HAVE_CPUSET_SETAFFINITY)
11
12 #define NGX_HAVE_CPU_AFFINITY 1
13
14 void ngx_setaffinity(uint64_t cpu_affinity, ngx_log_t *log);
15
16 #else
17
18 #define ngx_setaffinity(cpu_affinity, log)
19
20 #endif
21
22
23 #endif /* _NGX_SETAFFINITY_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_setproctitle.c - nginx-1.7.10

Global variables defined

- ngx_os_argv_last

Functions defined

- ngx_init_setproctitle
- ngx_setproctitle

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 #if (NGX_SETPROCTITLE_USES_ENV)
13
14 /*
15  * To change the process title in Linux and Solaris we have to set argv[1]
16  * to NULL and to copy the title to the same place where the argv[0] points to.
17  * However, argv[0] may be too small to hold a new title.  Fortunately, Linux
18  * and Solaris store argv[] and environ[] one after another.  So we should
19  * ensure that is the continuous memory and then we allocate the new memory
20  * for environ[] and copy it.  After this we could use the memory starting
21  * from argv[0] for our process title.
22  *
23  * The Solaris's standard /bin/ps does not show the changed process title.
24  * You have to use "/usr/ucb/ps -w" instead.  Besides, the UCB ps does not
25  * show a new title if its length less than the origin command line length.
26  * To avoid it we append to a new title the origin command line in the
27  * parenthesis.
28  */
29
30 extern char **environ;
31
32 static char *ngx_os_argv_last;
33
34 ngx_int_t
35 ngx_init_setproctitle(ngx_log_t *log)
36 {
37     u_char      *p;
38     size_t       size;
39     ngx_uint_t   i;
40
41     size = 0;
42
43     for (i = 0; environ[i]; i++) {
44         size += ngx_strlen(environ[i]) + 1;
45     }
46
47     p = ngx_alloc(size, log);
48     if (p == NULL) {
49         return NGX_ERROR;
50     }
51
52     ngx_os_argv_last = ngx_os_argv[0];
53
54     for (i = 0; ngx_os_argv[i]; i++) {
```

```
        if (ngx_os_argv_last == ngx_os_argv[i]) {
            ngx_os_argv_last = ngx_os_argv[i] + ngx_strlen(ngx_os_argv[i]) + 1;
        }
    }

    for (i = 0; environ[i]; i++) {
        if (ngx_os_argv_last == environ[i]) {

            size = ngx_strlen(environ[i]) + 1;
            ngx_os_argv_last = environ[i] + size;

            ngx_cpystrn(p, (u_char *) environ[i], size);
            environ[i] = (char *) p;
            p += size;
        }
    }

    ngx_os_argv_last--;

    return NGX_OK;
}


void
ngx_setproctitle(char *title)
{
    u_char     *p;

#if (NGX_SOLARIS)

    ngx_int_t   i;
    size_t      size;

#endif

    ngx_os_argv[1] = NULL;

    p = ngx_cpystrn((u_char *) ngx_os_argv[0], (u_char *) "nginx: ",
                    ngx_os_argv_last - ngx_os_argv[0]);

    p = ngx_cpystrn(p, (u_char *) title, ngx_os_argv_last - (char *) p);

#if (NGX_SOLARIS)

    size = 0;

     for (i = 0; i < ngx_argc; i++) {
         size += ngx_strlen(ngx_argv[i]) + 1;
     }

     if (size > (size_t) ((char *) p - ngx_os_argv[0])) {

         /*
          * ngx_setproctitle() is too rare operation so we use
          * the non-optimized copies
          */

         p = ngx_cpystrn(p, (u_char *) " (", ngx_os_argv_last - (char *) p);

         for (i = 0; i < ngx_argc; i++) {
             p = ngx_cpystrn(p, (u_char *) ngx_argv[i],
                             ngx_os_argv_last - (char *) p);
             p = ngx_cpystrn(p, (u_char *) " ", ngx_os_argv_last - (char *) p);
         }

         if (*(p - 1) == ' ') {
             *(p - 1) = ')';
         }
     }

 #endif

     if (ngx_os_argv_last - (char *) p) {
         ngx_memset(p, NGX_SETPROCTITLE_PAD, ngx_os_argv_last - (char *) p);
     }

```

```
131     ngx_log_debug1(NGX_LOG_DEBUG_CORE, ngx_cycle->log, 0,
132                    "setproctitle: \"%s\"", ngx_os_argv[0]);
133 }
134
135 #endif /* NGX_SETPROCTITLE_USES_ENV */
```

One Level Up Top Level

One Level Up Top Level

src/os/unix/ngx_solaris.h - nginx-1.7.10

Macros defined

- _NGX_SOLARIS_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_SOLARIS_H_INCLUDED_
9  #define _NGX_SOLARIS_H_INCLUDED_
10
11
12  ngx_chain_t *ngx_solaris_sendfilev_chain(ngx_connection_t *c, ngx_chain_t *in,
13      off_t limit);
14
15
16  #endif /* _NGX_SOLARIS_H_INCLUDED_ */
```

One Level Up Top Level

src/os/unix/ngx_solaris_init.c - nginx-1.7.10

Global variables defined

- ngx_solaris_io
- ngx_solaris_release
- ngx_solaris_sysname
- ngx_solaris_version

Functions defined

- ngx_os_specific_init
- ngx_os_specific_status

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12  char ngx_solaris_sysname[20];
13  char ngx_solaris_release[10];
14  char ngx_solaris_version[50];
15
16
17  static ngx_os_io_t ngx_solaris_io = {
18      ngx_unix_recv,
19      ngx_readv_chain,
20      ngx_udp_unix_recv,
21      ngx_unix_send,
22  #if (NGX_HAVE_SENDFILE)
23      ngx_solaris_sendfilev_chain,
24      NGX_IO_SENDFILE
25  #else
26      ngx_writev_chain,
27      0
28  #endif
29  };
30
31
32  ngx_int_t
33  ngx_os_specific_init(ngx_log_t *log)
34  {
35      if (sysinfo(SI_SYSNAME, ngx_solaris_sysname, sizeof(ngx_solaris_sysname))
36          == -1)
37      {
38          ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
39                        "sysinfo(SI_SYSNAME) failed");
40          return NGX_ERROR;
41      }
42
43      if (sysinfo(SI_RELEASE, ngx_solaris_release, sizeof(ngx_solaris_release))
44          == -1)
45      {
46          ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
47                        "sysinfo(SI_RELEASE) failed");
48          return NGX_ERROR;
```

```
    }

    if (sysinfo(SI_VERSION, ngx_solaris_version, sizeof(ngx_solaris_version))
        == -1)
    {
        ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                      "sysinfo(SI_SYSNAME) failed");
        return NGX_ERROR;
    }


    ngx_os_io = ngx_solaris_io;

    return NGX_OK;
}


void
ngx_os_specific_status(ngx_log_t *log)
{

    ngx_log_error(NGX_LOG_NOTICE, log, 0, "OS: %s %s",
                  ngx_solaris_sysname, ngx_solaris_release);

    ngx_log_error(NGX_LOG_NOTICE, log, 0, "version: %s",
                  ngx_solaris_version);
}
```

One Level Up Top Level

src/os/unix/ngx_sunpro_atomic_sparc64.h - nginx-1.7.10

Functions defined

- ngx_atomic_cmp_set
- ngx_atomic_fetch_add

Macros defined

- NGX_CASA
- NGX_CASA
- ngx_cpu_pause
- ngx_memory_barrier

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #if (NGX_PTR_SIZE == 4)
9  #define NGX_CASA  ngx_casa
10  #else
11  #define NGX_CASA  ngx_casxa
12  #endif
13
14
15  ngx_atomic_uint_t
16  ngx_casa(ngx_atomic_uint_t set, ngx_atomic_uint_t old, ngx_atomic_t *lock);
17
18  ngx_atomic_uint_t
19  ngx_casxa(ngx_atomic_uint_t set, ngx_atomic_uint_t old, ngx_atomic_t *lock);
20
21  /* the code in src/os/unix/ngx_sunpro_sparc64.il */
22
23
24  static ngx_inline ngx_atomic_uint_t
25  ngx_atomic_cmp_set(ngx_atomic_t *lock, ngx_atomic_uint_t old,
26      ngx_atomic_uint_t set)
27  {
28      set = NGX_CASA(set, old, lock);
29
30      return (set == old);
31  }
32
33
34  static ngx_inline ngx_atomic_int_t
35  ngx_atomic_fetch_add(ngx_atomic_t *value, ngx_atomic_int_t add)
36  {
37      ngx_atomic_uint_t  old, res;
38
39      old = *value;
40
41      for ( ;; ) {
42
43          res = old + add;
44
45          res = NGX_CASA(res, old, value);
46
47          if (res == old) {
48              return res;
```

```
        }

        old = res;
    }
}


#define ngx_memory_barrier()                                                 \
        __asm (".volatile");                                                 \
        __asm ("membar #LoadLoad | #LoadStore | #StoreStore | #StoreLoad");  \
        __asm (".nonvolatile")

#define ngx_cpu_pause()
```

One Level Up Top Level

src/os/unix/rfork_thread.S - nginx-1.7.10

Macros defined

- KERNCALL

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <sys/syscall.h>
9  #include <machine/asm.h>
10
11  /*
12   * rfork_thread(3) - rfork_thread(flags, stack, func, arg);
13   */
14
15  #define        KERNCALL        int $0x80
16
17  ENTRY(rfork_thread)
18          push        %ebp
19          mov        %esp, %ebp
20          push        %esi
21
22          mov        12(%ebp), %esi        # the thread stack address
23
24          sub        $4, %esi
25          mov        20(%ebp), %eax        # the thread argument
26          mov        %eax, (%esi)
27
28          sub        $4, %esi
29          mov        16(%ebp), %eax        # the thread start address
30          mov        %eax, (%esi)
31
32          push        8(%ebp)                # rfork(2) flags
33          push        $0
34          mov        $SYS_rfork, %eax
35          KERNCALL
36          jc        error
37
38          cmp        $0, %edx
39          jne        child
40
41  parent:
42          add        $8, %esp
43          pop        %esi
44          leave
45          ret
46
47  child:
48          mov        %esi, %esp
49          pop        %eax
50          call        *%eax                # call a thread start address ...
51          add        $4, %esp
52
53          push        %eax
54          push        $0
55          mov        $SYS_exit, %eax        # ... and exit(2) after a thread would return
56          KERNCALL
57
58  error:
59          add        $8, %esp
60          pop        %esi
61          leave
62          PIC_PROLOGUE
```

```
63
64          /* libc's cerror: jmp  PIC_PLT(HIDENAME(cerror)) */
65
66          push        %eax
67          call        PIC_PLT(CNAME(__error))
68          pop        %ecx
69          PIC_EPILOGUE
70          mov        %ecx, (%eax)
71          mov        $-1, %eax
72          mov        $-1, %edx
73          ret
```

One Level Up Top Level

src/core/ngx_crc.h - nginx-1.7.10

Functions defined

- ngx_crc

Macros defined

- _NGX_CRC_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_CRC_H_INCLUDED_
9  #define _NGX_CRC_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  /* 32-bit crc16 */
17
18  static ngx_inline uint32_t
19  ngx_crc(u_char *data, size_t len)
20  {
21      uint32_t  sum;
22
23      for (sum = 0; len; len--) {
24
25          /*
26           * gcc 2.95.2 x86 and icc 7.1.006 compile
27           * that operator into the single "rol" opcode,
28           * msvc 6.0sp2 compiles it into four opcodes.
29           */
30          sum = sum >> 1 | sum << 31;
31
32          sum += *data++;
33      }
34
35      return sum;
36  }
37
38
39  #endif /* _NGX_CRC_H_INCLUDED_ */
```

src/http/ngx_http_copy_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_copy_filter_commands
- ngx_http_copy_filter_module
- ngx_http_copy_filter_module_ctx
- ngx_http_next_body_filter

Data types defined

- ngx_http_copy_filter_conf_t

Functions defined

- ngx_http_copy_aio_event_handler
- ngx_http_copy_aio_handler
- ngx_http_copy_aio_sendfile_event_handler
- ngx_http_copy_filter
- ngx_http_copy_filter_create_conf
- ngx_http_copy_filter_init
- ngx_http_copy_filter_merge_conf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_bufs_t  bufs;
15  } ngx_http_copy_filter_conf_t;
16
17
18  #if (NGX_HAVE_FILE_AIO)
19  static void ngx_http_copy_aio_handler(ngx_output_chain_ctx_t *ctx,
20      ngx_file_t *file);
21  static void ngx_http_copy_aio_event_handler(ngx_event_t *ev);
22  #if (NGX_HAVE_AIO_SENDFILE)
23  static void ngx_http_copy_aio_sendfile_event_handler(ngx_event_t *ev);
24  #endif
25  #endif
26
27  static void *ngx_http_copy_filter_create_conf(ngx_conf_t *cf);
28  static char *ngx_http_copy_filter_merge_conf(ngx_conf_t *cf,
29      void *parent, void *child);
30  static ngx_int_t ngx_http_copy_filter_init(ngx_conf_t *cf);
```

```
31
32
33  static ngx_command_t  ngx_http_copy_filter_commands[] = {
34
35      { ngx_string("output_buffers"),
36        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
37        ngx_conf_set_bufs_slot,
38        NGX_HTTP_LOC_CONF_OFFSET,
39        offsetof(ngx_http_copy_filter_conf_t, bufs),
40        NULL },
41
42        ngx_null_command
43  };
44
45
46  static ngx_http_module_t  ngx_http_copy_filter_module_ctx = {
47      NULL,                                  /* preconfiguration */
48      ngx_http_copy_filter_init,             /* postconfiguration */
49
50      NULL,                                  /* create main configuration */
51      NULL,                                  /* init main configuration */
52
53      NULL,                                  /* create server configuration */
54      NULL,                                  /* merge server configuration */
55
56      ngx_http_copy_filter_create_conf,      /* create location configuration */
57      ngx_http_copy_filter_merge_conf        /* merge location configuration */
58  };
59
60
61  ngx_module_t  ngx_http_copy_filter_module = {
62      NGX_MODULE_V1,
63      &ngx_http_copy_filter_module_ctx,      /* module context */
64      ngx_http_copy_filter_commands,         /* module directives */
65      NGX_HTTP_MODULE,                       /* module type */
66      NULL,                                  /* init master */
67      NULL,                                  /* init module */
68      NULL,                                  /* init process */
69      NULL,                                  /* init thread */
70      NULL,                                  /* exit thread */
71      NULL,                                  /* exit process */
72      NULL,                                  /* exit master */
73      NGX_MODULE_V1_PADDING
74  };
75
76
77  static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;
78
79
80  static ngx_int_t
81  ngx_http_copy_filter(ngx_http_request_t *r, ngx_chain_t *in)
82  {
83      ngx_int_t                    rc;
84      ngx_connection_t            *c;
85      ngx_output_chain_ctx_t       *ctx;
86      ngx_http_core_loc_conf_t     *clcf;
87      ngx_http_copy_filter_conf_t  *conf;
88
89      c = r->connection;
90
91      ngx_log_debug2(NGX_LOG_DEBUG_HTTP, c->log, 0,
92                     "http copy filter: \"%V?%V\"", &r->uri, &r->args);
93
94      ctx = ngx_http_get_module_ctx(r, ngx_http_copy_filter_module);
95
96      if (ctx == NULL) {
97          ctx = ngx_pcalloc(r->pool, sizeof(ngx_output_chain_ctx_t));
98          if (ctx == NULL) {
99              return NGX_ERROR;
100         }
101
102         ngx_http_set_ctx(r, ctx, ngx_http_copy_filter_module);
103
104         conf = ngx_http_get_module_loc_conf(r, ngx_http_copy_filter_module);
105         clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
106
```

```
        ctx->sendfile = c->sendfile;
        ctx->need_in_memory = r->main_filter_need_in_memory
                              || r->filter_need_in_memory;
        ctx->need_in_temp = r->filter_need_temporary;

        ctx->alignment = clcf->directio_alignment;

        ctx->pool = r->pool;
        ctx->bufs = conf->bufs;
        ctx->tag = (ngx_buf_tag_t) &ngx_http_copy_filter_module;

        ctx->output_filter = (ngx_output_chain_filter_pt)
                                  ngx_http_next_body_filter;
        ctx->filter_ctx = r;

#if (NGX_HAVE_FILE_AIO)
        if (ngx_file_aio) {
            if (clcf->aio) {
                ctx->aio_handler = ngx_http_copy_aio_handler;
            }
#if (NGX_HAVE_AIO_SENDFILE)
            c->aio_sendfile = (clcf->aio == NGX_HTTP_AIO_SENDFILE);
#endif
        }
#endif

        if (in && in->buf && ngx_buf_size(in->buf)) {
            r->request_output = 1;
        }
    }

#if (NGX_HAVE_FILE_AIO)
    ctx->aio = r->aio;
#endif

    for ( ;; ) {
        rc = ngx_output_chain(ctx, in);

        if (ctx->in == NULL) {
            r->buffered &= ~NGX_HTTP_COPY_BUFFERED;

        } else {
            r->buffered |= NGX_HTTP_COPY_BUFFERED;
        }

        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, c->log, 0,
                       "http copy filter: %i \"%V?%V\"", rc, &r->uri, &r->args);

#if (NGX_HAVE_FILE_AIO && NGX_HAVE_AIO_SENDFILE)

        if (c->busy_sendfile) {
            ssize_t                n;
            off_t                  offset;
            ngx_file_t            *file;
            ngx_http_ephemeral_t  *e;

            if (r->aio) {
                c->busy_sendfile = NULL;
                return rc;
            }

            file = c->busy_sendfile->file;
            offset = c->busy_sendfile->file_pos;

            if (file->aio) {
                c->busy_count = (offset == file->aio->last_offset) ?
                                c->busy_count + 1 : 0;
                file->aio->last_offset = offset;

                if (c->busy_count > 2) {
                    ngx_log_error(NGX_LOG_ALERT, c->log, 0,
                                  "sendfile(%V) returned busy again",
                                  &file->name);
                    c->aio_sendfile = 0;
                }
            }
```

```
            c->busy_sendfile = NULL;
            e = (ngx_http_ephemeral_t *) &r->uri_start;

            n = ngx_file_aio_read(file, &e->aio_preload, 1, offset, r->pool);

            if (n > 0) {
                in = NULL;
                continue;
            }

            rc = n;

            if (rc == NGX_AGAIN) {
                file->aio->data = r;
                file->aio->handler = ngx_http_copy_aio_sendfile_event_handler;

                r->main->blocked++;
                r->aio = 1;
            }
        }
#endif

        return rc;
    }
}


#if (NGX_HAVE_FILE_AIO)

static void
ngx_http_copy_aio_handler(ngx_output_chain_ctx_t *ctx, ngx_file_t *file)
{
    ngx_http_request_t *r;

    r = ctx->filter_ctx;

    file->aio->data = r;
    file->aio->handler = ngx_http_copy_aio_event_handler;

    r->main->blocked++;
    r->aio = 1;
    ctx->aio = 1;
}


static void
ngx_http_copy_aio_event_handler(ngx_event_t *ev)
{
    ngx_event_aio_t     *aio;
    ngx_http_request_t  *r;

    aio = ev->data;
    r = aio->data;

    r->main->blocked--;
    r->aio = 0;

    r->connection->write->handler(r->connection->write);
}


#if (NGX_HAVE_AIO_SENDFILE)

static void
ngx_http_copy_aio_sendfile_event_handler(ngx_event_t *ev)
{
    ngx_event_aio_t     *aio;
    ngx_http_request_t  *r;

    aio = ev->data;
    r = aio->data;

    r->main->blocked--;
    r->aio = 0;
    ev->complete = 0;
```

```
259
260     r->connection->write->handler(r->connection->write);
261 }
262
263 #endif
264 #endif
265
266
267 static void *
268 ngx_http_copy_filter_create_conf(ngx_conf_t *cf)
269 {
270     ngx_http_copy_filter_conf_t *conf;
271
272     conf = ngx_palloc(cf->pool, sizeof(ngx_http_copy_filter_conf_t));
273     if (conf == NULL) {
274         return NULL;
275     }
276
277     conf->bufs.num = 0;
278
279     return conf;
280 }
281
282
283 static char *
284 ngx_http_copy_filter_merge_conf(ngx_conf_t *cf, void *parent, void *child)
285 {
286     ngx_http_copy_filter_conf_t *prev = parent;
287     ngx_http_copy_filter_conf_t *conf = child;
288
289     ngx_conf_merge_bufs_value(conf->bufs, prev->bufs, 1, 32768);
290
291     return NULL;
292 }
293
294
295 static ngx_int_t
296 ngx_http_copy_filter_init(ngx_conf_t *cf)
297 {
298     ngx_http_next_body_filter = ngx_http_top_body_filter;
299     ngx_http_top_body_filter = ngx_http_copy_filter;
300
301     return NGX_OK;
302 }
303
```

One Level Up Top Level

src/http/modules/ngx_http_access_module.c - nginx-1.7.10

Global variables defined

- ngx_http_access_commands
- ngx_http_access_module
- ngx_http_access_module_ctx

Data types defined

- ngx_http_access_loc_conf_t
- ngx_http_access_rule6_t
- ngx_http_access_rule_t
- ngx_http_access_rule_un_t

Functions defined

- ngx_http_access_create_loc_conf
- ngx_http_access_found
- ngx_http_access_handler
- ngx_http_access_inet
- ngx_http_access_inet6
- ngx_http_access_init
- ngx_http_access_merge_loc_conf
- ngx_http_access_rule
- ngx_http_access_unix

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      in_addr_t         mask;
15      in_addr_t         addr;
16      ngx_uint_t        deny;      /* unsigned  deny:1; */
17  } ngx_http_access_rule_t;
18
19  #if (NGX_HAVE_INET6)
20
21  typedef struct {
```

```
    struct in6_addr   addr;
    struct in6_addr   mask;
    ngx_uint_t        deny;      /* unsigned  deny:1; */
} ngx_http_access_rule6_t;

#endif

#if (NGX_HAVE_UNIX_DOMAIN)

typedef struct {
    ngx_uint_t        deny;      /* unsigned  deny:1; */
} ngx_http_access_rule_un_t;

#endif

typedef struct {
    ngx_array_t      *rules;     /* array of ngx_http_access_rule_t */
#if (NGX_HAVE_INET6)
    ngx_array_t      *rules6;    /* array of ngx_http_access_rule6_t */
#endif
#if (NGX_HAVE_UNIX_DOMAIN)
    ngx_array_t      *rules_un;  /* array of ngx_http_access_rule_un_t */
#endif
} ngx_http_access_loc_conf_t;


static ngx_int_t ngx_http_access_handler(ngx_http_request_t *r);
static ngx_int_t ngx_http_access_inet(ngx_http_request_t *r,
    ngx_http_access_loc_conf_t *alcf, in_addr_t addr);
#if (NGX_HAVE_INET6)
static ngx_int_t ngx_http_access_inet6(ngx_http_request_t *r,
    ngx_http_access_loc_conf_t *alcf, u_char *p);
#endif
#if (NGX_HAVE_UNIX_DOMAIN)
static ngx_int_t ngx_http_access_unix(ngx_http_request_t *r,
    ngx_http_access_loc_conf_t *alcf);
#endif
static ngx_int_t ngx_http_access_found(ngx_http_request_t *r, ngx_uint_t deny);
static char *ngx_http_access_rule(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static void *ngx_http_access_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_access_merge_loc_conf(ngx_conf_t *cf,
    void *parent, void *child);
static ngx_int_t ngx_http_access_init(ngx_conf_t *cf);


static ngx_command_t  ngx_http_access_commands[] = {

    { ngx_string("allow"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LMT_CONF
                        |NGX_CONF_TAKE1,
      ngx_http_access_rule,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("deny"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LMT_CONF
                        |NGX_CONF_TAKE1,
      ngx_http_access_rule,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};



static ngx_http_module_t  ngx_http_access_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_access_init,                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

```

```
    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_access_create_loc_conf,       /* create location configuration */
    ngx_http_access_merge_loc_conf         /* merge location configuration */
};


ngx_module_t  ngx_http_access_module = {
    NGX_MODULE_V1,
    &ngx_http_access_module_ctx,           /* module context */
    ngx_http_access_commands,              /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_access_handler(ngx_http_request_t *r)
{
    struct sockaddr_in          *sin;
    ngx_http_access_loc_conf_t  *alcf;
#if (NGX_HAVE_INET6)
    u_char                      *p;
    in_addr_t                    addr;
    struct sockaddr_in6         *sin6;
#endif

    alcf = ngx_http_get_module_loc_conf(r, ngx_http_access_module);

    switch (r->connection->sockaddr->sa_family) {

    case AF_INET:
        if (alcf->rules) {
            sin = (struct sockaddr_in *) r->connection->sockaddr;
            return ngx_http_access_inet(r, alcf, sin->sin_addr.s_addr);
        }
        break;

#if (NGX_HAVE_INET6)

    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) r->connection->sockaddr;
        p = sin6->sin6_addr.s6_addr;

        if (alcf->rules && IN6_IS_ADDR_V4MAPPED(&sin6->sin6_addr)) {
            addr = p[12] << 24;
            addr += p[13] << 16;
            addr += p[14] << 8;
            addr += p[15];
            return ngx_http_access_inet(r, alcf, htonl(addr));
        }

        if (alcf->rules6) {
            return ngx_http_access_inet6(r, alcf, p);
        }

        break;

#endif

#if (NGX_HAVE_UNIX_DOMAIN)

    case AF_UNIX:
        if (alcf->rules_un) {
            return ngx_http_access_unix(r, alcf);
        }

        break;
```

```
174
175 #endif
176     }
177
178     return NGX_DECLINED;
179 }
180
181
182 static ngx_int_t
183 ngx_http_access_inet(ngx_http_request_t *r, ngx_http_access_loc_conf_t *alcf,
184     in_addr_t addr)
185 {
186     ngx_uint_t               i;
187     ngx_http_access_rule_t  *rule;
188
189     rule = alcf->rules->elts;
190     for (i = 0; i < alcf->rules->nelts; i++) {
191
192         ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
193                        "access: %08XD %08XD %08XD",
194                        addr, rule[i].mask, rule[i].addr);
195
196         if ((addr & rule[i].mask) == rule[i].addr) {
197             return ngx_http_access_found(r, rule[i].deny);
198         }
199     }
200
201     return NGX_DECLINED;
202 }
203
204
205 #if (NGX_HAVE_INET6)
206
207 static ngx_int_t
208 ngx_http_access_inet6(ngx_http_request_t *r, ngx_http_access_loc_conf_t *alcf,
209     u_char *p)
210 {
211     ngx_uint_t               n;
212     ngx_uint_t               i;
213     ngx_http_access_rule6_t  *rule6;
214
215     rule6 = alcf->rules6->elts;
216     for (i = 0; i < alcf->rules6->nelts; i++) {
217
218 #if (NGX_DEBUG)
219         {
220         size_t  cl, ml, al;
221         u_char  ct[NGX_INET6_ADDRSTRLEN];
222         u_char  mt[NGX_INET6_ADDRSTRLEN];
223         u_char  at[NGX_INET6_ADDRSTRLEN];
224
225         cl = ngx_inet6_ntop(p, ct, NGX_INET6_ADDRSTRLEN);
226         ml = ngx_inet6_ntop(rule6[i].mask.s6_addr, mt, NGX_INET6_ADDRSTRLEN);
227         al = ngx_inet6_ntop(rule6[i].addr.s6_addr, at, NGX_INET6_ADDRSTRLEN);
228
229         ngx_log_debug6(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
230                        "access: %*s %*s %*s", cl, ct, ml, mt, al, at);
231         }
232 #endif
233
234         for (n = 0; n < 16; n++) {
235             if ((p[n] & rule6[i].mask.s6_addr[n]) != rule6[i].addr.s6_addr[n]) {
236                 goto next;
237             }
238         }
239
240         return ngx_http_access_found(r, rule6[i].deny);
241
242     next:
243         continue;
244     }
245
246     return NGX_DECLINED;
247 }
248
249 #endif
```

```
250
251
252 #if (NGX_HAVE_UNIX_DOMAIN)
253
254 static ngx_int_t
255 ngx_http_access_unix(ngx_http_request_t *r, ngx_http_access_loc_conf_t *alcf)
256 {
257     ngx_uint_t                  i;
258     ngx_http_access_rule_un_t  *rule_un;
259
260     rule_un = alcf->rules_un->elts;
261     for (i = 0; i < alcf->rules_un->nelts; i++) {
262
263         /* TODO: check path */
264         if (1) {
265             return ngx_http_access_found(r, rule_un[i].deny);
266         }
267     }
268
269     return NGX_DECLINED;
270 }
271
272 #endif
273
274
275 static ngx_int_t
276 ngx_http_access_found(ngx_http_request_t *r, ngx_uint_t deny)
277 {
278     ngx_http_core_loc_conf_t  *clcf;
279
280     if (deny) {
281         clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
282
283         if (clcf->satisfy == NGX_HTTP_SATISFY_ALL) {
284             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
285                           "access forbidden by rule");
286         }
287
288         return NGX_HTTP_FORBIDDEN;
289     }
290
291     return NGX_OK;
292 }
293
294
295 static char *
296 ngx_http_access_rule(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
297 {
298     ngx_http_access_loc_conf_t *alcf = conf;
299
300     ngx_int_t                   rc;
301     ngx_uint_t                  all;
302     ngx_str_t                  *value;
303     ngx_cidr_t                  cidr;
304     ngx_http_access_rule_t     *rule;
305 #if (NGX_HAVE_INET6)
306     ngx_http_access_rule6_t    *rule6;
307 #endif
308 #if (NGX_HAVE_UNIX_DOMAIN)
309     ngx_http_access_rule_un_t  *rule_un;
310 #endif
311
312     ngx_memzero(&cidr, sizeof(ngx_cidr_t));
313
314     value = cf->args->elts;
315
316     all = (value[1].len == 3 && ngx_strcmp(value[1].data, "all") == 0);
317
318     if (!all) {
319
320 #if (NGX_HAVE_UNIX_DOMAIN)
321
322         if (value[1].len == 5 && ngx_strcmp(value[1].data, "unix:") == 0) {
323             cidr.family = AF_UNIX;
324             rc = NGX_OK;
325
```

```
        } else {
            rc = ngx_ptocidr(&value[1], &cidr);
        }

#else
        rc = ngx_ptocidr(&value[1], &cidr);
#endif

        if (rc == NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid parameter \"%V\"", &value[1]);
            return NGX_CONF_ERROR;
        }

        if (rc == NGX_DONE) {
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                               "low address bits of %V are meaningless", &value[1]);
        }
    }

    if (cidr.family == AF_INET || all) {

        if (alcf->rules == NULL) {
            alcf->rules = ngx_array_create(cf->pool, 4,
                                           sizeof(ngx_http_access_rule_t));
            if (alcf->rules == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        rule = ngx_array_push(alcf->rules);
        if (rule == NULL) {
            return NGX_CONF_ERROR;
        }

        rule->mask = cidr.u.in.mask;
        rule->addr = cidr.u.in.addr;
        rule->deny = (value[0].data[0] == 'd') ? 1 : 0;
    }

#if (NGX_HAVE_INET6)
    if (cidr.family == AF_INET6 || all) {

        if (alcf->rules6 == NULL) {
            alcf->rules6 = ngx_array_create(cf->pool, 4,
                                            sizeof(ngx_http_access_rule6_t));
            if (alcf->rules6 == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        rule6 = ngx_array_push(alcf->rules6);
        if (rule6 == NULL) {
            return NGX_CONF_ERROR;
        }

        rule6->mask = cidr.u.in6.mask;
        rule6->addr = cidr.u.in6.addr;
        rule6->deny = (value[0].data[0] == 'd') ? 1 : 0;
    }
#endif

#if (NGX_HAVE_UNIX_DOMAIN)
    if (cidr.family == AF_UNIX || all) {

        if (alcf->rules_un == NULL) {
            alcf->rules_un = ngx_array_create(cf->pool, 1,
                                              sizeof(ngx_http_access_rule_un_t));
            if (alcf->rules_un == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        rule_un = ngx_array_push(alcf->rules_un);
        if (rule_un == NULL) {
            return NGX_CONF_ERROR;
```

```
402         }
403
404         rule_un->deny = (value[0].data[0] == 'd') ? 1 : 0;
405     }
406 #endif
407
408     return NGX_CONF_OK;
409 }
410
411
412 static void *
413 ngx_http_access_create_loc_conf(ngx_conf_t *cf)
414 {
415     ngx_http_access_loc_conf_t  *conf;
416
417     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_access_loc_conf_t));
418     if (conf == NULL) {
419         return NULL;
420     }
421
422     return conf;
423 }
424
425
426 static char *
427 ngx_http_access_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
428 {
429     ngx_http_access_loc_conf_t  *prev = parent;
430     ngx_http_access_loc_conf_t  *conf = child;
431
432     if (conf->rules == NULL
433 #if (NGX_HAVE_INET6)
434         && conf->rules6 == NULL
435 #endif
436 #if (NGX_HAVE_UNIX_DOMAIN)
437         && conf->rules_un == NULL
438 #endif
439     ) {
440         conf->rules = prev->rules;
441 #if (NGX_HAVE_INET6)
442         conf->rules6 = prev->rules6;
443 #endif
444 #if (NGX_HAVE_UNIX_DOMAIN)
445         conf->rules_un = prev->rules_un;
446 #endif
447     }
448
449     return NGX_CONF_OK;
450 }
451
452
453 static ngx_int_t
454 ngx_http_access_init(ngx_conf_t *cf)
455 {
456     ngx_http_handler_pt        *h;
457     ngx_http_core_main_conf_t  *cmcf;
458
459     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
460
461     h = ngx_array_push(&cmcf->phases[NGX_HTTP_ACCESS_PHASE].handlers);
462     if (h == NULL) {
463         return NGX_ERROR;
464     }
465
466     *h = ngx_http_access_handler;
467
468     return NGX_OK;
469 }
```

One Level Up Top Level

src/http/modules/ngx_http_addition_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_addition_commands
- ngx_http_addition_filter_module
- ngx_http_addition_filter_module_ctx
- ngx_http_next_body_filter
- ngx_http_next_header_filter

Data types defined

- ngx_http_addition_conf_t
- ngx_http_addition_ctx_t

Functions defined

- ngx_http_addition_body_filter
- ngx_http_addition_create_conf
- ngx_http_addition_filter_init
- ngx_http_addition_header_filter
- ngx_http_addition_merge_conf

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_str_t     before_body;
15      ngx_str_t     after_body;
16
17      ngx_hash_t    types;
18      ngx_array_t  *types_keys;
19  } ngx_http_addition_conf_t;
20
21
22  typedef struct {
23      ngx_uint_t    before_body_sent;
24  } ngx_http_addition_ctx_t;
25
26
27  static void *ngx_http_addition_create_conf(ngx_conf_t *cf);
28  static char *ngx_http_addition_merge_conf(ngx_conf_t *cf, void *parent,
29      void *child);
30  static ngx_int_t ngx_http_addition_filter_init(ngx_conf_t *cf);
```

```
31
32
33  static ngx_command_t  ngx_http_addition_commands[] = {
34
35      { ngx_string("add_before_body"),
36        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
37        ngx_conf_set_str_slot,
38        NGX_HTTP_LOC_CONF_OFFSET,
39        offsetof(ngx_http_addition_conf_t, before_body),
40        NULL },
41
42      { ngx_string("add_after_body"),
43        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
44        ngx_conf_set_str_slot,
45        NGX_HTTP_LOC_CONF_OFFSET,
46        offsetof(ngx_http_addition_conf_t, after_body),
47        NULL },
48
49      { ngx_string("addition_types"),
50        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
51        ngx_http_types_slot,
52        NGX_HTTP_LOC_CONF_OFFSET,
53        offsetof(ngx_http_addition_conf_t, types_keys),
54        &ngx_http_html_default_types[0] },
55
56        ngx_null_command
57  };
58
59
60  static ngx_http_module_t  ngx_http_addition_filter_module_ctx = {
61      NULL,                                  /* preconfiguration */
62      ngx_http_addition_filter_init,         /* postconfiguration */
63
64      NULL,                                  /* create main configuration */
65      NULL,                                  /* init main configuration */
66
67      NULL,                                  /* create server configuration */
68      NULL,                                  /* merge server configuration */
69
70      ngx_http_addition_create_conf,         /* create location configuration */
71      ngx_http_addition_merge_conf           /* merge location configuration */
72  };
73
74
75  ngx_module_t  ngx_http_addition_filter_module = {
76      NGX_MODULE_V1,
77      &ngx_http_addition_filter_module_ctx,  /* module context */
78      ngx_http_addition_commands,            /* module directives */
79      NGX_HTTP_MODULE,                       /* module type */
80      NULL,                                  /* init master */
81      NULL,                                  /* init module */
82      NULL,                                  /* init process */
83      NULL,                                  /* init thread */
84      NULL,                                  /* exit thread */
85      NULL,                                  /* exit process */
86      NULL,                                  /* exit master */
87      NGX_MODULE_V1_PADDING
88  };
89
90
91  static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
92  static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;
93
94
95  static ngx_int_t
96  ngx_http_addition_header_filter(ngx_http_request_t *r)
97  {
98      ngx_http_addition_ctx_t   *ctx;
99      ngx_http_addition_conf_t  *conf;
100
101      if (r->headers_out.status != NGX_HTTP_OK || r != r->main) {
102          return ngx_http_next_header_filter(r);
103      }
104
105      conf = ngx_http_get_module_loc_conf(r, ngx_http_addition_filter_module);
106
```

```
    if (conf->before_body.len == 0 && conf->after_body.len == 0) {
        return ngx_http_next_header_filter(r);
    }

    if (ngx_http_test_content_type(r, &conf->types) == NULL) {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_addition_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_addition_filter_module);

    ngx_http_clear_content_length(r);
    ngx_http_clear_accept_ranges(r);
    ngx_http_weak_etag(r);

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_addition_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    ngx_int_t                  rc;
    ngx_uint_t                 last;
    ngx_chain_t               *cl;
    ngx_http_request_t        *sr;
    ngx_http_addition_ctx_t   *ctx;
    ngx_http_addition_conf_t  *conf;

    if (in == NULL || r->header_only) {
        return ngx_http_next_body_filter(r, in);
    }

    ctx = ngx_http_get_module_ctx(r, ngx_http_addition_filter_module);

    if (ctx == NULL) {
        return ngx_http_next_body_filter(r, in);
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_addition_filter_module);

    if (!ctx->before_body_sent) {
        ctx->before_body_sent = 1;

        if (conf->before_body.len) {
            if (ngx_http_subrequest(r, &conf->before_body, NULL, &sr, NULL, 0)
                != NGX_OK)
            {
                return NGX_ERROR;
            }
        }
    }

    if (conf->after_body.len == 0) {
        ngx_http_set_ctx(r, NULL, ngx_http_addition_filter_module);
        return ngx_http_next_body_filter(r, in);
    }

    last = 0;

    for (cl = in; cl; cl = cl->next) {
        if (cl->buf->last_buf) {
            cl->buf->last_buf = 0;
            cl->buf->sync = 1;
            last = 1;
        }
    }

    rc = ngx_http_next_body_filter(r, in);

    if (rc == NGX_ERROR || !last || conf->after_body.len == 0) {
        return rc;
```

```
183     }
184
185     if (ngx_http_subrequest(r, &conf->after_body, NULL, &sr, NULL, 0)
186         != NGX_OK)
187     {
188         return NGX_ERROR;
189     }
190
191     ngx_http_set_ctx(r, NULL, ngx_http_addition_filter_module);
192
193     return ngx_http_send_special(r, NGX_HTTP_LAST);
194 }
195
196
197 static ngx_int_t
198 ngx_http_addition_filter_init(ngx_conf_t *cf)
199 {
200     ngx_http_next_header_filter = ngx_http_top_header_filter;
201     ngx_http_top_header_filter = ngx_http_addition_header_filter;
202
203     ngx_http_next_body_filter = ngx_http_top_body_filter;
204     ngx_http_top_body_filter = ngx_http_addition_body_filter;
205
206     return NGX_OK;
207 }
208
209
210 static void *
211 ngx_http_addition_create_conf(ngx_conf_t *cf)
212 {
213     ngx_http_addition_conf_t  *conf;
214
215     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_addition_conf_t));
216     if (conf == NULL) {
217         return NULL;
218     }
219
220     /*
221      * set by ngx_pcalloc():
222      *
223      *     conf->before_body = { 0, NULL };
224      *     conf->after_body = { 0, NULL };
225      *     conf->types = { NULL };
226      *     conf->types_keys = NULL;
227      */
228
229     return conf;
230 }
231
232
233 static char *
234 ngx_http_addition_merge_conf(ngx_conf_t *cf, void *parent, void *child)
235 {
236     ngx_http_addition_conf_t *prev = parent;
237     ngx_http_addition_conf_t *conf = child;
238
239     ngx_conf_merge_str_value(conf->before_body, prev->before_body, "");
240     ngx_conf_merge_str_value(conf->after_body, prev->after_body, "");
241
242     if (ngx_http_merge_types(cf, &conf->types_keys, &conf->types,
243                              &prev->types_keys, &prev->types,
244                              ngx_http_html_default_types)
245         != NGX_OK)
246     {
247         return NGX_CONF_ERROR;
248     }
249
250     return NGX_CONF_OK;
251 }
```

One Level Up Top Level

src/http/modules/ngx_http_auth_basic_module.c - nginx-1.7.10

Global variables defined

- ngx_http_auth_basic_commands
- ngx_http_auth_basic_module
- ngx_http_auth_basic_module_ctx

Data types defined

- ngx_http_auth_basic_ctx_t
- ngx_http_auth_basic_loc_conf_t

Functions defined

- ngx_http_auth_basic_close
- ngx_http_auth_basic_create_loc_conf
- ngx_http_auth_basic_crypt_handler
- ngx_http_auth_basic_handler
- ngx_http_auth_basic_init
- ngx_http_auth_basic_merge_loc_conf
- ngx_http_auth_basic_set_realm
- ngx_http_auth_basic_user_file

Macros defined

- NGX_HTTP_AUTH_BUF_SIZE

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <ngx_crypt.h>
12
13
14  #define NGX_HTTP_AUTH_BUF_SIZE  2048
15
16
17  typedef struct {
18      ngx_str_t                 passwd;
19  } ngx_http_auth_basic_ctx_t;
20
21
22  typedef struct {
```

```
    ngx_http_complex_value_t  *realm;
    ngx_http_complex_value_t   user_file;
} ngx_http_auth_basic_loc_conf_t;


static ngx_int_t ngx_http_auth_basic_handler(ngx_http_request_t *r);
static ngx_int_t ngx_http_auth_basic_crypt_handler(ngx_http_request_t *r,
    ngx_http_auth_basic_ctx_t *ctx, ngx_str_t *passwd, ngx_str_t *realm);
static ngx_int_t ngx_http_auth_basic_set_realm(ngx_http_request_t *r,
    ngx_str_t *realm);
static void ngx_http_auth_basic_close(ngx_file_t *file);
static void *ngx_http_auth_basic_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_auth_basic_merge_loc_conf(ngx_conf_t *cf,
    void *parent, void *child);
static ngx_int_t ngx_http_auth_basic_init(ngx_conf_t *cf);
static char *ngx_http_auth_basic_user_file(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


static ngx_command_t  ngx_http_auth_basic_commands[] = {

    { ngx_string("auth_basic"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LMT_CONF
                        |NGX_CONF_TAKE1,
      ngx_http_set_complex_value_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_auth_basic_loc_conf_t, realm),
      NULL },

    { ngx_string("auth_basic_user_file"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LMT_CONF
                        |NGX_CONF_TAKE1,
      ngx_http_auth_basic_user_file,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_auth_basic_loc_conf_t, user_file),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_auth_basic_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_auth_basic_init,              /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_auth_basic_create_loc_conf,   /* create location configuration */
    ngx_http_auth_basic_merge_loc_conf     /* merge location configuration */
};


ngx_module_t  ngx_http_auth_basic_module = {
    NGX_MODULE_V1,
    &ngx_http_auth_basic_module_ctx,       /* module context */
    ngx_http_auth_basic_commands,          /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_auth_basic_handler(ngx_http_request_t *r)
{
    off_t                           offset;
```

```
    ssize_t                         n;
    ngx_fd_t                        fd;
    ngx_int_t                       rc;
    ngx_err_t                       err;
    ngx_str_t                       pwd, realm, user_file;
    ngx_uint_t                      i, level, login, left, passwd;
    ngx_file_t                      file;
    ngx_http_auth_basic_ctx_t       *ctx;
    ngx_http_auth_basic_loc_conf_t  *alcf;
    u_char                          buf[NGX_HTTP_AUTH_BUF_SIZE];
    enum {
        sw_login,
        sw_passwd,
        sw_skip
    } state;

    alcf = ngx_http_get_module_loc_conf(r, ngx_http_auth_basic_module);

    if (alcf->realm == NULL || alcf->user_file.value.data == NULL) {
        return NGX_DECLINED;
    }

    if (ngx_http_complex_value(r, alcf->realm, &realm) != NGX_OK) {
        return NGX_ERROR;
    }

    if (realm.len == 3 && ngx_strncmp(realm.data, "off", 3) == 0) {
        return NGX_DECLINED;
    }

    ctx = ngx_http_get_module_ctx(r, ngx_http_auth_basic_module);

    if (ctx) {
        return ngx_http_auth_basic_crypt_handler(r, ctx, &ctx->passwd,
                                                 &realm);
    }

    rc = ngx_http_auth_basic_user(r);

    if (rc == NGX_DECLINED) {

        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "no user/password was provided for basic authentication");

        return ngx_http_auth_basic_set_realm(r, &realm);
    }

    if (rc == NGX_ERROR) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (ngx_http_complex_value(r, &alcf->user_file, &user_file) != NGX_OK) {
        return NGX_ERROR;
    }

    fd = ngx_open_file(user_file.data, NGX_FILE_RDONLY, NGX_FILE_OPEN, 0);

    if (fd == NGX_INVALID_FILE) {
        err = ngx_errno;

        if (err == NGX_ENOENT) {
            level = NGX_LOG_ERR;
            rc = NGX_HTTP_FORBIDDEN;

        } else {
            level = NGX_LOG_CRIT;
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        ngx_log_error(level, r->connection->log, err,
                      ngx_open_file_n " \"%s\" failed", user_file.data);

        return rc;
    }

    ngx_memzero(&file, sizeof(ngx_file_t));
```

```
    file.fd = fd;
    file.name = user_file;
    file.log = r->connection->log;

    state = sw_login;
    passwd = 0;
    login = 0;
    left = 0;
    offset = 0;

    for ( ;; ) {
        i = left;

        n = ngx_read_file(&file, buf + left, NGX_HTTP_AUTH_BUF_SIZE - left,
                          offset);

        if (n == NGX_ERROR) {
            ngx_http_auth_basic_close(&file);
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        if (n == 0) {
            break;
        }

        for (i = left; i < left + n; i++) {
            switch (state) {

            case sw_login:
                if (login == 0) {

                    if (buf[i] == '#' || buf[i] == CR) {
                        state = sw_skip;
                        break;
                    }

                    if (buf[i] == LF) {
                        break;
                    }
                }

                if (buf[i] != r->headers_in.user.data[login]) {
                    state = sw_skip;
                    break;
                }

                if (login == r->headers_in.user.len) {
                    state = sw_passwd;
                    passwd = i + 1;
                }

                login++;

                break;

            case sw_passwd:
                if (buf[i] == LF || buf[i] == CR || buf[i] == ':') {
                    buf[i] = '\0';

                    ngx_http_auth_basic_close(&file);

                    pwd.len = i - passwd;
                    pwd.data = &buf[passwd];

                    return ngx_http_auth_basic_crypt_handler(r, NULL, &pwd,
                                                             &realm);
                }

                break;

            case sw_skip:
                if (buf[i] == LF) {
                    state = sw_login;
                    login = 0;
                }
```

```
                break;
            }
        }

        if (state == sw_passwd) {
            left = left + n - passwd;
            ngx_memmove(buf, &buf[passwd], left);
            passwd = 0;

        } else {
            left = 0;
        }

        offset += n;
    }

    ngx_http_auth_basic_close(&file);

    if (state == sw_passwd) {
        pwd.len = i - passwd;
        pwd.data = ngx_pnalloc(r->pool, pwd.len + 1);
        if (pwd.data == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        ngx_cpystrn(pwd.data, &buf[passwd], pwd.len + 1);

        return ngx_http_auth_basic_crypt_handler(r, NULL, &pwd, &realm);
    }

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "user \"%V\" was not found in \"%V\"",
                  &r->headers_in.user, &user_file);

    return ngx_http_auth_basic_set_realm(r, &realm);
}


static ngx_int_t
ngx_http_auth_basic_crypt_handler(ngx_http_request_t *r,
    ngx_http_auth_basic_ctx_t *ctx, ngx_str_t *passwd, ngx_str_t *realm)
{
    ngx_int_t   rc;
    u_char     *encrypted;

    rc = ngx_crypt(r->pool, r->headers_in.passwd.data, passwd->data,
                   &encrypted);

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "rc: %d user: \"%V\" salt: \"%s\"",
                   rc, &r->headers_in.user, passwd->data);

    if (rc == NGX_OK) {
        if (ngx_strcmp(encrypted, passwd->data) == 0) {
            return NGX_OK;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "encrypted: \"%s\"", encrypted);

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "user \"%V\": password mismatch",
                      &r->headers_in.user);

        return ngx_http_auth_basic_set_realm(r, realm);
    }

    if (rc == NGX_ERROR) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    /* rc == NGX_AGAIN */

    if (ctx == NULL) {
        ctx = ngx_palloc(r->pool, sizeof(ngx_http_auth_basic_ctx_t));
```

```
        if (ctx == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        ngx_http_set_ctx(r, ctx, ngx_http_auth_basic_module);

        ctx->passwd.len = passwd->len;
        passwd->len++;

        ctx->passwd.data = ngx_pstrdup(r->pool, passwd);
        if (ctx->passwd.data == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

    }

    /* TODO: add mutex event */

    return rc;
}


static ngx_int_t
ngx_http_auth_basic_set_realm(ngx_http_request_t *r, ngx_str_t *realm)
{
    size_t   len;
    u_char  *basic, *p;

    r->headers_out.www_authenticate = ngx_list_push(&r->headers_out.headers);
    if (r->headers_out.www_authenticate == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    len = sizeof("Basic realm=\"\"") - 1 + realm->len;

    basic = ngx_pnalloc(r->pool, len);
    if (basic == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    p = ngx_cpymem(basic, "Basic realm=\"", sizeof("Basic realm=\"") - 1);
    p = ngx_cpymem(p, realm->data, realm->len);
    *p = '"';

    r->headers_out.www_authenticate->hash = 1;
    ngx_str_set(&r->headers_out.www_authenticate->key, "WWW-Authenticate");
    r->headers_out.www_authenticate->value.data = basic;
    r->headers_out.www_authenticate->value.len = len;

    return NGX_HTTP_UNAUTHORIZED;
}

static void
ngx_http_auth_basic_close(ngx_file_t *file)
{
    if (ngx_close_file(file->fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, file->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", file->name.data);
    }
}


static void *
ngx_http_auth_basic_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_auth_basic_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_auth_basic_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    return conf;
}
```

```
403 static char *
404 ngx_http_auth_basic_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
405 {
406     ngx_http_auth_basic_loc_conf_t  *prev = parent;
407     ngx_http_auth_basic_loc_conf_t  *conf = child;
408 
409     if (conf->realm == NULL) {
410         conf->realm = prev->realm;
411     }
412 
413     if (conf->user_file.value.data == NULL) {
414         conf->user_file = prev->user_file;
415     }
416 
417     return NGX_CONF_OK;
418 }
419 
420 
421 static ngx_int_t
422 ngx_http_auth_basic_init(ngx_conf_t *cf)
423 {
424     ngx_http_handler_pt        *h;
425     ngx_http_core_main_conf_t  *cmcf;
426 
427     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
428 
429     h = ngx_array_push(&cmcf->phases[NGX_HTTP_ACCESS_PHASE].handlers);
430     if (h == NULL) {
431         return NGX_ERROR;
432     }
433 
434     *h = ngx_http_auth_basic_handler;
435 
436     return NGX_OK;
437 }
438 
439 
440 static char *
441 ngx_http_auth_basic_user_file(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
442 {
443     ngx_http_auth_basic_loc_conf_t *alcf = conf;
444 
445     ngx_str_t                         *value;
446     ngx_http_compile_complex_value_t   ccv;
447 
448     if (alcf->user_file.value.data) {
449         return "is duplicate";
450     }
451 
452     value = cf->args->elts;
453 
454     ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
455 
456     ccv.cf = cf;
457     ccv.value = &value[1];
458     ccv.complex_value = &alcf->user_file;
459     ccv.zero = 1;
460     ccv.conf_prefix = 1;
461 
462     if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
463         return NGX_CONF_ERROR;
464     }
465 
466     return NGX_CONF_OK;
467 }
```

One Level Up Top Level

src/core/ngx_crypt.c - nginx-1.7.10

Functions defined

- ngx_crypt
- ngx_crypt_apr1
- ngx_crypt_plain
- ngx_crypt_sha
- ngx_crypt_ssha
- ngx_crypt_to64

Source code

```
1
2  /*
3  * Copyright (C) Maxim Dounin
4  */
5
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9  #include <ngx_crypt.h>
10 #include <ngx_md5.h>
11 #if (NGX_HAVE_SHA1)
12 #include <ngx_sha1.h>
13 #endif
14
15
16 #if (NGX_CRYPT)
17
18 static ngx_int_t ngx_crypt_apr1(ngx_pool_t *pool, u_char *key, u_char *salt,
19     u_char **encrypted);
20 static ngx_int_t ngx_crypt_plain(ngx_pool_t *pool, u_char *key, u_char *salt,
21     u_char **encrypted);
22
23 #if (NGX_HAVE_SHA1)
24
25 static ngx_int_t ngx_crypt_ssha(ngx_pool_t *pool, u_char *key, u_char *salt,
26     u_char **encrypted);
27 static ngx_int_t ngx_crypt_sha(ngx_pool_t *pool, u_char *key, u_char *salt,
28     u_char **encrypted);
29
30 #endif
31
32
33 static u_char *ngx_crypt_to64(u_char *p, uint32_t v, size_t n);
34
35
36 ngx_int_t
37 ngx_crypt(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
38 {
39     if (ngx_strncmp(salt, "$apr1$", sizeof("$apr1$") - 1) == 0) {
40         return ngx_crypt_apr1(pool, key, salt, encrypted);
41
42     } else if (ngx_strncmp(salt, "{PLAIN}", sizeof("{PLAIN}") - 1) == 0) {
43         return ngx_crypt_plain(pool, key, salt, encrypted);
44
45 #if (NGX_HAVE_SHA1)
46     } else if (ngx_strncmp(salt, "{SSHA}", sizeof("{SSHA}") - 1) == 0) {
47         return ngx_crypt_ssha(pool, key, salt, encrypted);
48
49     } else if (ngx_strncmp(salt, "{SHA}", sizeof("{SHA}") - 1) == 0) {
50         return ngx_crypt_sha(pool, key, salt, encrypted);
51 #endif
```

```
    }

    /* fallback to libc crypt() */

    return ngx_libc_crypt(pool, key, salt, encrypted);
}


static ngx_int_t
ngx_crypt_apr1(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
{
    ngx_int_t          n;
    ngx_uint_t         i;
    u_char            *p, *last, final[16];
    size_t             saltlen, keylen;
    ngx_md5_t          md5, ctx1;

    /* Apache's apr1 crypt is Poul-Henning Kamp's md5 crypt with $apr1$ magic */

    keylen = ngx_strlen(key);

    /* true salt: no magic, max 8 chars, stop at first $ */

    salt += sizeof("$apr1$") - 1;
    last = salt + 8;
    for (p = salt; *p && *p != '$' && p < last; p++) { /* void */ }
    saltlen = p - salt;

    /* hash key and salt */

    ngx_md5_init(&md5);
    ngx_md5_update(&md5, key, keylen);
    ngx_md5_update(&md5, (u_char *) "$apr1$", sizeof("$apr1$") - 1);
    ngx_md5_update(&md5, salt, saltlen);

    ngx_md5_init(&ctx1);
    ngx_md5_update(&ctx1, key, keylen);
    ngx_md5_update(&ctx1, salt, saltlen);
    ngx_md5_update(&ctx1, key, keylen);
    ngx_md5_final(final, &ctx1);

    for (n = keylen; n > 0; n -= 16) {
        ngx_md5_update(&md5, final, n > 16 ? 16 : n);
    }

    ngx_memzero(final, sizeof(final));

    for (i = keylen; i; i >>= 1) {
         if (i & 1) {
             ngx_md5_update(&md5, final, 1);

         } else {
             ngx_md5_update(&md5, key, 1);
         }
     }

     ngx_md5_final(final, &md5);

     for (i = 0; i < 1000; i++) {
         ngx_md5_init(&ctx1);

         if (i & 1) {
             ngx_md5_update(&ctx1, key, keylen);

         } else {
             ngx_md5_update(&ctx1, final, 16);
         }

         if (i % 3) {
             ngx_md5_update(&ctx1, salt, saltlen);
         }

         if (i % 7) {
             ngx_md5_update(&ctx1, key, keylen);
         }

```

```
        if (i & 1) {
            ngx_md5_update(&ctx1, final, 16);

        } else {
            ngx_md5_update(&ctx1, key, keylen);
        }

        ngx_md5_final(final, &ctx1);
    }

    /* output */

    *encrypted = ngx_pnalloc(pool, sizeof("$apr1$") - 1 + saltlen + 1 + 22 + 1);
    if (*encrypted == NULL) {
        return NGX_ERROR;
    }

    p = ngx_cpymem(*encrypted, "$apr1$", sizeof("$apr1$") - 1);
    p = ngx_copy(p, salt, saltlen);
    *p++ = '$';

    p = ngx_crypt_to64(p, (final[ 0]<<16) | (final[ 6]<<8) | final[12], 4);
    p = ngx_crypt_to64(p, (final[ 1]<<16) | (final[ 7]<<8) | final[13], 4);
    p = ngx_crypt_to64(p, (final[ 2]<<16) | (final[ 8]<<8) | final[14], 4);
    p = ngx_crypt_to64(p, (final[ 3]<<16) | (final[ 9]<<8) | final[15], 4);
    p = ngx_crypt_to64(p, (final[ 4]<<16) | (final[10]<<8) | final[ 5], 4);
    p = ngx_crypt_to64(p, final[11], 2);
    *p = '\0';

    return NGX_OK;
}


static u_char *
ngx_crypt_to64(u_char *p, uint32_t v, size_t n)
{
    static u_char   itoa64[] =
        "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

    while (n--) {
       *p++ = itoa64[v & 0x3f];
       v >>= 6;
    }

    return p;
}


static ngx_int_t
ngx_crypt_plain(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
{
    size_t   len;
    u_char  *p;

    len = ngx_strlen(key);

    *encrypted = ngx_pnalloc(pool, sizeof("{PLAIN}") - 1 + len + 1);
    if (*encrypted == NULL) {
        return NGX_ERROR;
    }

    p = ngx_cpymem(*encrypted, "{PLAIN}", sizeof("{PLAIN}") - 1);
    ngx_memcpy(p, key, len + 1);

    return NGX_OK;
}


#if (NGX_HAVE_SHA1)

static ngx_int_t
ngx_crypt_ssha(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
{
    size_t       len;
    ngx_int_t    rc;
    ngx_str_t    encoded, decoded;
```

```
    ngx_sha1_t   sha1;

    /* "{SSHA}" base64(SHA1(key salt) salt) */

    /* decode base64 salt to find out true salt */

    encoded.data = salt + sizeof("{SSHA}") - 1;
    encoded.len = ngx_strlen(encoded.data);

    len = ngx_max(ngx_base64_decoded_length(encoded.len), 20);

    decoded.data = ngx_pnalloc(pool, len);
    if (decoded.data == NULL) {
        return NGX_ERROR;
    }

    rc = ngx_decode_base64(&decoded, &encoded);

    if (rc != NGX_OK || decoded.len < 20) {
        decoded.len = 20;
    }

    /* update SHA1 from key and salt */

    ngx_sha1_init(&sha1);
    ngx_sha1_update(&sha1, key, ngx_strlen(key));
    ngx_sha1_update(&sha1, decoded.data + 20, decoded.len - 20);
    ngx_sha1_final(decoded.data, &sha1);

    /* encode it back to base64 */

    len = sizeof("{SSHA}") - 1 + ngx_base64_encoded_length(decoded.len) + 1;

    *encrypted = ngx_pnalloc(pool, len);
    if (*encrypted == NULL) {
        return NGX_ERROR;
    }

    encoded.data = ngx_cpymem(*encrypted, "{SSHA}", sizeof("{SSHA}") - 1);
    ngx_encode_base64(&encoded, &decoded);
    encoded.data[encoded.len] = '\0';

    return NGX_OK;
}


static ngx_int_t
ngx_crypt_sha(ngx_pool_t *pool, u_char *key, u_char *salt, u_char **encrypted)
{
    size_t      len;
    ngx_str_t   encoded, decoded;
    ngx_sha1_t  sha1;
    u_char      digest[20];

    /* "{SHA}" base64(SHA1(key)) */

    decoded.len = sizeof(digest);
    decoded.data = digest;

    ngx_sha1_init(&sha1);
    ngx_sha1_update(&sha1, key, ngx_strlen(key));
    ngx_sha1_final(digest, &sha1);

    len = sizeof("{SHA}") - 1 + ngx_base64_encoded_length(decoded.len) + 1;

    *encrypted = ngx_pnalloc(pool, len);
    if (*encrypted == NULL) {
        return NGX_ERROR;
    }

    encoded.data = ngx_cpymem(*encrypted, "{SHA}", sizeof("{SHA}") - 1);
    ngx_encode_base64(&encoded, &decoded);
    encoded.data[encoded.len] = '\0';

    return NGX_OK;
}
```

```
280
281 #endif /* NGX_HAVE_SHA1 */
282
283 #endif /* NGX_CRYPT */
```

One Level Up Top Level

src/core/ngx_sha1.h - nginx-1.7.10

Data types defined

- ngx_sha1_t

Macros defined

- _NGX_SHA1_H_INCLUDED_
- ngx_sha1_final
- ngx_sha1_init
- ngx_sha1_update

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_SHA1_H_INCLUDED_
9  #define _NGX_SHA1_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #if (NGX_HAVE_OPENSSL_SHA1_H)
17  #include <openssl/sha.h>
18  #else
19  #include <sha.h>
20  #endif
21
22
23  typedef SHA_CTX  ngx_sha1_t;
24
25
26  #define ngx_sha1_init    SHA1_Init
27  #define ngx_sha1_update  SHA1_Update
28  #define ngx_sha1_final   SHA1_Final
29
30
31  #endif /* _NGX_SHA1_H_INCLUDED_ */
```

src/http/modules/ngx_http_auth_request_module.c - nginx-1.7.10

Global variables defined

- ngx_http_auth_request_commands
- ngx_http_auth_request_module
- ngx_http_auth_request_module_ctx

Data types defined

- ngx_http_auth_request_conf_t
- ngx_http_auth_request_ctx_t
- ngx_http_auth_request_variable_t

Functions defined

- ngx_http_auth_request
- ngx_http_auth_request_create_conf
- ngx_http_auth_request_done
- ngx_http_auth_request_handler
- ngx_http_auth_request_init
- ngx_http_auth_request_merge_conf
- ngx_http_auth_request_set
- ngx_http_auth_request_set_variables
- ngx_http_auth_request_variable

Source code

```
1
2  /*
3  * Copyright (C) Maxim Dounin
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_str_t                 uri;
15      ngx_array_t              *vars;
16  } ngx_http_auth_request_conf_t;
17
18
19  typedef struct {
20      ngx_uint_t                done;
21      ngx_uint_t                status;
22      ngx_http_request_t       *subrequest;
23  } ngx_http_auth_request_ctx_t;
24
```

```
25
26  typedef struct {
27      ngx_int_t                  index;
28      ngx_http_complex_value_t   value;
29      ngx_http_set_variable_pt   set_handler;
30  } ngx_http_auth_request_variable_t;
31
32
33  static ngx_int_t ngx_http_auth_request_handler(ngx_http_request_t *r);
34  static ngx_int_t ngx_http_auth_request_done(ngx_http_request_t *r,
35      void *data, ngx_int_t rc);
36  static ngx_int_t ngx_http_auth_request_set_variables(ngx_http_request_t *r,
37      ngx_http_auth_request_conf_t *arcf, ngx_http_auth_request_ctx_t *ctx);
38  static ngx_int_t ngx_http_auth_request_variable(ngx_http_request_t *r,
39      ngx_http_variable_value_t *v, uintptr_t data);
40  static void *ngx_http_auth_request_create_conf(ngx_conf_t *cf);
41  static char *ngx_http_auth_request_merge_conf(ngx_conf_t *cf,
42      void *parent, void *child);
43  static ngx_int_t ngx_http_auth_request_init(ngx_conf_t *cf);
44  static char *ngx_http_auth_request(ngx_conf_t *cf, ngx_command_t *cmd,
45      void *conf);
46  static char *ngx_http_auth_request_set(ngx_conf_t *cf, ngx_command_t *cmd,
47      void *conf);
48
49
50  static ngx_command_t  ngx_http_auth_request_commands[] = {
51
52      { ngx_string("auth_request"),
53        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
54        ngx_http_auth_request,
55        NGX_HTTP_LOC_CONF_OFFSET,
56        0,
57        NULL },
58
59      { ngx_string("auth_request_set"),
60        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
61        ngx_http_auth_request_set,
62        NGX_HTTP_LOC_CONF_OFFSET,
63        0,
64        NULL },
65
66        ngx_null_command
67  };
68
69
70  static ngx_http_module_t  ngx_http_auth_request_module_ctx = {
71      NULL,                                  /* preconfiguration */
72      ngx_http_auth_request_init,            /* postconfiguration */
73
74      NULL,                                  /* create main configuration */
75      NULL,                                  /* init main configuration */
76
77      NULL,                                  /* create server configuration */
78      NULL,                                  /* merge server configuration */
79
80      ngx_http_auth_request_create_conf,     /* create location configuration */
81      ngx_http_auth_request_merge_conf       /* merge location configuration */
82  };
83
84
85  ngx_module_t  ngx_http_auth_request_module = {
86      NGX_MODULE_V1,
87      &ngx_http_auth_request_module_ctx,     /* module context */
88      ngx_http_auth_request_commands,        /* module directives */
89      NGX_HTTP_MODULE,                       /* module type */
90      NULL,                                  /* init master */
91      NULL,                                  /* init module */
92      NULL,                                  /* init process */
93      NULL,                                  /* init thread */
94      NULL,                                  /* exit thread */
95      NULL,                                  /* exit process */
96      NULL,                                  /* exit master */
97      NGX_MODULE_V1_PADDING
98  };
99
100
```

```
static ngx_int_t
ngx_http_auth_request_handler(ngx_http_request_t *r)
{
    ngx_table_elt_t               *h, *ho;
    ngx_http_request_t            *sr;
    ngx_http_post_subrequest_t    *ps;
    ngx_http_auth_request_ctx_t   *ctx;
    ngx_http_auth_request_conf_t  *arcf;

    arcf = ngx_http_get_module_loc_conf(r, ngx_http_auth_request_module);

    if (arcf->uri.len == 0) {
        return NGX_DECLINED;
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "auth request handler");

    ctx = ngx_http_get_module_ctx(r, ngx_http_auth_request_module);

    if (ctx != NULL) {
        if (!ctx->done) {
            return NGX_AGAIN;
        }

        /*
         * as soon as we are done - explicitly set variables to make
         * sure they will be available after internal redirects
         */

        if (ngx_http_auth_request_set_variables(r, arcf, ctx) != NGX_OK) {
            return NGX_ERROR;
        }

        /* return appropriate status */

        if (ctx->status == NGX_HTTP_FORBIDDEN) {
            return ctx->status;
        }

        if (ctx->status == NGX_HTTP_UNAUTHORIZED) {
            sr = ctx->subrequest;

            h = sr->headers_out.www_authenticate;

            if (!h && sr->upstream) {
                h = sr->upstream->headers_in.www_authenticate;
            }

            if (h) {
                ho = ngx_list_push(&r->headers_out.headers);
                if (ho == NULL) {
                    return NGX_ERROR;
                }

                *ho = *h;

                r->headers_out.www_authenticate = ho;
            }

            return ctx->status;
        }

        if (ctx->status >= NGX_HTTP_OK
            && ctx->status < NGX_HTTP_SPECIAL_RESPONSE)
        {
            return NGX_OK;
        }

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "auth request unexpected status: %d", ctx->status);

        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_auth_request_ctx_t));
```

```
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ps = ngx_palloc(r->pool, sizeof(ngx_http_post_subrequest_t));
    if (ps == NULL) {
        return NGX_ERROR;
    }

    ps->handler = ngx_http_auth_request_done;
    ps->data = ctx;

    if (ngx_http_subrequest(r, &arcf->uri, NULL, &sr, ps,
                            NGX_HTTP_SUBREQUEST_WAITED)
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    /*
     * allocate fake request body to avoid attempts to read it and to make
     * sure real body file (if already read) won't be closed by upstream
     */

    sr->request_body = ngx_pcalloc(r->pool, sizeof(ngx_http_request_body_t));
    if (sr->request_body == NULL) {
        return NGX_ERROR;
    }

    sr->header_only = 1;

    ctx->subrequest = sr;

    ngx_http_set_ctx(r, ctx, ngx_http_auth_request_module);

    return NGX_AGAIN;
}


static ngx_int_t
ngx_http_auth_request_done(ngx_http_request_t *r, void *data, ngx_int_t rc)
{
    ngx_http_auth_request_ctx_t   *ctx = data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "auth request done s:%d", r->headers_out.status);

    ctx->done = 1;
    ctx->status = r->headers_out.status;

    return rc;
}


static ngx_int_t
ngx_http_auth_request_set_variables(ngx_http_request_t *r,
    ngx_http_auth_request_conf_t *arcf, ngx_http_auth_request_ctx_t *ctx)
{
    ngx_str_t                          val;
    ngx_http_variable_t               *v;
    ngx_http_variable_value_t         *vv;
    ngx_http_auth_request_variable_t  *av, *last;
    ngx_http_core_main_conf_t         *cmcf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "auth request set variables");

    if (arcf->vars == NULL) {
        return NGX_OK;
    }

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
    v = cmcf->variables.elts;

    av = arcf->vars->elts;
    last = av + arcf->vars->nelts;
```

```
253
254     while (av < last) {
255         /*
256          * explicitly set new value to make sure it will be available after
257          * internal redirects
258          */
259
260         vv = &r->variables[av->index];
261
262         if (ngx_http_complex_value(ctx->subrequest, &av->value, &val)
263             != NGX_OK)
264         {
265             return NGX_ERROR;
266         }
267
268         vv->valid = 1;
269         vv->not_found = 0;
270         vv->data = val.data;
271         vv->len = val.len;
272
273         if (av->set_handler) {
274             /*
275              * set_handler only available in cmcf->variables_keys, so we store
276              * it explicitly
277              */
278
279             av->set_handler(r, vv, v[av->index].data);
280         }
281
282         av++;
283     }
284
285     return NGX_OK;
286 }
287
288
289 static ngx_int_t
290 ngx_http_auth_request_variable(ngx_http_request_t *r,
291     ngx_http_variable_value_t *v, uintptr_t data)
292 {
293     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
294                    "auth request variable");
295
296     v->not_found = 1;
297
298     return NGX_OK;
299 }
300
301
302 static void *
303 ngx_http_auth_request_create_conf(ngx_conf_t *cf)
304 {
305     ngx_http_auth_request_conf_t  *conf;
306
307     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_auth_request_conf_t));
308     if (conf == NULL) {
309         return NULL;
310     }
311
312     /*
313      * set by ngx_pcalloc():
314      *
315      *     conf->uri = { 0, NULL };
316      */
317
318     conf->vars = NGX_CONF_UNSET_PTR;
319
320     return conf;
321 }
322
323
324 static char *
325 ngx_http_auth_request_merge_conf(ngx_conf_t *cf, void *parent, void *child)
326 {
327     ngx_http_auth_request_conf_t *prev = parent;
328     ngx_http_auth_request_conf_t *conf = child;
```

```
    ngx_conf_merge_str_value(conf->uri, prev->uri, "");
    ngx_conf_merge_ptr_value(conf->vars, prev->vars, NULL);

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_auth_request_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt        *h;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_ACCESS_PHASE].handlers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    *h = ngx_http_auth_request_handler;

    return NGX_OK;
}


static char *
ngx_http_auth_request(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_auth_request_conf_t *arcf = conf;

    ngx_str_t        *value;

    if (arcf->uri.data != NULL) {
        return "is duplicate";
    }

    value = cf->args->elts;

    if (ngx_strcmp(value[1].data, "off") == 0) {
        arcf->uri.len = 0;
        arcf->uri.data = (u_char *) "";

        return NGX_CONF_OK;
    }

    arcf->uri = value[1];

    return NGX_CONF_OK;
}


static char *
ngx_http_auth_request_set(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_auth_request_conf_t *arcf = conf;

    ngx_str_t                          *value;
    ngx_http_variable_t                *v;
    ngx_http_auth_request_variable_t   *av;
    ngx_http_compile_complex_value_t   ccv;

    value = cf->args->elts;

    if (value[1].data[0] != '$') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid variable name \"%V\"", &value[1]);
        return NGX_CONF_ERROR;
    }

    value[1].len--;
    value[1].data++;

    if (arcf->vars == NGX_CONF_UNSET_PTR) {
        arcf->vars = ngx_array_create(cf->pool, 1,
```

```
                                       sizeof(ngx_http_auth_request_variable_t));
        if (arcf->vars == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    av = ngx_array_push(arcf->vars);
    if (av == NULL) {
        return NGX_CONF_ERROR;
    }

    v = ngx_http_add_variable(cf, &value[1], NGX_HTTP_VAR_CHANGEABLE);
    if (v == NULL) {
        return NGX_CONF_ERROR;
    }

    av->index = ngx_http_get_variable_index(cf, &value[1]);
    if (av->index == NGX_ERROR) {
        return NGX_CONF_ERROR;
    }

    if (v->get_handler == NULL) {
        v->get_handler = ngx_http_auth_request_variable;
        v->data = (uintptr_t) av;
    }

    av->set_handler = v->set_handler;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[2];
    ccv.complex_value = &av->value;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}
```

One Level Up Top Level

src/http/modules/ngx_http_autoindex_module.c - nginx-1.7.10

Global variables defined

- ngx_http_autoindex_commands
- ngx_http_autoindex_format
- ngx_http_autoindex_module
- ngx_http_autoindex_module_ctx

Data types defined

- ngx_http_autoindex_entry_t
- ngx_http_autoindex_loc_conf_t

Functions defined

- ngx_http_autoindex_cmp_entries
- ngx_http_autoindex_create_loc_conf
- ngx_http_autoindex_error
- ngx_http_autoindex_handler
- ngx_http_autoindex_html
- ngx_http_autoindex_init
- ngx_http_autoindex_json
- ngx_http_autoindex_jsonp_callback
- ngx_http_autoindex_merge_loc_conf
- ngx_http_autoindex_xml

Macros defined

- NGX_HTTP_AUTOINDEX_HTML
- NGX_HTTP_AUTOINDEX_JSON
- NGX_HTTP_AUTOINDEX_JSONP
- NGX_HTTP_AUTOINDEX_NAME_LEN
- NGX_HTTP_AUTOINDEX_PREALLOCATE
- NGX_HTTP_AUTOINDEX_XML

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
```

```
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_http.h>


 #if 0

 typedef struct {
     ngx_buf_t     *buf;
     size_t         size;
     ngx_pool_t    *pool;
     size_t         alloc_size;
     ngx_chain_t  **last_out;
 } ngx_http_autoindex_ctx_t;

 #endif


 typedef struct {
     ngx_str_t      name;
     size_t         utf_len;
     size_t         escape;
     size_t         escape_html;

     unsigned       dir:1;
     unsigned       file:1;

     time_t         mtime;
     off_t          size;
 } ngx_http_autoindex_entry_t;


 typedef struct {
     ngx_flag_t     enable;
     ngx_uint_t     format;
     ngx_flag_t     localtime;
     ngx_flag_t     exact_size;
 } ngx_http_autoindex_loc_conf_t;


 #define NGX_HTTP_AUTOINDEX_HTML         0
 #define NGX_HTTP_AUTOINDEX_JSON         1
 #define NGX_HTTP_AUTOINDEX_JSONP        2
 #define NGX_HTTP_AUTOINDEX_XML          3

 #define NGX_HTTP_AUTOINDEX_PREALLOCATE  50

 #define NGX_HTTP_AUTOINDEX_NAME_LEN     50


 static ngx_buf_t *ngx_http_autoindex_html(ngx_http_request_t *r,
     ngx_array_t *entries);
 static ngx_buf_t *ngx_http_autoindex_json(ngx_http_request_t *r,
     ngx_array_t *entries, ngx_str_t *callback);
 static ngx_int_t ngx_http_autoindex_jsonp_callback(ngx_http_request_t *r,
     ngx_str_t *callback);
 static ngx_buf_t *ngx_http_autoindex_xml(ngx_http_request_t *r,
     ngx_array_t *entries);

 static int ngx_libc_cdecl ngx_http_autoindex_cmp_entries(const void *one,
     const void *two);
 static ngx_int_t ngx_http_autoindex_error(ngx_http_request_t *r,
     ngx_dir_t *dir, ngx_str_t *name);

 static ngx_int_t ngx_http_autoindex_init(ngx_conf_t *cf);
 static void *ngx_http_autoindex_create_loc_conf(ngx_conf_t *cf);
 static char *ngx_http_autoindex_merge_loc_conf(ngx_conf_t *cf,
     void *parent, void *child);


 static ngx_conf_enum_t  ngx_http_autoindex_format[] = {
     { ngx_string("html"), NGX_HTTP_AUTOINDEX_HTML },
     { ngx_string("json"), NGX_HTTP_AUTOINDEX_JSON },
```

```
    { ngx_string("jsonp"), NGX_HTTP_AUTOINDEX_JSONP },
    { ngx_string("xml"), NGX_HTTP_AUTOINDEX_XML },
    { ngx_null_string, 0 }
};


static ngx_command_t  ngx_http_autoindex_commands[] = {

    { ngx_string("autoindex"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_autoindex_loc_conf_t, enable),
      NULL },

    { ngx_string("autoindex_format"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_autoindex_loc_conf_t, format),
      &ngx_http_autoindex_format },

    { ngx_string("autoindex_localtime"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_autoindex_loc_conf_t, localtime),
      NULL },

    { ngx_string("autoindex_exact_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_autoindex_loc_conf_t, exact_size),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_autoindex_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_autoindex_init,               /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_autoindex_create_loc_conf,    /* create location configuration */
    ngx_http_autoindex_merge_loc_conf      /* merge location configuration */
};


ngx_module_t  ngx_http_autoindex_module = {
    NGX_MODULE_V1,
    &ngx_http_autoindex_module_ctx,        /* module context */
    ngx_http_autoindex_commands,           /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_autoindex_handler(ngx_http_request_t *r)
{
    u_char                     *last, *filename;
    size_t                      len, allocated, root;
```

```
    ngx_err_t                       err;
    ngx_buf_t                      *b;
    ngx_int_t                       rc;
    ngx_str_t                       path, callback;
    ngx_dir_t                       dir;
    ngx_uint_t                      level, format;
    ngx_pool_t                     *pool;
    ngx_chain_t                     out;
    ngx_array_t                     entries;
    ngx_http_autoindex_entry_t     *entry;
    ngx_http_autoindex_loc_conf_t  *alcf;

    if (r->uri.data[r->uri.len - 1] != '/') {
        return NGX_DECLINED;
    }

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD))) {
        return NGX_DECLINED;
    }

    alcf = ngx_http_get_module_loc_conf(r, ngx_http_autoindex_module);

    if (!alcf->enable) {
        return NGX_DECLINED;
    }

    /* NGX_DIR_MASK_LEN is lesser than NGX_HTTP_AUTOINDEX_PREALLOCATE */

    last = ngx_http_map_uri_to_path(r, &path, &root,
                                    NGX_HTTP_AUTOINDEX_PREALLOCATE);
    if (last == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    allocated = path.len;
    path.len = last - path.data;
    if (path.len > 1) {
        path.len--;
    }
    path.data[path.len] = '\0';

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http autoindex: \"%s\"", path.data);

    format = alcf->format;

    if (format == NGX_HTTP_AUTOINDEX_JSONP) {
        if (ngx_http_autoindex_jsonp_callback(r, &callback) != NGX_OK) {
            return NGX_HTTP_BAD_REQUEST;
        }

        if (callback.len == 0) {
            format = NGX_HTTP_AUTOINDEX_JSON;
        }
    }

    if (ngx_open_dir(&path, &dir) == NGX_ERROR) {
        err = ngx_errno;

        if (err == NGX_ENOENT
            || err == NGX_ENOTDIR
            || err == NGX_ENAMETOOLONG)
        {
            level = NGX_LOG_ERR;
            rc = NGX_HTTP_NOT_FOUND;

        } else if (err == NGX_EACCES) {
            level = NGX_LOG_ERR;
            rc = NGX_HTTP_FORBIDDEN;

        } else {
            level = NGX_LOG_CRIT;
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        ngx_log_error(level, r->connection->log, err,
```

```
                      ngx_open_dir_n " \"%s\" failed", path.data);

        return rc;
    }

#if (NGX_SUPPRESS_WARN)

    /* MSVC thinks 'entries' may be used without having been initialized */
    ngx_memzero(&entries, sizeof(ngx_array_t));

#endif

    /* TODO: pool should be temporary pool */
    pool = r->pool;

    if (ngx_array_init(&entries, pool, 40, sizeof(ngx_http_autoindex_entry_t))
        != NGX_OK)
    {
        return ngx_http_autoindex_error(r, &dir, &path);
    }

    r->headers_out.status = NGX_HTTP_OK;

    switch (format) {

    case NGX_HTTP_AUTOINDEX_JSON:
        ngx_str_set(&r->headers_out.content_type, "application/json");
        break;

    case NGX_HTTP_AUTOINDEX_JSONP:
        ngx_str_set(&r->headers_out.content_type, "application/javascript");
        break;

    case NGX_HTTP_AUTOINDEX_XML:
        ngx_str_set(&r->headers_out.content_type, "text/xml");
        ngx_str_set(&r->headers_out.charset, "utf-8");
        break;

    default: /* NGX_HTTP_AUTOINDEX_HTML */
        ngx_str_set(&r->headers_out.content_type, "text/html");
        break;
    }

    r->headers_out.content_type_len = r->headers_out.content_type.len;
    r->headers_out.content_type_lowcase = NULL;

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
        if (ngx_close_dir(&dir) == NGX_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, r->connection->log, ngx_errno,
                          ngx_close_dir_n " \"%V\" failed", &path);
        }

        return rc;
    }

    filename = path.data;
    filename[path.len] = '/';

    for ( ;; ) {
        ngx_set_errno(0);

        if (ngx_read_dir(&dir) == NGX_ERROR) {
            err = ngx_errno;

            if (err != NGX_ENOMOREFILES) {
                ngx_log_error(NGX_LOG_CRIT, r->connection->log, err,
                              ngx_read_dir_n " \"%V\" failed", &path);
                return ngx_http_autoindex_error(r, &dir, &path);
            }

            break;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
```

```
                    "http autoindex file: \"%s\"", ngx_de_name(&dir));

        len = ngx_de_namelen(&dir);

        if (ngx_de_name(&dir)[0] == '.') {
            continue;
        }

        if (!dir.valid_info) {

            /* 1 byte for '/' and 1 byte for terminating '\0' */

            if (path.len + 1 + len + 1 > allocated) {
                allocated = path.len + 1 + len + 1
                                     + NGX_HTTP_AUTOINDEX_PREALLOCATE;

                filename = ngx_pnalloc(pool, allocated);
                if (filename == NULL) {
                    return ngx_http_autoindex_error(r, &dir, &path);
                }

                last = ngx_cpystrn(filename, path.data, path.len + 1);
                *last++ = '/';
            }

            ngx_cpystrn(last, ngx_de_name(&dir), len + 1);

            if (ngx_de_info(filename, &dir) == NGX_FILE_ERROR) {
                err = ngx_errno;

                if (err != NGX_ENOENT && err != NGX_ELOOP) {
                    ngx_log_error(NGX_LOG_CRIT, r->connection->log, err,
                                  ngx_de_info_n " \"%s\" failed", filename);

                    if (err == NGX_EACCES) {
                        continue;
                    }

                    return ngx_http_autoindex_error(r, &dir, &path);
                }

                if (ngx_de_link_info(filename, &dir) == NGX_FILE_ERROR) {
                    ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
                                  ngx_de_link_info_n " \"%s\" failed",
                                  filename);
                    return ngx_http_autoindex_error(r, &dir, &path);
                }
            }
        }

        entry = ngx_array_push(&entries);
        if (entry == NULL) {
            return ngx_http_autoindex_error(r, &dir, &path);
        }

        entry->name.len = len;

        entry->name.data = ngx_pnalloc(pool, len + 1);
        if (entry->name.data == NULL) {
            return ngx_http_autoindex_error(r, &dir, &path);
        }

        ngx_cpystrn(entry->name.data, ngx_de_name(&dir), len + 1);

        entry->dir = ngx_de_is_dir(&dir);
        entry->file = ngx_de_is_file(&dir);
        entry->mtime = ngx_de_mtime(&dir);
        entry->size = ngx_de_size(&dir);
    }

    if (ngx_close_dir(&dir) == NGX_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, ngx_errno,
                      ngx_close_dir_n " \"%V\" failed", &path);
    }

    if (entries.nelts > 1) {
```

```
        ngx_qsort(entries.elts, (size_t) entries.nelts,
                  sizeof(ngx_http_autoindex_entry_t),
                  ngx_http_autoindex_cmp_entries);
    }

    switch (format) {

    case NGX_HTTP_AUTOINDEX_JSON:
        b = ngx_http_autoindex_json(r, &entries, NULL);
        break;

    case NGX_HTTP_AUTOINDEX_JSONP:
        b = ngx_http_autoindex_json(r, &entries, &callback);
        break;

    case NGX_HTTP_AUTOINDEX_XML:
        b = ngx_http_autoindex_xml(r, &entries);
        break;

    default: /* NGX_HTTP_AUTOINDEX_HTML */
        b = ngx_http_autoindex_html(r, &entries);
        break;
    }

    if (b == NULL) {
        return NGX_ERROR;
    }

    /* TODO: free temporary pool */

    if (r == r->main) {
        b->last_buf = 1;
    }

    b->last_in_chain = 1;

    out.buf = b;
    out.next = NULL;

    return ngx_http_output_filter(r, &out);
}


static ngx_buf_t *
ngx_http_autoindex_html(ngx_http_request_t *r, ngx_array_t *entries)
{
    u_char                          *last, scale;
    off_t                            length;
    size_t                           len, char_len, escape_html;
    ngx_tm_t                         tm;
    ngx_buf_t                       *b;
    ngx_int_t                        size;
    ngx_uint_t                       i, utf8;
    ngx_time_t                      *tp;
    ngx_http_autoindex_entry_t      *entry;
    ngx_http_autoindex_loc_conf_t   *alcf;

    static u_char  title[] =
        "<html>" CRLF
        "<head><title>Index of "
    ;

    static u_char  header[] =
        "</title></head>" CRLF
        "<body bgcolor=\"white\">" CRLF
        "<h1>Index of "
    ;

    static u_char  tail[] =
        "</body>" CRLF
        "</html>" CRLF
    ;

    static char  *months[] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                               "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

```

```
    if (r->headers_out.charset.len == 5
        && ngx_strncasecmp(r->headers_out.charset.data, (u_char *) "utf-8", 5)
           == 0)
    {
        utf8 = 1;

    } else {
        utf8 = 0;
    }

    escape_html = ngx_escape_html(NULL, r->uri.data, r->uri.len);

    len = sizeof(title) - 1
          + r->uri.len + escape_html
          + sizeof(header) - 1
          + r->uri.len + escape_html
          + sizeof("</h1>") - 1
          + sizeof("<hr><pre><a href=\"../\">../</a>" CRLF) - 1
          + sizeof("</pre><hr>") - 1
          + sizeof(tail) - 1;

    entry = entries->elts;
    for (i = 0; i < entries->nelts; i++) {
        entry[i].escape = 2 * ngx_escape_uri(NULL, entry[i].name.data,
                                             entry[i].name.len,
                                             NGX_ESCAPE_URI_COMPONENT);

        entry[i].escape_html = ngx_escape_html(NULL, entry[i].name.data,
                                               entry[i].name.len);

        if (utf8) {
            entry[i].utf_len = ngx_utf8_length(entry[i].name.data,
                                               entry[i].name.len);
        } else {
            entry[i].utf_len = entry[i].name.len;
        }

        len += sizeof("<a href=\"\"") - 1
            + entry[i].name.len + entry[i].escape
            + 1                                             /* 1 is for "/" */
            + sizeof("\">") - 1
            + entry[i].name.len - entry[i].utf_len
            + entry[i].escape_html
            + NGX_HTTP_AUTOINDEX_NAME_LEN + sizeof("&gt;") - 2
            + sizeof("</a>") - 1
            + sizeof(" 28-Sep-1970 12:00 ") - 1
            + 20                                            /* the file size */
            + 2;
    }

    b = ngx_create_temp_buf(r->pool, len);
    if (b == NULL) {
        return NULL;
    }

    b->last = ngx_cpymem(b->last, title, sizeof(title) - 1);

    if (escape_html) {
        b->last = (u_char *) ngx_escape_html(b->last, r->uri.data, r->uri.len);
        b->last = ngx_cpymem(b->last, header, sizeof(header) - 1);
        b->last = (u_char *) ngx_escape_html(b->last, r->uri.data, r->uri.len);

    } else {
        b->last = ngx_cpymem(b->last, r->uri.data, r->uri.len);
        b->last = ngx_cpymem(b->last, header, sizeof(header) - 1);
        b->last = ngx_cpymem(b->last, r->uri.data, r->uri.len);
    }

    b->last = ngx_cpymem(b->last, "</h1>", sizeof("</h1>") - 1);

    b->last = ngx_cpymem(b->last, "<hr><pre><a href=\"../\">../</a>" CRLF,
                         sizeof("<hr><pre><a href=\"../\">../</a>" CRLF) - 1);

    alcf = ngx_http_get_module_loc_conf(r, ngx_http_autoindex_module);
    tp = ngx_timeofday();

```

```
    for (i = 0; i < entries->nelts; i++) {
        b->last = ngx_cpymem(b->last, "<a href=\"", sizeof("<a href=\"") - 1);

        if (entry[i].escape) {
            ngx_escape_uri(b->last, entry[i].name.data, entry[i].name.len,
                           NGX_ESCAPE_URI_COMPONENT);

            b->last += entry[i].name.len + entry[i].escape;

        } else {
            b->last = ngx_cpymem(b->last, entry[i].name.data,
                                 entry[i].name.len);
        }

        if (entry[i].dir) {
            *b->last++ = '/';
        }

        *b->last++ = '"';
        *b->last++ = '>';

        len = entry[i].utf_len;

        if (entry[i].name.len != len) {
            if (len > NGX_HTTP_AUTOINDEX_NAME_LEN) {
                char_len = NGX_HTTP_AUTOINDEX_NAME_LEN - 3 + 1;

            } else {
                char_len = NGX_HTTP_AUTOINDEX_NAME_LEN + 1;
            }

            last = b->last;
            b->last = ngx_utf8_cpystrn(b->last, entry[i].name.data,
                                       char_len, entry[i].name.len + 1);

            if (entry[i].escape_html) {
                b->last = (u_char *) ngx_escape_html(last, entry[i].name.data,
                                                     b->last - last);
            }

            last = b->last;

        } else {
            if (entry[i].escape_html) {
                if (len > NGX_HTTP_AUTOINDEX_NAME_LEN) {
                    char_len = NGX_HTTP_AUTOINDEX_NAME_LEN - 3;

                } else {
                    char_len = len;
                }

                b->last = (u_char *) ngx_escape_html(b->last,
                                                     entry[i].name.data, char_len);
                last = b->last;

            } else {
                b->last = ngx_cpystrn(b->last, entry[i].name.data,
                                      NGX_HTTP_AUTOINDEX_NAME_LEN + 1);
                last = b->last - 3;
            }
        }

        if (len > NGX_HTTP_AUTOINDEX_NAME_LEN) {
            b->last = ngx_cpymem(last, "..&gt;</a>", sizeof("..&gt;</a>") - 1);

        } else {
            if (entry[i].dir && NGX_HTTP_AUTOINDEX_NAME_LEN - len > 0) {
                *b->last++ = '/';
                len++;
            }

            b->last = ngx_cpymem(b->last, "</a>", sizeof("</a>") - 1);

            if (NGX_HTTP_AUTOINDEX_NAME_LEN - len > 0) {
                ngx_memset(b->last, ' ', NGX_HTTP_AUTOINDEX_NAME_LEN - len);
                b->last += NGX_HTTP_AUTOINDEX_NAME_LEN - len;
```

```
            }
        }

        *b->last++ = ' ';

        ngx_gmtime(entry[i].mtime + tp->gmtoff * 60 * alcf->localtime, &tm);

        b->last = ngx_sprintf(b->last, "%02d-%s-%d %02d:%02d ",
                              tm.ngx_tm_mday,
                              months[tm.ngx_tm_mon - 1],
                              tm.ngx_tm_year,
                              tm.ngx_tm_hour,
                              tm.ngx_tm_min);

        if (alcf->exact_size) {
            if (entry[i].dir) {
                b->last = ngx_cpymem(b->last,  "                  -",
                                     sizeof("                  -") - 1);
            } else {
                b->last = ngx_sprintf(b->last, "%19O", entry[i].size);
            }

        } else {
            if (entry[i].dir) {
                b->last = ngx_cpymem(b->last,  "      -",
                                     sizeof("      -") - 1);

            } else {
                length = entry[i].size;

                if (length > 1024 * 1024 * 1024 - 1) {
                    size = (ngx_int_t) (length / (1024 * 1024 * 1024));
                    if ((length % (1024 * 1024 * 1024))
                                                > (1024 * 1024 * 1024 / 2 - 1))
                    {
                        size++;
                    }
                    scale = 'G';

                } else if (length > 1024 * 1024 - 1) {
                    size = (ngx_int_t) (length / (1024 * 1024));
                    if ((length % (1024 * 1024)) > (1024 * 1024 / 2 - 1)) {
                        size++;
                    }
                    scale = 'M';

                } else if (length > 9999) {
                    size = (ngx_int_t) (length / 1024);
                    if (length % 1024 > 511) {
                        size++;
                    }
                    scale = 'K';

                } else {
                    size = (ngx_int_t) length;
                    scale = '\0';
                }

                if (scale) {
                    b->last = ngx_sprintf(b->last, "%6i%c", size, scale);

                } else {
                    b->last = ngx_sprintf(b->last, " %6i", size);
                }
            }
        }

        *b->last++ = CR;
        *b->last++ = LF;
    }

    b->last = ngx_cpymem(b->last, "</pre><hr>", sizeof("</pre><hr>") - 1);

    b->last = ngx_cpymem(b->last, tail, sizeof(tail) - 1);

    return b;
```

```
}


static ngx_buf_t *
ngx_http_autoindex_json(ngx_http_request_t *r, ngx_array_t *entries,
    ngx_str_t *callback)
{
    size_t                      len;
    ngx_buf_t                  *b;
    ngx_uint_t                  i;
    ngx_http_autoindex_entry_t  *entry;

    len = sizeof("[" CRLF CRLF "]") - 1;

    if (callback) {
        len += sizeof("/* callback */" CRLF "();") - 1 + callback->len;
    }

    entry = entries->elts;

    for (i = 0; i < entries->nelts; i++) {
        entry[i].escape = ngx_escape_json(NULL, entry[i].name.data,
                                          entry[i].name.len);

        len += sizeof("{  }," CRLF) - 1
            + sizeof("\"name\":\"\"") - 1
            + entry[i].name.len + entry[i].escape
            + sizeof(", \"type\":\"directory\"") - 1
            + sizeof(", \"mtime\":\"Wed, 31 Dec 1986 10:00:00 GMT\"") - 1;

        if (entry[i].file) {
            len += sizeof(", \"size\":") - 1 + NGX_OFF_T_LEN;
        }
    }

    b = ngx_create_temp_buf(r->pool, len);
    if (b == NULL) {
        return NULL;
    }

    if (callback) {
        b->last = ngx_cpymem(b->last, "/* callback */" CRLF,
                             sizeof("/* callback */" CRLF) - 1);

        b->last = ngx_cpymem(b->last, callback->data, callback->len);

        *b->last++ = '(';
    }

    *b->last++ = '[';

    for (i = 0; i < entries->nelts; i++) {
        b->last = ngx_cpymem(b->last, CRLF "{ \"name\":\"",
                             sizeof(CRLF "{ \"name\":\"") - 1);

        if (entry[i].escape) {
            b->last = (u_char *) ngx_escape_json(b->last, entry[i].name.data,
                                                 entry[i].name.len);
        } else {
            b->last = ngx_cpymem(b->last, entry[i].name.data,
                                 entry[i].name.len);
        }

        b->last = ngx_cpymem(b->last, "\", \"type\":\"",
                             sizeof("\", \"type\":\"") - 1);

        if (entry[i].dir) {
            b->last = ngx_cpymem(b->last, "directory", sizeof("directory") - 1);

        } else if (entry[i].file) {
            b->last = ngx_cpymem(b->last, "file", sizeof("file") - 1);

        } else {
            b->last = ngx_cpymem(b->last, "other", sizeof("other") - 1);
        }

```

```
765         b->last = ngx_cpymem(b->last, "\", \"mtime\":\"",
766                              sizeof("\", \"mtime\":\"") - 1);
767 
768         b->last = ngx_http_time(b->last, entry[i].mtime);
769 
770         if (entry[i].file) {
771             b->last = ngx_cpymem(b->last, "\", \"size\":",
772                                  sizeof("\", \"size\":") - 1);
773             b->last = ngx_sprintf(b->last, "%O", entry[i].size);
774 
775         } else {
776             *b->last++ = '"';
777         }
778 
779         b->last = ngx_cpymem(b->last, " },", sizeof(" },") - 1);
780     }
781 
782     if (i > 0) {
783         b->last--;  /* strip last comma */
784     }
785 
786     b->last = ngx_cpymem(b->last, CRLF "]", sizeof(CRLF "]") - 1);
787 
788     if (callback) {
789         *b->last++ = ')'; *b->last++ = ';';
790     }
791 
792     return b;
793 }
794 
795 
796 static ngx_int_t
797 ngx_http_autoindex_jsonp_callback(ngx_http_request_t *r, ngx_str_t *callback)
798 {
799     u_char      *p, c, ch;
800     ngx_uint_t   i;
801 
802     if (ngx_http_arg(r, (u_char *) "callback", 8, callback) != NGX_OK) {
803         callback->len = 0;
804         return NGX_OK;
805     }
806 
807     if (callback->len > 128) {
808         ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
809                       "client sent too long callback name: \"%V\"", callback);
810         return NGX_DECLINED;
811     }
812 
813     p = callback->data;
814 
815     for (i = 0; i < callback->len; i++) {
816         ch = p[i];
817 
818         c = (u_char) (ch | 0x20);
819         if (c >= 'a' && c <= 'z') {
820             continue;
821         }
822 
823         if ((ch >= '0' && ch <= '9') || ch == '_' || ch == '.') {
824             continue;
825         }
826 
827         ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
828                       "client sent invalid callback name: \"%V\"", callback);
829 
830         return NGX_DECLINED;
831     }
832 
833     return NGX_OK;
834 }
835 
836 
837 static ngx_buf_t *
838 ngx_http_autoindex_xml(ngx_http_request_t *r, ngx_array_t *entries)
839 {
840     size_t                          len;
```

```
    ngx_tm_t                        tm;
    ngx_buf_t                      *b;
    ngx_str_t                       type;
    ngx_uint_t                      i;
    ngx_http_autoindex_entry_t     *entry;

    static u_char  head[] = "<?xml version=\"1.0\"?>" CRLF "<list>" CRLF;
    static u_char  tail[] = "</list>" CRLF;

    len = sizeof(head) - 1 + sizeof(tail) - 1;

    entry = entries->elts;

    for (i = 0; i < entries->nelts; i++) {
        entry[i].escape = ngx_escape_html(NULL, entry[i].name.data,
                                          entry[i].name.len);

        len += sizeof("<directory></directory>" CRLF) - 1
            + entry[i].name.len + entry[i].escape
            + sizeof(" mtime=\"1986-12-31T10:00:00Z\"") - 1;

        if (entry[i].file) {
            len += sizeof(" size=\"\"") - 1 + NGX_OFF_T_LEN;
        }
    }

    b = ngx_create_temp_buf(r->pool, len);
    if (b == NULL) {
        return NULL;
    }

    b->last = ngx_cpymem(b->last, head, sizeof(head) - 1);

    for (i = 0; i < entries->nelts; i++) {
        *b->last++ = '<';

        if (entry[i].dir) {
            ngx_str_set(&type, "directory");

        } else if (entry[i].file) {
            ngx_str_set(&type, "file");

        } else {
            ngx_str_set(&type, "other");
        }

        b->last = ngx_cpymem(b->last, type.data, type.len);

        b->last = ngx_cpymem(b->last, " mtime=\"", sizeof(" mtime=\"") - 1);

        ngx_gmtime(entry[i].mtime, &tm);

        b->last = ngx_sprintf(b->last, "%4d-%02d-%02dT%02d:%02d:%02dZ",
                              tm.ngx_tm_year, tm.ngx_tm_mon,
                              tm.ngx_tm_mday, tm.ngx_tm_hour,
                              tm.ngx_tm_min, tm.ngx_tm_sec);

        if (entry[i].file) {
            b->last = ngx_cpymem(b->last, "\" size=\"",
                                 sizeof("\" size=\"") - 1);
            b->last = ngx_sprintf(b->last, "%O", entry[i].size);
        }

        *b->last++ = '"'; *b->last++ = '>';

        if (entry[i].escape) {
            b->last = (u_char *) ngx_escape_html(b->last, entry[i].name.data,
                                                 entry[i].name.len);
        } else {
            b->last = ngx_cpymem(b->last, entry[i].name.data,
                                 entry[i].name.len);
        }

        *b->last++ = '<'; *b->last++ = '/';

        b->last = ngx_cpymem(b->last, type.data, type.len);
```

```
        *b->last++ = '>';

        *b->last++ = CR; *b->last++ = LF;
    }

    b->last = ngx_cpymem(b->last, tail, sizeof(tail) - 1);

    return b;
}


static int ngx_libc_cdecl
ngx_http_autoindex_cmp_entries(const void *one, const void *two)
{
    ngx_http_autoindex_entry_t *first = (ngx_http_autoindex_entry_t *) one;
    ngx_http_autoindex_entry_t *second = (ngx_http_autoindex_entry_t *) two;

    if (first->dir && !second->dir) {
        /* move the directories to the start */
        return -1;
    }

    if (!first->dir && second->dir) {
        /* move the directories to the start */
        return 1;
    }

    return (int) ngx_strcmp(first->name.data, second->name.data);
}


#if 0

static ngx_buf_t *
ngx_http_autoindex_alloc(ngx_http_autoindex_ctx_t *ctx, size_t size)
{
    ngx_chain_t  *cl;

    if (ctx->buf) {

        if ((size_t) (ctx->buf->end - ctx->buf->last) >= size) {
            return ctx->buf;
        }

        ctx->size += ctx->buf->last - ctx->buf->pos;
    }

    ctx->buf = ngx_create_temp_buf(ctx->pool, ctx->alloc_size);
    if (ctx->buf == NULL) {
        return NULL;
    }

    cl = ngx_alloc_chain_link(ctx->pool);
    if (cl == NULL) {
        return NULL;
    }

    cl->buf = ctx->buf;
    cl->next = NULL;

    *ctx->last_out = cl;
    ctx->last_out = &cl->next;

    return ctx->buf;
}

#endif


static ngx_int_t
ngx_http_autoindex_error(ngx_http_request_t *r, ngx_dir_t *dir, ngx_str_t *name)
{
    if (ngx_close_dir(dir) == NGX_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, ngx_errno,
                      ngx_close_dir_n " \"%V\" failed", name);
```

```
993     }
994
995     return r->header_sent ? NGX_ERROR : NGX_HTTP_INTERNAL_SERVER_ERROR;
996 }
997
998
999 static void *
1000 ngx_http_autoindex_create_loc_conf(ngx_conf_t *cf)
1001 {
1002     ngx_http_autoindex_loc_conf_t  *conf;
1003
1004     conf = ngx_palloc(cf->pool, sizeof(ngx_http_autoindex_loc_conf_t));
1005     if (conf == NULL) {
1006         return NULL;
1007     }
1008
1009     conf->enable = NGX_CONF_UNSET;
1010     conf->format = NGX_CONF_UNSET_UINT;
1011     conf->localtime = NGX_CONF_UNSET;
1012     conf->exact_size = NGX_CONF_UNSET;
1013
1014     return conf;
1015 }
1016
1017
1018 static char *
1019 ngx_http_autoindex_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
1020 {
1021     ngx_http_autoindex_loc_conf_t *prev = parent;
1022     ngx_http_autoindex_loc_conf_t *conf = child;
1023
1024     ngx_conf_merge_value(conf->enable, prev->enable, 0);
1025     ngx_conf_merge_uint_value(conf->format, prev->format,
1026                               NGX_HTTP_AUTOINDEX_HTML);
1027     ngx_conf_merge_value(conf->localtime, prev->localtime, 0);
1028     ngx_conf_merge_value(conf->exact_size, prev->exact_size, 1);
1029
1030     return NGX_CONF_OK;
1031 }
1032
1033
1034 static ngx_int_t
1035 ngx_http_autoindex_init(ngx_conf_t *cf)
1036 {
1037     ngx_http_handler_pt        *h;
1038     ngx_http_core_main_conf_t  *cmcf;
1039
1040     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
1041
1042     h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
1043     if (h == NULL) {
1044         return NGX_ERROR;
1045     }
1046
1047     *h = ngx_http_autoindex_handler;
1048
1049     return NGX_OK;
1050 }
```

One Level Up Top Level

src/http/modules/ngx_http_browser_module.c - nginx-1.7.10

Global variables defined

- ngx_http_browser_commands
- ngx_http_browser_module
- ngx_http_browser_module_ctx
- ngx_http_browsers
- ngx_http_modern_browser_masks

Data types defined

- ngx_http_browser_conf_t
- ngx_http_browser_variable_t
- ngx_http_modern_browser_mask_t
- ngx_http_modern_browser_t

Functions defined

- ngx_http_ancient_browser
- ngx_http_ancient_browser_value
- ngx_http_browser
- ngx_http_browser_add_variable
- ngx_http_browser_create_conf
- ngx_http_browser_merge_conf
- ngx_http_browser_variable
- ngx_http_modern_browser
- ngx_http_modern_browser_sort
- ngx_http_modern_browser_value
- ngx_http_msie_variable

Macros defined

- NGX_HTTP_ANCIENT_BROWSER
- NGX_HTTP_MODERN_BROWSER

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
```

```
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_http.h>


 /*
 * The module can check browser versions conforming to the following formats:
 * X, X.X, X.X.X, and X.X.X.X.  The maximum values of each format may be
 * 4000, 4000.99, 4000.99.99, and 4000.99.99.99.
 */


 #define  NGX_HTTP_MODERN_BROWSER   0
 #define  NGX_HTTP_ANCIENT_BROWSER  1


 typedef struct {
     u_char                      browser[12];
     size_t                      skip;
     size_t                      add;
     u_char                      name[12];
 } ngx_http_modern_browser_mask_t;


 typedef struct {
     ngx_uint_t                  version;
     size_t                      skip;
     size_t                      add;
     u_char                      name[12];
 } ngx_http_modern_browser_t;


 typedef struct {
     ngx_str_t                   name;
     ngx_http_get_variable_pt    handler;
     uintptr_t                   data;
 } ngx_http_browser_variable_t;


 typedef struct {
     ngx_array_t                *modern_browsers;
     ngx_array_t                *ancient_browsers;
     ngx_http_variable_value_t  *modern_browser_value;
     ngx_http_variable_value_t  *ancient_browser_value;

     unsigned                    modern_unlisted_browsers:1;
     unsigned                    netscape4:1;
 } ngx_http_browser_conf_t;


 static ngx_int_t ngx_http_msie_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);
 static ngx_int_t ngx_http_browser_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data);

 static ngx_uint_t ngx_http_browser(ngx_http_request_t *r,
     ngx_http_browser_conf_t *cf);

 static ngx_int_t ngx_http_browser_add_variable(ngx_conf_t *cf);
 static void *ngx_http_browser_create_conf(ngx_conf_t *cf);
 static char *ngx_http_browser_merge_conf(ngx_conf_t *cf, void *parent,
     void *child);
 static int ngx_libc_cdecl ngx_http_modern_browser_sort(const void *one,
     const void *two);
 static char *ngx_http_modern_browser(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static char *ngx_http_ancient_browser(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static char *ngx_http_modern_browser_value(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static char *ngx_http_ancient_browser_value(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);

```

```
81
82  static ngx_command_t  ngx_http_browser_commands[] = {
83
84      { ngx_string("modern_browser"),
85        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
86        ngx_http_modern_browser,
87        NGX_HTTP_LOC_CONF_OFFSET,
88        0,
89        NULL },
90
91      { ngx_string("ancient_browser"),
92        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
93        ngx_http_ancient_browser,
94        NGX_HTTP_LOC_CONF_OFFSET,
95        0,
96        NULL },
97
98      { ngx_string("modern_browser_value"),
99        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
100        ngx_http_modern_browser_value,
101       NGX_HTTP_LOC_CONF_OFFSET,
102       0,
103       NULL },
104
105     { ngx_string("ancient_browser_value"),
106       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
107       ngx_http_ancient_browser_value,
108       NGX_HTTP_LOC_CONF_OFFSET,
109       0,
110       NULL },
111
112       ngx_null_command
113 };
114
115
116 static ngx_http_module_t  ngx_http_browser_module_ctx = {
117     ngx_http_browser_add_variable,         /* preconfiguration */
118     NULL,                                  /* postconfiguration */
119
120     NULL,                                  /* create main configuration */
121     NULL,                                  /* init main configuration */
122
123     NULL,                                  /* create server configuration */
124     NULL,                                  /* merge server configuration */
125
126     ngx_http_browser_create_conf,          /* create location configuration */
127     ngx_http_browser_merge_conf            /* merge location configuration */
128 };
129
130
131 ngx_module_t  ngx_http_browser_module = {
132     NGX_MODULE_V1,
133     &ngx_http_browser_module_ctx,          /* module context */
134     ngx_http_browser_commands,             /* module directives */
135     NGX_HTTP_MODULE,                       /* module type */
136     NULL,                                  /* init master */
137     NULL,                                  /* init module */
138     NULL,                                  /* init process */
139     NULL,                                  /* init thread */
140     NULL,                                  /* exit thread */
141     NULL,                                  /* exit process */
142     NULL,                                  /* exit master */
143     NGX_MODULE_V1_PADDING
144 };
145
146
147 static ngx_http_modern_browser_mask_t  ngx_http_modern_browser_masks[] = {
148
149     /* Opera must be the first browser to check */
150
151     /*
152      * "Opera/7.50 (X11; FreeBSD i386; U)  [en]"
153      * "Mozilla/5.0 (X11; FreeBSD i386; U) Opera 7.50  [en]"
154      * "Mozilla/4.0 (compatible; MSIE 6.0; X11; FreeBSD i386) Opera 7.50  [en]"
155      * "Opera/8.0 (Windows NT 5.1; U; ru)"
156      * "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; en) Opera 8.0"
```

```
     * "Opera/9.01 (X11; FreeBSD 6 i386; U; en)"
     */

    { "opera",
      0,
      sizeof("Opera ") - 1,
      "Opera"},

    /* "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" */

    { "msie",
      sizeof("Mozilla/4.0 (compatible; ") - 1,
      sizeof("MSIE ") - 1,
      "MSIE "},

    /*
     * "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0) Gecko/20020610"
     * "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru-RU; rv:1.5) Gecko/20031006"
     * "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru-RU; rv:1.6) Gecko/20040206
     *              Firefox/0.8"
     * "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru-RU; rv:1.7.8)
     *              Gecko/20050511 Firefox/1.0.4"
     * "Mozilla/5.0 (X11; U; FreeBSD i386; en-US; rv:1.8.0.5) Gecko/20060729
     *              Firefox/1.5.0.5"
     */

    { "gecko",
      sizeof("Mozilla/5.0 (") - 1,
      sizeof("rv:") - 1,
      "rv:"},

    /*
     * "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; ru-ru) AppleWebKit/125.2
     *              (KHTML, like Gecko) Safari/125.7"
     * "Mozilla/5.0 (SymbianOS/9.1; U; en-us) AppleWebKit/413
     *              (KHTML, like Gecko) Safari/413"
     * "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/418
     *              (KHTML, like Gecko) Safari/417.9.3"
     * "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; ru-ru) AppleWebKit/418.8
     *              (KHTML, like Gecko) Safari/419.3"
     */

    { "safari",
      sizeof("Mozilla/5.0 (") - 1,
      sizeof("Safari/") - 1,
      "Safari/"},

    /*
     * "Mozilla/5.0 (compatible; Konqueror/3.1; Linux)"
     * "Mozilla/5.0 (compatible; Konqueror/3.4; Linux) KHTML/3.4.2 (like Gecko)"
     * "Mozilla/5.0 (compatible; Konqueror/3.5; FreeBSD) KHTML/3.5.1
     *              (like Gecko)"
     */

    { "konqueror",
      sizeof("Mozilla/5.0 (compatible; ") - 1,
      sizeof("Konqueror/") - 1,
      "Konqueror/"},

    { "", 0, 0, "" }

};


static ngx_http_browser_variable_t  ngx_http_browsers[] = {
    { ngx_string("msie"), ngx_http_msie_variable, 0 },
    { ngx_string("modern_browser"), ngx_http_browser_variable,
          NGX_HTTP_MODERN_BROWSER },
    { ngx_string("ancient_browser"), ngx_http_browser_variable,
          NGX_HTTP_ANCIENT_BROWSER },
    { ngx_null_string, NULL, 0 }
};


static ngx_int_t
ngx_http_browser_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
```

```
    uintptr_t data)
{
    ngx_uint_t                 rc;
    ngx_http_browser_conf_t  *cf;

    cf = ngx_http_get_module_loc_conf(r, ngx_http_browser_module);

    rc = ngx_http_browser(r, cf);

    if (data == NGX_HTTP_MODERN_BROWSER && rc == NGX_HTTP_MODERN_BROWSER) {
        *v = *cf->modern_browser_value;
        return NGX_OK;
    }

    if (data == NGX_HTTP_ANCIENT_BROWSER && rc == NGX_HTTP_ANCIENT_BROWSER) {
        *v = *cf->ancient_browser_value;
        return NGX_OK;
    }

    *v = ngx_http_variable_null_value;
    return NGX_OK;
}


static ngx_uint_t
ngx_http_browser(ngx_http_request_t *r, ngx_http_browser_conf_t *cf)
{
    size_t                      len;
    u_char                     *name, *ua, *last, c;
    ngx_str_t                  *ancient;
    ngx_uint_t                  i, version, ver, scale;
    ngx_http_modern_browser_t  *modern;

    if (r->headers_in.user_agent == NULL) {
        if (cf->modern_unlisted_browsers) {
            return NGX_HTTP_MODERN_BROWSER;
        }

        return NGX_HTTP_ANCIENT_BROWSER;
    }

    ua = r->headers_in.user_agent->value.data;
    len = r->headers_in.user_agent->value.len;
    last = ua + len;

    if (cf->modern_browsers) {
        modern = cf->modern_browsers->elts;

        for (i = 0; i < cf->modern_browsers->nelts; i++) {
            name = ua + modern[i].skip;

            if (name >= last) {
                continue;
            }

            name = (u_char *) ngx_strstr(name, modern[i].name);

            if (name == NULL) {
                continue;
            }

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "browser: \"%s\"", name);

            name += modern[i].add;

            if (name >= last) {
                continue;
            }

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "version: \"%ui\" \"%s\"", modern[i].version, name);

            version = 0;
            ver = 0;
            scale = 1000000;
```

```
309
310             while (name < last) {
311
312                 c = *name++;
313
314                 if (c >= '0' && c <= '9') {
315                     ver = ver * 10 + (c - '0');
316                     continue;
317                 }
318
319                 if (c == '.') {
320                     version += ver * scale;
321
322                     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
323                                    "version: \"%ui\" \"%ui\"",
324                                    modern[i].version, version);
325
326                     if (version > modern[i].version) {
327                         return NGX_HTTP_MODERN_BROWSER;
328                     }
329
330                     ver = 0;
331                     scale /= 100;
332                     continue;
333                 }
334
335                 break;
336             }
337
338             version += ver * scale;
339
340             ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
341                            "version: \"%ui\" \"%ui\"",
342                            modern[i].version, version);
343
344             if (version >= modern[i].version) {
345                 return NGX_HTTP_MODERN_BROWSER;
346             }
347
348             return NGX_HTTP_ANCIENT_BROWSER;
349         }
350
351         if (!cf->modern_unlisted_browsers) {
352             return NGX_HTTP_ANCIENT_BROWSER;
353         }
354     }
355
356     if (cf->netscape4) {
357         if (len > sizeof("Mozilla/4.72 ") - 1
358             && ngx_strncmp(ua, "Mozilla/", sizeof("Mozilla/") - 1) == 0
359             && ua[8] > '0' && ua[8] < '5')
360         {
361             return NGX_HTTP_ANCIENT_BROWSER;
362         }
363     }
364
365     if (cf->ancient_browsers) {
366         ancient = cf->ancient_browsers->elts;
367
368         for (i = 0; i < cf->ancient_browsers->nelts; i++) {
369             if (len >= ancient[i].len
370                 && ngx_strstr(ua, ancient[i].data) != NULL)
371             {
372                 return NGX_HTTP_ANCIENT_BROWSER;
373             }
374         }
375     }
376
377     if (cf->modern_unlisted_browsers) {
378         return NGX_HTTP_MODERN_BROWSER;
379     }
380
381     return NGX_HTTP_ANCIENT_BROWSER;
382 }
383
384
```

```
static ngx_int_t
ngx_http_msie_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    if (r->headers_in.msie) {
        *v = ngx_http_variable_true_value;
        return NGX_OK;
    }

    *v = ngx_http_variable_null_value;
    return NGX_OK;
}


static ngx_int_t
ngx_http_browser_add_variable(ngx_conf_t *cf)
{
    ngx_http_browser_variable_t   *var;
    ngx_http_variable_t           *v;

    for (var = ngx_http_browsers; var->name.len; var++) {

        v = ngx_http_add_variable(cf, &var->name, NGX_HTTP_VAR_CHANGEABLE);
        if (v == NULL) {
            return NGX_ERROR;
        }

        v->get_handler = var->handler;
        v->data = var->data;
    }

    return NGX_OK;
}


static void *
ngx_http_browser_create_conf(ngx_conf_t *cf)
{
    ngx_http_browser_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_browser_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->modern_browsers = NULL;
     *     conf->ancient_browsers = NULL;
     *     conf->modern_browser_value = NULL;
     *     conf->ancient_browser_value = NULL;
     *
     *     conf->modern_unlisted_browsers = 0;
     *     conf->netscape4 = 0;
     */

    return conf;
}


static char *
ngx_http_browser_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_browser_conf_t *prev = parent;
    ngx_http_browser_conf_t *conf = child;

    ngx_uint_t                  i, n;
    ngx_http_modern_browser_t  *browsers, *opera;

    /*
     * At the merge the skip field is used to store the browser slot,
     * it will be used in sorting and then will overwritten
     * with a real skip value.  The zero value means Opera.
     */

```

```
461     if (conf->modern_browsers == NULL && conf->modern_unlisted_browsers == 0) {
462         conf->modern_browsers = prev->modern_browsers;
463         conf->modern_unlisted_browsers = prev->modern_unlisted_browsers;
464 
465     } else if (conf->modern_browsers != NULL) {
466         browsers = conf->modern_browsers->elts;
467 
468         for (i = 0; i < conf->modern_browsers->nelts; i++) {
469             if (browsers[i].skip == 0) {
470                 goto found;
471             }
472         }
473 
474         /*
475          * Opera may contain MSIE string, so if Opera was not enumerated
476          * as modern browsers, then add it and set a unreachable version
477          */
478 
479         opera = ngx_array_push(conf->modern_browsers);
480         if (opera == NULL) {
481             return NGX_CONF_ERROR;
482         }
483 
484         opera->skip = 0;
485         opera->version = 4001000000U;
486 
487         browsers = conf->modern_browsers->elts;
488 
489 found:
490 
491         ngx_qsort(browsers, (size_t) conf->modern_browsers->nelts,
492                   sizeof(ngx_http_modern_browser_t),
493                   ngx_http_modern_browser_sort);
494 
495         for (i = 0; i < conf->modern_browsers->nelts; i++) {
496              n = browsers[i].skip;
497 
498              browsers[i].skip = ngx_http_modern_browser_masks[n].skip;
499              browsers[i].add = ngx_http_modern_browser_masks[n].add;
500              (void) ngx_cpystrn(browsers[i].name,
501                                 ngx_http_modern_browser_masks[n].name, 12);
502         }
503     }
504 
505     if (conf->ancient_browsers == NULL && conf->netscape4 == 0) {
506         conf->ancient_browsers = prev->ancient_browsers;
507         conf->netscape4 = prev->netscape4;
508     }
509 
510     if (conf->modern_browser_value == NULL) {
511         conf->modern_browser_value = prev->modern_browser_value;
512     }
513 
514     if (conf->modern_browser_value == NULL) {
515         conf->modern_browser_value = &ngx_http_variable_true_value;
516     }
517 
518     if (conf->ancient_browser_value == NULL) {
519         conf->ancient_browser_value = prev->ancient_browser_value;
520     }
521 
522     if (conf->ancient_browser_value == NULL) {
523         conf->ancient_browser_value = &ngx_http_variable_true_value;
524     }
525 
526     return NGX_CONF_OK;
527 }
528 
529 
530 static int ngx_libc_cdecl
531 ngx_http_modern_browser_sort(const void *one, const void *two)
532 {
533     ngx_http_modern_browser_t *first = (ngx_http_modern_browser_t *) one;
534     ngx_http_modern_browser_t *second = (ngx_http_modern_browser_t *) two;
535 
536     return (first->skip - second->skip);
```

```
}


static char *
ngx_http_modern_browser(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_browser_conf_t *bcf = conf;

    u_char                           c;
    ngx_str_t                       *value;
    ngx_uint_t                       i, n, version, ver, scale;
    ngx_http_modern_browser_t       *browser;
    ngx_http_modern_browser_mask_t  *mask;

    value = cf->args->elts;

    if (cf->args->nelts == 2) {
        if (ngx_strcmp(value[1].data, "unlisted") == 0) {
            bcf->modern_unlisted_browsers = 1;
            return NGX_CONF_OK;
        }

        return NGX_CONF_ERROR;
    }

    if (bcf->modern_browsers == NULL) {
        bcf->modern_browsers = ngx_array_create(cf->pool, 5,
                                                sizeof(ngx_http_modern_browser_t));
        if (bcf->modern_browsers == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    browser = ngx_array_push(bcf->modern_browsers);
    if (browser == NULL) {
        return NGX_CONF_ERROR;
    }

    mask = ngx_http_modern_browser_masks;

    for (n = 0; mask[n].browser[0] != '\0'; n++) {
        if (ngx_strcasecmp(mask[n].browser, value[1].data) == 0) {
            goto found;
        }
    }

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "unknown browser name \"%V\"", &value[1]);

    return NGX_CONF_ERROR;

found:

    /*
     * at this stage the skip field is used to store the browser slot,
     * it will be used in sorting in merge stage and then will overwritten
     * with a real value
     */

    browser->skip = n;

    version = 0;
    ver = 0;
    scale = 1000000;

    for (i = 0; i < value[2].len; i++) {

        c = value[2].data[i];

        if (c >= '0' && c <= '9') {
            ver = ver * 10 + (c - '0');
            continue;
        }

        if (c == '.') {
            version += ver * scale;
```

```
            ver = 0;
            scale /= 100;
            continue;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid browser version \"%V\"", &value[2]);

        return NGX_CONF_ERROR;
    }

    version += ver * scale;

    browser->version = version;

    return NGX_CONF_OK;
}


static char *
ngx_http_ancient_browser(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_browser_conf_t *bcf = conf;

    ngx_str_t   *value, *browser;
    ngx_uint_t   i;

    value = cf->args->elts;

    for (i = 1; i < cf->args->nelts; i++) {
        if (ngx_strcmp(value[i].data, "netscape4") == 0) {
            bcf->netscape4 = 1;
            continue;
        }

        if (bcf->ancient_browsers == NULL) {
            bcf->ancient_browsers = ngx_array_create(cf->pool, 4,
                                                     sizeof(ngx_str_t));
            if (bcf->ancient_browsers == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        browser = ngx_array_push(bcf->ancient_browsers);
        if (browser == NULL) {
            return NGX_CONF_ERROR;
        }

        *browser = value[i];
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_modern_browser_value(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_browser_conf_t *bcf = conf;

    ngx_str_t  *value;

    bcf->modern_browser_value = ngx_palloc(cf->pool,
                                           sizeof(ngx_http_variable_value_t));
    if (bcf->modern_browser_value == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    bcf->modern_browser_value->len = value[1].len;
    bcf->modern_browser_value->valid = 1;
    bcf->modern_browser_value->no_cacheable = 0;
    bcf->modern_browser_value->not_found = 0;
    bcf->modern_browser_value->data = value[1].data;

```

```
    return NGX_CONF_OK;
}


static char *
ngx_http_ancient_browser_value(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_browser_conf_t *bcf = conf;

    ngx_str_t  *value;

    bcf->ancient_browser_value = ngx_palloc(cf->pool,
                                            sizeof(ngx_http_variable_value_t));
    if (bcf->ancient_browser_value == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    bcf->ancient_browser_value->len = value[1].len;
    bcf->ancient_browser_value->valid = 1;
    bcf->ancient_browser_value->no_cacheable = 0;
    bcf->ancient_browser_value->not_found = 0;
    bcf->ancient_browser_value->data = value[1].data;

    return NGX_CONF_OK;
}
```

One Level Up Top Level

src/http/modules/ngx_http_charset_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_charset_default_types
- ngx_http_charset_filter_commands
- ngx_http_charset_filter_module
- ngx_http_charset_filter_module_ctx
- ngx_http_next_body_filter
- ngx_http_next_header_filter

Data types defined

- ngx_http_charset_conf_ctx_t
- ngx_http_charset_ctx_t
- ngx_http_charset_loc_conf_t
- ngx_http_charset_main_conf_t
- ngx_http_charset_recode_t
- ngx_http_charset_t
- ngx_http_charset_tables_t

Functions defined

- ngx_http_add_charset
- ngx_http_charset_body_filter
- ngx_http_charset_create_loc_conf
- ngx_http_charset_create_main_conf
- ngx_http_charset_ctx
- ngx_http_charset_get_buf
- ngx_http_charset_get_buffer
- ngx_http_charset_header_filter
- ngx_http_charset_map
- ngx_http_charset_map_block
- ngx_http_charset_merge_loc_conf
- ngx_http_charset_postconfiguration
- ngx_http_charset_recode
- ngx_http_charset_recode_from_utf8

- ngx_http_charset_recode_to_utf8
- ngx_http_destination_charset
- ngx_http_get_charset
- ngx_http_main_request_charset
- ngx_http_set_charset
- ngx_http_set_charset_slot
- ngx_http_source_charset

Macros defined

- NGX_HTML_ENTITY_LEN
- NGX_HTTP_CHARSET_OFF
- NGX_HTTP_CHARSET_VAR
- NGX_HTTP_NO_CHARSET
- NGX_UTF_LEN

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define NGX_HTTP_CHARSET_OFF    -2
14  #define NGX_HTTP_NO_CHARSET     -3
15  #define NGX_HTTP_CHARSET_VAR    0x10000
16
17  /* 1 byte length and up to 3 bytes for the UTF-8 encoding of the UCS-2 */
18  #define NGX_UTF_LEN             4
19
20  #define NGX_HTML_ENTITY_LEN     (sizeof("&#1114111;") - 1)
21
22
23  typedef struct {
24      u_char                   **tables;
25      ngx_str_t                  name;
26
27      unsigned                   length:16;
28      unsigned                   utf8:1;
29  } ngx_http_charset_t;
30
31
32  typedef struct {
33      ngx_int_t                  src;
34      ngx_int_t                  dst;
35  } ngx_http_charset_recode_t;
36
37
38  typedef struct {
39      ngx_int_t                  src;
40      ngx_int_t                  dst;
41      u_char                    *src2dst;
```

```
    u_char                     *dst2src;
} ngx_http_charset_tables_t;


typedef struct {
    ngx_array_t                 charsets;       /* ngx_http_charset_t */
    ngx_array_t                 tables;         /* ngx_http_charset_tables_t */
    ngx_array_t                 recodes;        /* ngx_http_charset_recode_t */
} ngx_http_charset_main_conf_t;


typedef struct {
    ngx_int_t                   charset;
    ngx_int_t                   source_charset;
    ngx_flag_t                  override_charset;

    ngx_hash_t                  types;
    ngx_array_t                *types_keys;
} ngx_http_charset_loc_conf_t;


typedef struct {
    u_char                     *table;
    ngx_int_t                   charset;
    ngx_str_t                   charset_name;

    ngx_chain_t                *busy;
    ngx_chain_t                *free_bufs;
    ngx_chain_t                *free_buffers;

    size_t                      saved_len;
    u_char                      saved[NGX_UTF_LEN];

    unsigned                    length:16;
    unsigned                    from_utf8:1;
    unsigned                    to_utf8:1;
} ngx_http_charset_ctx_t;


typedef struct {
    ngx_http_charset_tables_t  *table;
    ngx_http_charset_t         *charset;
    ngx_uint_t                  characters;
} ngx_http_charset_conf_ctx_t;


static ngx_int_t ngx_http_destination_charset(ngx_http_request_t *r,
    ngx_str_t *name);
static ngx_int_t ngx_http_main_request_charset(ngx_http_request_t *r,
    ngx_str_t *name);
static ngx_int_t ngx_http_source_charset(ngx_http_request_t *r,
    ngx_str_t *name);
static ngx_int_t ngx_http_get_charset(ngx_http_request_t *r, ngx_str_t *name);
static ngx_inline void ngx_http_set_charset(ngx_http_request_t *r,
    ngx_str_t *charset);
static ngx_int_t ngx_http_charset_ctx(ngx_http_request_t *r,
    ngx_http_charset_t *charsets, ngx_int_t charset, ngx_int_t source_charset);
static ngx_uint_t ngx_http_charset_recode(ngx_buf_t *b, u_char *table);
 static ngx_chain_t *ngx_http_charset_recode_from_utf8(ngx_pool_t *pool,
     ngx_buf_t *buf, ngx_http_charset_ctx_t *ctx);
 static ngx_chain_t *ngx_http_charset_recode_to_utf8(ngx_pool_t *pool,
     ngx_buf_t *buf, ngx_http_charset_ctx_t *ctx);

 static ngx_chain_t *ngx_http_charset_get_buf(ngx_pool_t *pool,
     ngx_http_charset_ctx_t *ctx);
 static ngx_chain_t *ngx_http_charset_get_buffer(ngx_pool_t *pool,
     ngx_http_charset_ctx_t *ctx, size_t size);

 static char *ngx_http_charset_map_block(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static char *ngx_http_charset_map(ngx_conf_t *cf, ngx_command_t *dummy,
     void *conf);

 static char *ngx_http_set_charset_slot(ngx_conf_t *cf, ngx_command_t *cmd,
     void *conf);
 static ngx_int_t ngx_http_add_charset(ngx_array_t *charsets, ngx_str_t *name);
```

```
118
119 static void *ngx_http_charset_create_main_conf(ngx_conf_t *cf);
120 static void *ngx_http_charset_create_loc_conf(ngx_conf_t *cf);
121 static char *ngx_http_charset_merge_loc_conf(ngx_conf_t *cf,
122     void *parent, void *child);
123 static ngx_int_t ngx_http_charset_postconfiguration(ngx_conf_t *cf);
124
125
126 ngx_str_t  ngx_http_charset_default_types[] = {
127     ngx_string("text/html"),
128     ngx_string("text/xml"),
129     ngx_string("text/plain"),
130     ngx_string("text/vnd.wap.wml"),
131     ngx_string("application/javascript"),
132     ngx_string("application/rss+xml"),
133     ngx_null_string
134 };
135
136
137 static ngx_command_t  ngx_http_charset_filter_commands[] = {
138
139     { ngx_string("charset"),
140       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF
141                         |NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
142       ngx_http_set_charset_slot,
143       NGX_HTTP_LOC_CONF_OFFSET,
144       offsetof(ngx_http_charset_loc_conf_t, charset),
145       NULL },
146
147     { ngx_string("source_charset"),
148       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF
149                         |NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
150       ngx_http_set_charset_slot,
151       NGX_HTTP_LOC_CONF_OFFSET,
152       offsetof(ngx_http_charset_loc_conf_t, source_charset),
153       NULL },
154
155     { ngx_string("override_charset"),
156       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF
157                         |NGX_HTTP_LIF_CONF|NGX_CONF_FLAG,
158       ngx_conf_set_flag_slot,
159       NGX_HTTP_LOC_CONF_OFFSET,
160       offsetof(ngx_http_charset_loc_conf_t, override_charset),
161       NULL },
162
163     { ngx_string("charset_types"),
164       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
165       ngx_http_types_slot,
166       NGX_HTTP_LOC_CONF_OFFSET,
167       offsetof(ngx_http_charset_loc_conf_t, types_keys),
168       &ngx_http_charset_default_types[0] },
169
170     { ngx_string("charset_map"),
171       NGX_HTTP_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_TAKE2,
172       ngx_http_charset_map_block,
173       NGX_HTTP_MAIN_CONF_OFFSET,
174       0,
175       NULL },
176
177       ngx_null_command
178 };
179
180
181 static ngx_http_module_t  ngx_http_charset_filter_module_ctx = {
182     NULL,                                  /* preconfiguration */
183     ngx_http_charset_postconfiguration,    /* postconfiguration */
184
185     ngx_http_charset_create_main_conf,     /* create main configuration */
186     NULL,                                  /* init main configuration */
187
188     NULL,                                  /* create server configuration */
189     NULL,                                  /* merge server configuration */
190
191     ngx_http_charset_create_loc_conf,      /* create location configuration */
192     ngx_http_charset_merge_loc_conf        /* merge location configuration */
193 };
```

```
194
195
196 ngx_module_t  ngx_http_charset_filter_module = {
197     NGX_MODULE_V1,
198     &ngx_http_charset_filter_module_ctx,    /* module context */
199     ngx_http_charset_filter_commands,       /* module directives */
200     NGX_HTTP_MODULE,                        /* module type */
201     NULL,                                   /* init master */
202     NULL,                                   /* init module */
203     NULL,                                   /* init process */
204     NULL,                                   /* init thread */
205     NULL,                                   /* exit thread */
206     NULL,                                   /* exit process */
207     NULL,                                   /* exit master */
208     NGX_MODULE_V1_PADDING
209 };
210
211
212 static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
213 static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;
214
215
216 static ngx_int_t
217 ngx_http_charset_header_filter(ngx_http_request_t *r)
218 {
219     ngx_int_t                    charset, source_charset;
220     ngx_str_t                    dst, src;
221     ngx_http_charset_t          *charsets;
222     ngx_http_charset_main_conf_t  *mcf;
223
224     if (r == r->main) {
225         charset = ngx_http_destination_charset(r, &dst);
226
227     } else {
228         charset = ngx_http_main_request_charset(r, &dst);
229     }
230
231     if (charset == NGX_ERROR) {
232         return NGX_ERROR;
233     }
234
235     if (charset == NGX_DECLINED) {
236         return ngx_http_next_header_filter(r);
237     }
238
239     /* charset: charset index or NGX_HTTP_NO_CHARSET */
240
241     source_charset = ngx_http_source_charset(r, &src);
242
243     if (source_charset == NGX_ERROR) {
244         return NGX_ERROR;
245     }
246
247     /*
248      * source_charset: charset index, NGX_HTTP_NO_CHARSET,
249      *                 or NGX_HTTP_CHARSET_OFF
250      */
251
252     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
253                    "charset: \"%V\" > \"%V\"", &src, &dst);
254
255     if (source_charset == NGX_HTTP_CHARSET_OFF) {
256         ngx_http_set_charset(r, &dst);
257
258         return ngx_http_next_header_filter(r);
259     }
260
261     if (charset == NGX_HTTP_NO_CHARSET
262         || source_charset == NGX_HTTP_NO_CHARSET)
263     {
264         if (source_charset != charset
265             || ngx_strncasecmp(dst.data, src.data, dst.len) != 0)
266         {
267             goto no_charset_map;
268         }
269
```

```
        ngx_http_set_charset(r, &dst);

        return ngx_http_next_header_filter(r);
    }

    if (source_charset == charset) {
        r->headers_out.content_type.len = r->headers_out.content_type_len;

        ngx_http_set_charset(r, &dst);

        return ngx_http_next_header_filter(r);
    }

    /* source_charset != charset */

    if (r->headers_out.content_encoding
        && r->headers_out.content_encoding->value.len)
    {
        return ngx_http_next_header_filter(r);
    }

    mcf = ngx_http_get_module_main_conf(r, ngx_http_charset_filter_module);
    charsets = mcf->charsets.elts;

    if (charsets[source_charset].tables == NULL
        || charsets[source_charset].tables[charset] == NULL)
    {
        goto no_charset_map;
    }

    r->headers_out.content_type.len = r->headers_out.content_type_len;

    ngx_http_set_charset(r, &dst);

    return ngx_http_charset_ctx(r, charsets, charset, source_charset);

no_charset_map:

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "no \"charset_map\" between the charsets \"%V\" and \"%V\"",
                  &src, &dst);

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_destination_charset(ngx_http_request_t *r, ngx_str_t *name)
{
    ngx_int_t                    charset;
    ngx_http_charset_t          *charsets;
    ngx_http_variable_value_t   *vv;
    ngx_http_charset_loc_conf_t *mlcf;
    ngx_http_charset_main_conf_t *mcf;

    if (r->headers_out.content_type.len == 0) {
        return NGX_DECLINED;
    }

    if (r->headers_out.override_charset
        && r->headers_out.override_charset->len)
    {
        *name = *r->headers_out.override_charset;

        charset = ngx_http_get_charset(r, name);

        if (charset != NGX_HTTP_NO_CHARSET) {
            return charset;
        }

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "unknown charset \"%V\" to override", name);

        return NGX_DECLINED;
    }

```

```
    mlcf = ngx_http_get_module_loc_conf(r, ngx_http_charset_filter_module);
    charset = mlcf->charset;

    if (charset == NGX_HTTP_CHARSET_OFF) {
        return NGX_DECLINED;
    }

    if (r->headers_out.charset.len) {
        if (mlcf->override_charset == 0) {
            return NGX_DECLINED;
        }

    } else {
        if (ngx_http_test_content_type(r, &mlcf->types) == NULL) {
            return NGX_DECLINED;
        }
    }

    if (charset < NGX_HTTP_CHARSET_VAR) {
        mcf = ngx_http_get_module_main_conf(r, ngx_http_charset_filter_module);
        charsets = mcf->charsets.elts;
        *name = charsets[charset].name;
        return charset;
    }

    vv = ngx_http_get_indexed_variable(r, charset - NGX_HTTP_CHARSET_VAR);

    if (vv == NULL || vv->not_found) {
        return NGX_ERROR;
    }

    name->len = vv->len;
    name->data = vv->data;

    return ngx_http_get_charset(r, name);
}


static ngx_int_t
ngx_http_main_request_charset(ngx_http_request_t *r, ngx_str_t *src)
{
    ngx_int_t                 charset;
    ngx_str_t                *main_charset;
    ngx_http_charset_ctx_t   *ctx;

    ctx = ngx_http_get_module_ctx(r->main, ngx_http_charset_filter_module);

    if (ctx) {
        *src = ctx->charset_name;
        return ctx->charset;
    }

    main_charset = &r->main->headers_out.charset;

    if (main_charset->len == 0) {
        return NGX_DECLINED;
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_charset_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r->main, ctx, ngx_http_charset_filter_module);

    charset = ngx_http_get_charset(r, main_charset);

    ctx->charset = charset;
    ctx->charset_name = *main_charset;
    *src = *main_charset;

    return charset;
}


static ngx_int_t
```

```
ngx_http_source_charset(ngx_http_request_t *r, ngx_str_t *name)
{
    ngx_int_t                     charset;
    ngx_http_charset_t           *charsets;
    ngx_http_variable_value_t    *vv;
    ngx_http_charset_loc_conf_t  *lcf;
    ngx_http_charset_main_conf_t  *mcf;

    if (r->headers_out.charset.len) {
        *name = r->headers_out.charset;
        return ngx_http_get_charset(r, name);
    }

    lcf = ngx_http_get_module_loc_conf(r, ngx_http_charset_filter_module);

    charset = lcf->source_charset;

    if (charset == NGX_HTTP_CHARSET_OFF) {
        name->len = 0;
        return charset;
    }

    if (charset < NGX_HTTP_CHARSET_VAR) {
        mcf = ngx_http_get_module_main_conf(r, ngx_http_charset_filter_module);
        charsets = mcf->charsets.elts;
        *name = charsets[charset].name;
        return charset;
    }

    vv = ngx_http_get_indexed_variable(r, charset - NGX_HTTP_CHARSET_VAR);

    if (vv == NULL || vv->not_found) {
        return NGX_ERROR;
    }

    name->len = vv->len;
    name->data = vv->data;

    return ngx_http_get_charset(r, name);
}


static ngx_int_t
ngx_http_get_charset(ngx_http_request_t *r, ngx_str_t *name)
{
    ngx_uint_t                     i, n;
    ngx_http_charset_t            *charset;
    ngx_http_charset_main_conf_t  *mcf;

    mcf = ngx_http_get_module_main_conf(r, ngx_http_charset_filter_module);

    charset = mcf->charsets.elts;
    n = mcf->charsets.nelts;

    for (i = 0; i < n; i++) {
        if (charset[i].name.len != name->len) {
            continue;
        }

        if (ngx_strncasecmp(charset[i].name.data, name->data, name->len) == 0) {
            return i;
        }
    }

    return NGX_HTTP_NO_CHARSET;
}


static ngx_inline void
ngx_http_set_charset(ngx_http_request_t *r, ngx_str_t *charset)
{
    if (r != r->main) {
        return;
    }

    if (r->headers_out.status == NGX_HTTP_MOVED_PERMANENTLY
```

```
        || r->headers_out.status == NGX_HTTP_MOVED_TEMPORARILY)
    {
        /*
         * do not set charset for the redirect because NN 4.x
         * use this charset instead of the next page charset
         */

        r->headers_out.charset.len = 0;
        return;
    }

    r->headers_out.charset = *charset;
}


static ngx_int_t
ngx_http_charset_ctx(ngx_http_request_t *r, ngx_http_charset_t *charsets,
    ngx_int_t charset, ngx_int_t source_charset)
{
    ngx_http_charset_ctx_t  *ctx;

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_charset_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_charset_filter_module);

    ctx->table = charsets[source_charset].tables[charset];
    ctx->charset = charset;
    ctx->charset_name = charsets[charset].name;
    ctx->length = charsets[charset].length;
    ctx->from_utf8 = charsets[source_charset].utf8;
    ctx->to_utf8 = charsets[charset].utf8;

    r->filter_need_in_memory = 1;

    if ((ctx->to_utf8 || ctx->from_utf8) && r == r->main) {
        ngx_http_clear_content_length(r);

    } else {
        r->filter_need_temporary = 1;
    }

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_charset_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    ngx_int_t                rc;
    ngx_buf_t               *b;
    ngx_chain_t             *cl, *out, **ll;
    ngx_http_charset_ctx_t  *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_charset_filter_module);

    if (ctx == NULL || ctx->table == NULL) {
        return ngx_http_next_body_filter(r, in);
    }

    if ((ctx->to_utf8 || ctx->from_utf8) || ctx->busy) {

        out = NULL;
        ll = &out;

        for (cl = in; cl; cl = cl->next) {
            b = cl->buf;

            if (ngx_buf_size(b) == 0) {

                *ll = ngx_alloc_chain_link(r->pool);
                if (*ll == NULL) {
                    return NGX_ERROR;
                }
```

```
574
575                 (*ll)->buf = b;
576                 (*ll)->next = NULL;
577
578                 ll = &(*ll)->next;
579
580                 continue;
581             }
582
583             if (ctx->to_utf8) {
584                 *ll = ngx_http_charset_recode_to_utf8(r->pool, b, ctx);
585
586             } else {
587                 *ll = ngx_http_charset_recode_from_utf8(r->pool, b, ctx);
588             }
589
590             if (*ll == NULL) {
591                 return NGX_ERROR;
592             }
593
594             while (*ll) {
595                 ll = &(*ll)->next;
596             }
597         }
598
599         rc = ngx_http_next_body_filter(r, out);
600
601         if (out) {
602             if (ctx->busy == NULL) {
603                 ctx->busy = out;
604
605             } else {
606                 for (cl = ctx->busy; cl->next; cl = cl->next) { /* void */ }
607                 cl->next = out;
608             }
609         }
610
611         while (ctx->busy) {
612
613             cl = ctx->busy;
614             b = cl->buf;
615
616             if (ngx_buf_size(b) != 0) {
617                 break;
618             }
619
620             ctx->busy = cl->next;
621
622             if (b->tag != (ngx_buf_tag_t) &ngx_http_charset_filter_module) {
623                 continue;
624             }
625
626             if (b->shadow) {
627                 b->shadow->pos = b->shadow->last;
628             }
629
630             if (b->pos) {
631                 cl->next = ctx->free_buffers;
632                 ctx->free_buffers = cl;
633                 continue;
634             }
635
636             cl->next = ctx->free_bufs;
637             ctx->free_bufs = cl;
638         }
639
640         return rc;
641     }
642
643     for (cl = in; cl; cl = cl->next) {
644         (void) ngx_http_charset_recode(cl->buf, ctx->table);
645     }
646
647     return ngx_http_next_body_filter(r, in);
648 }
649
```

```
650
651 static ngx_uint_t
652 ngx_http_charset_recode(ngx_buf_t *b, u_char *table)
653 {
654     u_char  *p, *last;
655
656     last = b->last;
657
658     for (p = b->pos; p < last; p++) {
659
660         if (*p != table[*p]) {
661             goto recode;
662         }
663     }
664
665     return 0;
666
667 recode:
668
669     do {
670         if (*p != table[*p]) {
671             *p = table[*p];
672         }
673
674         p++;
675
676     } while (p < last);
677
678     b->in_file = 0;
679
680     return 1;
681 }
682
683
684 static ngx_chain_t *
685 ngx_http_charset_recode_from_utf8(ngx_pool_t *pool, ngx_buf_t *buf,
686     ngx_http_charset_ctx_t *ctx)
687 {
688     size_t        len, size;
689     u_char        c, *p, *src, *dst, *saved, **table;
690     uint32_t      n;
691     ngx_buf_t    *b;
692     ngx_uint_t    i;
693     ngx_chain_t  *out, *cl, **ll;
694
695     src = buf->pos;
696
697     if (ctx->saved_len == 0) {
698
699         for ( /* void */ ; src < buf->last; src++) {
700
701             if (*src < 0x80) {
702                 continue;
703             }
704
705             len = src - buf->pos;
706
707             if (len > 512) {
708                 out = ngx_http_charset_get_buf(pool, ctx);
709                 if (out == NULL) {
710                     return NULL;
711                 }
712
713                 b = out->buf;
714
715                 b->temporary = buf->temporary;
716                 b->memory = buf->memory;
717                 b->mmap = buf->mmap;
718                 b->flush = buf->flush;
719
720                 b->pos = buf->pos;
721                 b->last = src;
722
723                 out->buf = b;
724                 out->next = NULL;
725
```

```
                size = buf->last - src;

                saved = src;
                n = ngx_utf8_decode(&saved, size);

                if (n == 0xfffffffe) {
                    /* incomplete UTF-8 symbol */

                    ngx_memcpy(ctx->saved, src, size);
                    ctx->saved_len = size;

                    b->shadow = buf;

                    return out;
                }

            } else {
                out = NULL;
                size = len + buf->last - src;
                src = buf->pos;
            }

            if (size < NGX_HTML_ENTITY_LEN) {
                size += NGX_HTML_ENTITY_LEN;
            }

            cl = ngx_http_charset_get_buffer(pool, ctx, size);
            if (cl == NULL) {
                return NULL;
            }

            if (out) {
                out->next = cl;

            } else {
                out = cl;
            }

            b = cl->buf;
            dst = b->pos;

            goto recode;
        }

        out = ngx_alloc_chain_link(pool);
        if (out == NULL) {
            return NULL;
        }

        out->buf = buf;
        out->next = NULL;

        return out;
    }

    /* process incomplete UTF sequence from previous buffer */

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pool->log, 0,
                   "http charset utf saved: %z", ctx->saved_len);

    p = src;

    for (i = ctx->saved_len; i < NGX_UTF_LEN; i++) {
        ctx->saved[i] = *p++;

        if (p == buf->last) {
            break;
        }
    }

    saved = ctx->saved;
    n = ngx_utf8_decode(&saved, i);

    c = '\0';

    if (n < 0x10000) {
```

```
        table = (u_char **) ctx->table;
        p = table[n >> 8];

        if (p) {
            c = p[n & 0xff];
        }

    } else if (n == 0xfffffffe) {

        /* incomplete UTF-8 symbol */

        if (i < NGX_UTF_LEN) {
            out = ngx_http_charset_get_buf(pool, ctx);
            if (out == NULL) {
                return NULL;
            }

            b = out->buf;

            b->pos = buf->pos;
            b->last = buf->last;
            b->sync = 1;
            b->shadow = buf;

            ngx_memcpy(&ctx->saved[ctx->saved_len], src, i);
            ctx->saved_len += i;

            return out;
        }
    }

    size = buf->last - buf->pos;

    if (size < NGX_HTML_ENTITY_LEN) {
        size += NGX_HTML_ENTITY_LEN;
    }

    cl = ngx_http_charset_get_buffer(pool, ctx, size);
    if (cl == NULL) {
        return NULL;
    }

    out = cl;

    b = cl->buf;
    dst = b->pos;

    if (c) {
        *dst++ = c;

    } else if (n == 0xfffffffe) {
        *dst++ = '?';

        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pool->log, 0,
                       "http charset invalid utf 0");

        saved = &ctx->saved[NGX_UTF_LEN];

    } else if (n > 0x10ffff) {
        *dst++ = '?';

        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pool->log, 0,
                       "http charset invalid utf 1");

    } else {
        dst = ngx_sprintf(dst, "&#%uD;", n);
    }

    src += (saved - ctx->saved) - ctx->saved_len;
    ctx->saved_len = 0;

recode:

    ll = &cl->next;

    table = (u_char **) ctx->table;
```

```
878
879     while (src < buf->last) {
880
881         if ((size_t) (b->end - dst) < NGX_HTML_ENTITY_LEN) {
882             b->last = dst;
883
884             size = buf->last - src + NGX_HTML_ENTITY_LEN;
885
886             cl = ngx_http_charset_get_buffer(pool, ctx, size);
887             if (cl == NULL) {
888                 return NULL;
889             }
890
891             *ll = cl;
892             ll = &cl->next;
893
894             b = cl->buf;
895             dst = b->pos;
896         }
897
898         if (*src < 0x80) {
899             *dst++ = *src++;
900             continue;
901         }
902
903         len = buf->last - src;
904
905         n = ngx_utf8_decode(&src, len);
906
907         if (n < 0x10000) {
908
909             p = table[n >> 8];
910
911             if (p) {
912                 c = p[n & 0xff];
913
914                 if (c) {
915                     *dst++ = c;
916                     continue;
917                 }
918             }
919
920             dst = ngx_sprintf(dst, "&#%uD;", n);
921
922             continue;
923         }
924
925         if (n == 0xfffffffe) {
926             /* incomplete UTF-8 symbol */
927
928             ngx_memcpy(ctx->saved, src, len);
929             ctx->saved_len = len;
930
931             if (b->pos == dst) {
932                 b->sync = 1;
933                 b->temporary = 0;
934             }
935
936             break;
937         }
938
939         if (n > 0x10ffff) {
940             *dst++ = '?';
941
942             ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pool->log, 0,
943                            "http charset invalid utf 2");
944
945             continue;
946         }
947
948         /* n > 0xffff */
949
950         dst = ngx_sprintf(dst, "&#%uD;", n);
951     }
952
953     b->last = dst;
```

```
954
955     b->last_buf = buf->last_buf;
956     b->last_in_chain = buf->last_in_chain;
957     b->flush = buf->flush;
958
959     b->shadow = buf;
960
961     return out;
962 }
963
964
965 static ngx_chain_t *
966 ngx_http_charset_recode_to_utf8(ngx_pool_t *pool, ngx_buf_t *buf,
967     ngx_http_charset_ctx_t *ctx)
968 {
969     size_t        len, size;
970     u_char       *p, *src, *dst, *table;
971     ngx_buf_t    *b;
972     ngx_chain_t  *out, *cl, **ll;
973
974     table = ctx->table;
975
976     for (src = buf->pos; src < buf->last; src++) {
977         if (table[*src * NGX_UTF_LEN] == '\1') {
978             continue;
979         }
980
981         goto recode;
982     }
983
984     out = ngx_alloc_chain_link(pool);
985     if (out == NULL) {
986         return NULL;
987     }
988
989     out->buf = buf;
990     out->next = NULL;
991
992     return out;
993
994 recode:
995
996     /*
997      * we assume that there are about half of characters to be recoded,
998      * so we preallocate "size / 2 + size / 2 * ctx->length"
999      */
1000
1001     len = src - buf->pos;
1002
1003     if (len > 512) {
1004         out = ngx_http_charset_get_buf(pool, ctx);
1005         if (out == NULL) {
1006             return NULL;
1007         }
1008
1009         b = out->buf;
1010
1011         b->temporary = buf->temporary;
1012         b->memory = buf->memory;
1013         b->mmap = buf->mmap;
1014         b->flush = buf->flush;
1015
1016         b->pos = buf->pos;
1017         b->last = src;
1018
1019         out->buf = b;
1020         out->next = NULL;
1021
1022         size = buf->last - src;
1023         size = size / 2 + size / 2 * ctx->length;
1024
1025     } else {
1026         out = NULL;
1027
1028         size = buf->last - src;
1029         size = len + size / 2 + size / 2 * ctx->length;
```

```
1030
1031            src = buf->pos;
1032        }
1033
1034        cl = ngx_http_charset_get_buffer(pool, ctx, size);
1035        if (cl == NULL) {
1036            return NULL;
1037        }
1038
1039        if (out) {
1040            out->next = cl;
1041
1042        } else {
1043            out = cl;
1044        }
1045
1046        ll = &cl->next;
1047
1048        b = cl->buf;
1049        dst = b->pos;
1050
1051        while (src < buf->last) {
1052
1053            p = &table[*src++ * NGX_UTF_LEN];
1054            len = *p++;
1055
1056            if ((size_t) (b->end - dst) < len) {
1057                b->last = dst;
1058
1059                size = buf->last - src;
1060                size = len + size / 2 + size / 2 * ctx->length;
1061
1062                cl = ngx_http_charset_get_buffer(pool, ctx, size);
1063                if (cl == NULL) {
1064                    return NULL;
1065                }
1066
1067                *ll = cl;
1068                ll = &cl->next;
1069
1070                b = cl->buf;
1071                dst = b->pos;
1072            }
1073
1074            while (len) {
1075                *dst++ = *p++;
1076                len--;
1077            }
1078        }
1079
1080        b->last = dst;
1081
1082        b->last_buf = buf->last_buf;
1083        b->last_in_chain = buf->last_in_chain;
1084        b->flush = buf->flush;
1085
1086        b->shadow = buf;
1087
1088        return out;
1089    }
1090
1091
1092    static ngx_chain_t *
1093    ngx_http_charset_get_buf(ngx_pool_t *pool, ngx_http_charset_ctx_t *ctx)
1094    {
1095        ngx_chain_t  *cl;
1096
1097        cl = ctx->free_bufs;
1098
1099        if (cl) {
1100            ctx->free_bufs = cl->next;
1101
1102            cl->buf->shadow = NULL;
1103            cl->next = NULL;
1104
1105            return cl;
```

```
1106      }
1107
1108      cl = ngx_alloc_chain_link(pool);
1109      if (cl == NULL) {
1110          return NULL;
1111      }
1112
1113      cl->buf = ngx_calloc_buf(pool);
1114      if (cl->buf == NULL) {
1115          return NULL;
1116      }
1117
1118      cl->next = NULL;
1119
1120      cl->buf->tag = (ngx_buf_tag_t) &ngx_http_charset_filter_module;
1121
1122      return cl;
1123  }
1124
1125
1126  static ngx_chain_t *
1127  ngx_http_charset_get_buffer(ngx_pool_t *pool, ngx_http_charset_ctx_t *ctx,
1128      size_t size)
1129  {
1130      ngx_buf_t    *b;
1131      ngx_chain_t  *cl, **ll;
1132
1133      for (ll = &ctx->free_buffers, cl = ctx->free_buffers;
1134           cl;
1135           ll = &cl->next, cl = cl->next)
1136      {
1137          b = cl->buf;
1138
1139          if ((size_t) (b->end - b->start) >= size) {
1140              *ll = cl->next;
1141              cl->next = NULL;
1142
1143              b->pos = b->start;
1144              b->temporary = 1;
1145              b->shadow = NULL;
1146
1147              return cl;
1148          }
1149      }
1150
1151      cl = ngx_alloc_chain_link(pool);
1152      if (cl == NULL) {
1153          return NULL;
1154      }
1155
1156      cl->buf = ngx_create_temp_buf(pool, size);
1157      if (cl->buf == NULL) {
1158          return NULL;
1159      }
1160
1161      cl->next = NULL;
1162
1163      cl->buf->temporary = 1;
1164      cl->buf->tag = (ngx_buf_tag_t) &ngx_http_charset_filter_module;
1165
1166      return cl;
1167  }
1168
1169
1170  static char *
1171  ngx_http_charset_map_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1172  {
1173      ngx_http_charset_main_conf_t  *mcf = conf;
1174
1175      char                         *rv;
1176      u_char                       *p, *dst2src, **pp;
1177      ngx_int_t                     src, dst;
1178      ngx_uint_t                    i, n;
1179      ngx_str_t                    *value;
1180      ngx_conf_t                    pvcf;
1181      ngx_http_charset_t           *charset;
```

```
    ngx_http_charset_tables_t    *table;
    ngx_http_charset_conf_ctx_t   ctx;

    value = cf->args->elts;

    src = ngx_http_add_charset(&mcf->charsets, &value[1]);
    if (src == NGX_ERROR) {
        return NGX_CONF_ERROR;
    }

    dst = ngx_http_add_charset(&mcf->charsets, &value[2]);
    if (dst == NGX_ERROR) {
        return NGX_CONF_ERROR;
    }

    if (src == dst) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"charset_map\" between the same charsets "
                           "\"%V\" and \"%V\"", &value[1], &value[2]);
        return NGX_CONF_ERROR;
    }

    table = mcf->tables.elts;
    for (i = 0; i < mcf->tables.nelts; i++) {
        if ((src == table->src && dst == table->dst)
             || (src == table->dst && dst == table->src))
        {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "duplicate \"charset_map\" between "
                               "\"%V\" and \"%V\"", &value[1], &value[2]);
            return NGX_CONF_ERROR;
        }
    }

    table = ngx_array_push(&mcf->tables);
    if (table == NULL) {
        return NGX_CONF_ERROR;
    }

    table->src = src;
    table->dst = dst;

    if (ngx_strcasecmp(value[2].data, (u_char *) "utf-8") == 0) {
        table->src2dst = ngx_pcalloc(cf->pool, 256 * NGX_UTF_LEN);
        if (table->src2dst == NULL) {
            return NGX_CONF_ERROR;
        }

        table->dst2src = ngx_pcalloc(cf->pool, 256 * sizeof(void *));
        if (table->dst2src == NULL) {
            return NGX_CONF_ERROR;
        }

        dst2src = ngx_pcalloc(cf->pool, 256);
        if (dst2src == NULL) {
            return NGX_CONF_ERROR;
        }

        pp = (u_char **) &table->dst2src[0];
        pp[0] = dst2src;

        for (i = 0; i < 128; i++) {
            p = &table->src2dst[i * NGX_UTF_LEN];
            p[0] = '\1';
            p[1] = (u_char) i;
            dst2src[i] = (u_char) i;
        }

        for (/* void */; i < 256; i++) {
            p = &table->src2dst[i * NGX_UTF_LEN];
            p[0] = '\1';
            p[1] = '?';
        }

    } else {
        table->src2dst = ngx_palloc(cf->pool, 256);
```

```
        if (table->src2dst == NULL) {
            return NGX_CONF_ERROR;
        }

        table->dst2src = ngx_palloc(cf->pool, 256);
        if (table->dst2src == NULL) {
            return NGX_CONF_ERROR;
        }

        for (i = 0; i < 128; i++) {
            table->src2dst[i] = (u_char) i;
            table->dst2src[i] = (u_char) i;
        }

        for (/* void */; i < 256; i++) {
            table->src2dst[i] = '?';
            table->dst2src[i] = '?';
        }
    }

    charset = mcf->charsets.elts;

    ctx.table = table;
    ctx.charset = &charset[dst];
    ctx.characters = 0;

    pvcf = *cf;
    cf->ctx = &ctx;
    cf->handler = ngx_http_charset_map;
    cf->handler_conf = conf;

    rv = ngx_conf_parse(cf, NULL);

    *cf = pvcf;

    if (ctx.characters) {
        n = ctx.charset->length;
        ctx.charset->length /= ctx.characters;

        if (((n * 10) / ctx.characters) % 10 > 4) {
            ctx.charset->length++;
        }
    }

    return rv;
}


static char *
ngx_http_charset_map(ngx_conf_t *cf, ngx_command_t *dummy, void *conf)
{
    u_char                       *p, *dst2src, **pp;
    uint32_t                      n;
    ngx_int_t                     src, dst;
    ngx_str_t                    *value;
    ngx_uint_t                    i;
    ngx_http_charset_tables_t    *table;
    ngx_http_charset_conf_ctx_t  *ctx;

    if (cf->args->nelts != 2) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid parameters number");
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    src = ngx_hextoi(value[0].data, value[0].len);
    if (src == NGX_ERROR || src > 255) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid value \"%V\"", &value[0]);
        return NGX_CONF_ERROR;
    }

    ctx = cf->ctx;
    table = ctx->table;

```

```
1334      if (ctx->charset->utf8) {
1335          p = &table->src2dst[src * NGX_UTF_LEN];
1336
1337          *p++ = (u_char) (value[1].len / 2);
1338
1339          for (i = 0; i < value[1].len; i += 2) {
1340              dst = ngx_hextoi(&value[1].data[i], 2);
1341              if (dst == NGX_ERROR || dst > 255) {
1342                  ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1343                                     "invalid value \"%V\"", &value[1]);
1344                  return NGX_CONF_ERROR;
1345              }
1346
1347              *p++ = (u_char) dst;
1348          }
1349
1350          i /= 2;
1351
1352          ctx->charset->length += i;
1353          ctx->characters++;
1354
1355          p = &table->src2dst[src * NGX_UTF_LEN] + 1;
1356
1357          n = ngx_utf8_decode(&p, i);
1358
1359          if (n > 0xffff) {
1360              ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1361                                 "invalid value \"%V\"", &value[1]);
1362              return NGX_CONF_ERROR;
1363          }
1364
1365          pp = (u_char **) &table->dst2src[0];
1366
1367          dst2src = pp[n >> 8];
1368
1369          if (dst2src == NULL) {
1370              dst2src = ngx_pcalloc(cf->pool, 256);
1371              if (dst2src == NULL) {
1372                  return NGX_CONF_ERROR;
1373              }
1374
1375              pp[n >> 8] = dst2src;
1376          }
1377
1378          dst2src[n & 0xff] = (u_char) src;
1379
1380      } else {
1381          dst = ngx_hextoi(value[1].data, value[1].len);
1382          if (dst == NGX_ERROR || dst > 255) {
1383              ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1384                                 "invalid value \"%V\"", &value[1]);
1385              return NGX_CONF_ERROR;
1386          }
1387
1388          table->src2dst[src] = (u_char) dst;
1389          table->dst2src[dst] = (u_char) src;
1390      }
1391
1392      return NGX_CONF_OK;
1393  }
1394
1395
1396  static char *
1397  ngx_http_set_charset_slot(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1398  {
1399      char  *p = conf;
1400
1401      ngx_int_t                    *cp;
1402      ngx_str_t                    *value, var;
1403      ngx_http_charset_main_conf_t  *mcf;
1404
1405      cp = (ngx_int_t *) (p + cmd->offset);
1406
1407      if (*cp != NGX_CONF_UNSET) {
1408          return "is duplicate";
1409      }
```

```
1410
1411      value = cf->args->elts;
1412
1413      if (cmd->offset == offsetof(ngx_http_charset_loc_conf_t, charset)
1414          && ngx_strcmp(value[1].data, "off") == 0)
1415      {
1416          *cp = NGX_HTTP_CHARSET_OFF;
1417          return NGX_CONF_OK;
1418      }
1419
1420
1421      if (value[1].data[0] == '$') {
1422          var.len = value[1].len - 1;
1423          var.data = value[1].data + 1;
1424
1425          *cp = ngx_http_get_variable_index(cf, &var);
1426
1427          if (*cp == NGX_ERROR) {
1428              return NGX_CONF_ERROR;
1429          }
1430
1431          *cp += NGX_HTTP_CHARSET_VAR;
1432
1433          return NGX_CONF_OK;
1434      }
1435
1436      mcf = ngx_http_conf_get_module_main_conf(cf,
1437                                               ngx_http_charset_filter_module);
1438
1439      *cp = ngx_http_add_charset(&mcf->charsets, &value[1]);
1440      if (*cp == NGX_ERROR) {
1441          return NGX_CONF_ERROR;
1442      }
1443
1444      return NGX_CONF_OK;
1445  }
1446
1447
1448  static ngx_int_t
1449  ngx_http_add_charset(ngx_array_t *charsets, ngx_str_t *name)
1450  {
1451      ngx_uint_t           i;
1452      ngx_http_charset_t  *c;
1453
1454      c = charsets->elts;
1455      for (i = 0; i < charsets->nelts; i++) {
1456          if (name->len != c[i].name.len) {
1457              continue;
1458          }
1459
1460          if (ngx_strcasecmp(name->data, c[i].name.data) == 0) {
1461              break;
1462          }
1463      }
1464
1465      if (i < charsets->nelts) {
1466          return i;
1467      }
1468
1469      c = ngx_array_push(charsets);
1470      if (c == NULL) {
1471          return NGX_ERROR;
1472      }
1473
1474      c->tables = NULL;
1475      c->name = *name;
1476      c->length = 0;
1477
1478      if (ngx_strcasecmp(name->data, (u_char *) "utf-8") == 0) {
1479          c->utf8 = 1;
1480
1481      } else {
1482          c->utf8 = 0;
1483      }
1484
1485      return i;
```

```
}


static void *
ngx_http_charset_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_charset_main_conf_t  *mcf;

    mcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_charset_main_conf_t));
    if (mcf == NULL) {
        return NULL;
    }

    if (ngx_array_init(&mcf->charsets, cf->pool, 2, sizeof(ngx_http_charset_t))
        != NGX_OK)
    {
        return NULL;
    }

    if (ngx_array_init(&mcf->tables, cf->pool, 1,
                       sizeof(ngx_http_charset_tables_t))
        != NGX_OK)
    {
        return NULL;
    }

    if (ngx_array_init(&mcf->recodes, cf->pool, 2,
                       sizeof(ngx_http_charset_recode_t))
        != NGX_OK)
    {
        return NULL;
    }

    return mcf;
}


static void *
ngx_http_charset_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_charset_loc_conf_t  *lcf;

    lcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_charset_loc_conf_t));
    if (lcf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     lcf->types = { NULL };
     *     lcf->types_keys = NULL;
     */

    lcf->charset = NGX_CONF_UNSET;
    lcf->source_charset = NGX_CONF_UNSET;
    lcf->override_charset = NGX_CONF_UNSET;

    return lcf;
}


static char *
ngx_http_charset_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_charset_loc_conf_t *prev = parent;
    ngx_http_charset_loc_conf_t *conf = child;

    ngx_uint_t                   i;
    ngx_http_charset_recode_t   *recode;
    ngx_http_charset_main_conf_t  *mcf;

    if (ngx_http_merge_types(cf, &conf->types_keys, &conf->types,
                             &prev->types_keys, &prev->types,
                             ngx_http_charset_default_types)
        != NGX_OK)
```

```
1562      {
1563          return NGX_CONF_ERROR;
1564      }
1565
1566      ngx_conf_merge_value(conf->override_charset, prev->override_charset, 0);
1567      ngx_conf_merge_value(conf->charset, prev->charset, NGX_HTTP_CHARSET_OFF);
1568      ngx_conf_merge_value(conf->source_charset, prev->source_charset,
1569                           NGX_HTTP_CHARSET_OFF);
1570
1571      if (conf->charset == NGX_HTTP_CHARSET_OFF
1572          || conf->source_charset == NGX_HTTP_CHARSET_OFF
1573          || conf->charset == conf->source_charset)
1574      {
1575          return NGX_CONF_OK;
1576      }
1577
1578      if (conf->source_charset >= NGX_HTTP_CHARSET_VAR
1579          || conf->charset >= NGX_HTTP_CHARSET_VAR)
1580      {
1581          return NGX_CONF_OK;
1582      }
1583
1584      mcf = ngx_http_conf_get_module_main_conf(cf,
1585                                               ngx_http_charset_filter_module);
1586      recode = mcf->recodes.elts;
1587      for (i = 0; i < mcf->recodes.nelts; i++) {
1588          if (conf->source_charset == recode[i].src
1589              && conf->charset == recode[i].dst)
1590          {
1591              return NGX_CONF_OK;
1592          }
1593      }
1594
1595      recode = ngx_array_push(&mcf->recodes);
1596      if (recode == NULL) {
1597          return NGX_CONF_ERROR;
1598      }
1599
1600      recode->src = conf->source_charset;
1601      recode->dst = conf->charset;
1602
1603      return NGX_CONF_OK;
1604  }
1605
1606
1607  static ngx_int_t
1608  ngx_http_charset_postconfiguration(ngx_conf_t *cf)
1609  {
1610      u_char                        **src, **dst;
1611      ngx_int_t                       c;
1612      ngx_uint_t                      i, t;
1613      ngx_http_charset_t             *charset;
1614      ngx_http_charset_recode_t      *recode;
1615      ngx_http_charset_tables_t      *tables;
1616      ngx_http_charset_main_conf_t   *mcf;
1617
1618      mcf = ngx_http_conf_get_module_main_conf(cf,
1619                                               ngx_http_charset_filter_module);
1620
1621      recode = mcf->recodes.elts;
1622      tables = mcf->tables.elts;
1623      charset = mcf->charsets.elts;
1624
1625      for (i = 0; i < mcf->recodes.nelts; i++) {
1626
1627          c = recode[i].src;
1628
1629          for (t = 0; t < mcf->tables.nelts; t++) {
1630
1631              if (c == tables[t].src && recode[i].dst == tables[t].dst) {
1632                  goto next;
1633              }
1634
1635              if (c == tables[t].dst && recode[i].dst == tables[t].src) {
1636                  goto next;
1637              }
```

```
1638        }
1639
1640        ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
1641                      "no \"charset_map\" between the charsets \"%V\" and \"%V\"",
1642                      &charset[c].name, &charset[recode[i].dst].name);
1643        return NGX_ERROR;
1644
1645    next:
1646        continue;
1647    }
1648
1649
1650    for (t = 0; t < mcf->tables.nelts; t++) {
1651
1652        src = charset[tables[t].src].tables;
1653
1654        if (src == NULL) {
1655            src = ngx_pcalloc(cf->pool, sizeof(u_char *) * mcf->charsets.nelts);
1656            if (src == NULL) {
1657                return NGX_ERROR;
1658            }
1659
1660            charset[tables[t].src].tables = src;
1661        }
1662
1663        dst = charset[tables[t].dst].tables;
1664
1665        if (dst == NULL) {
1666            dst = ngx_pcalloc(cf->pool, sizeof(u_char *) * mcf->charsets.nelts);
1667            if (dst == NULL) {
1668                return NGX_ERROR;
1669            }
1670
1671            charset[tables[t].dst].tables = dst;
1672        }
1673
1674        src[tables[t].dst] = tables[t].src2dst;
1675        dst[tables[t].src] = tables[t].dst2src;
1676    }
1677
1678    ngx_http_next_header_filter = ngx_http_top_header_filter;
1679    ngx_http_top_header_filter = ngx_http_charset_header_filter;
1680
1681    ngx_http_next_body_filter = ngx_http_top_body_filter;
1682    ngx_http_top_body_filter = ngx_http_charset_body_filter;
1683
1684    return NGX_OK;
1685 }
```

One Level Up Top Level

src/http/modules/ngx_http_chunked_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_chunked_filter_module
- ngx_http_chunked_filter_module_ctx
- ngx_http_next_body_filter
- ngx_http_next_header_filter

Data types defined

- ngx_http_chunked_filter_ctx_t

Functions defined

- ngx_http_chunked_body_filter
- ngx_http_chunked_filter_init
- ngx_http_chunked_header_filter

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_chain_t         *free;
15      ngx_chain_t         *busy;
16  } ngx_http_chunked_filter_ctx_t;
17
18
19  static ngx_int_t ngx_http_chunked_filter_init(ngx_conf_t *cf);
20
21
22  static ngx_http_module_t  ngx_http_chunked_filter_module_ctx = {
23      NULL,                                  /* preconfiguration */
24      ngx_http_chunked_filter_init,          /* postconfiguration */
25
26      NULL,                                  /* create main configuration */
27      NULL,                                  /* init main configuration */
28
29      NULL,                                  /* create server configuration */
30      NULL,                                  /* merge server configuration */
31
32      NULL,                                  /* create location configuration */
33      NULL                                   /* merge location configuration */
34  };
35
36
37  ngx_module_t  ngx_http_chunked_filter_module = {
38      NGX_MODULE_V1,
39      &ngx_http_chunked_filter_module_ctx,   /* module context */
```

```
    NULL,                                  /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static ngx_int_t
ngx_http_chunked_header_filter(ngx_http_request_t *r)
{
    ngx_http_core_loc_conf_t       *clcf;
    ngx_http_chunked_filter_ctx_t  *ctx;

    if (r->headers_out.status == NGX_HTTP_NOT_MODIFIED
        || r->headers_out.status == NGX_HTTP_NO_CONTENT
        || r->headers_out.status < NGX_HTTP_OK
        || r != r->main
        || (r->method & NGX_HTTP_HEAD))
    {
        return ngx_http_next_header_filter(r);
    }

    if (r->headers_out.content_length_n == -1) {
        if (r->http_version < NGX_HTTP_VERSION_11) {
            r->keepalive = 0;

        } else {
            clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

            if (clcf->chunked_transfer_encoding) {
                r->chunked = 1;

                ctx = ngx_pcalloc(r->pool,
                                  sizeof(ngx_http_chunked_filter_ctx_t));
                if (ctx == NULL) {
                    return NGX_ERROR;
                }

                ngx_http_set_ctx(r, ctx, ngx_http_chunked_filter_module);

            } else {
                r->keepalive = 0;
            }
        }
    }

    return ngx_http_next_header_filter(r);
}


 static ngx_int_t
 ngx_http_chunked_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
 {
     u_char                        *chunk;
     off_t                          size;
     ngx_int_t                      rc;
     ngx_buf_t                     *b;
     ngx_chain_t                   *out, *cl, *tl, **ll;
     ngx_http_chunked_filter_ctx_t  *ctx;

     if (in == NULL || !r->chunked || r->header_only) {
         return ngx_http_next_body_filter(r, in);
     }

     ctx = ngx_http_get_module_ctx(r, ngx_http_chunked_filter_module);

```

```
    out = NULL;
    ll = &out;

    size = 0;
    cl = in;

    for ( ;; ) {
        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http chunk: %d", ngx_buf_size(cl->buf));

        size += ngx_buf_size(cl->buf);

        if (cl->buf->flush
            || cl->buf->sync
            || ngx_buf_in_memory(cl->buf)
            || cl->buf->in_file)
        {
            tl = ngx_alloc_chain_link(r->pool);
            if (tl == NULL) {
                return NGX_ERROR;
            }

            tl->buf = cl->buf;
            *ll = tl;
            ll = &tl->next;
        }

        if (cl->next == NULL) {
            break;
        }

        cl = cl->next;
    }

    if (size) {
        tl = ngx_chain_get_free_buf(r->pool, &ctx->free);
        if (tl == NULL) {
            return NGX_ERROR;
        }

        b = tl->buf;
        chunk = b->start;

        if (chunk == NULL) {
            /* the "0000000000000000" is 64-bit hexadecimal string */

            chunk = ngx_palloc(r->pool, sizeof("0000000000000000" CRLF) - 1);
            if (chunk == NULL) {
                return NGX_ERROR;
            }

            b->start = chunk;
            b->end = chunk + sizeof("0000000000000000" CRLF) - 1;
        }

        b->tag = (ngx_buf_tag_t) &ngx_http_chunked_filter_module;
        b->memory = 0;
        b->temporary = 1;
        b->pos = chunk;
        b->last = ngx_sprintf(chunk, "%xO" CRLF, size);

        tl->next = out;
        out = tl;
    }

    if (cl->buf->last_buf) {
        tl = ngx_chain_get_free_buf(r->pool, &ctx->free);
        if (tl == NULL) {
            return NGX_ERROR;
        }

        b = tl->buf;

        b->tag = (ngx_buf_tag_t) &ngx_http_chunked_filter_module;
        b->temporary = 0;
        b->memory = 1;
```

```
192         b->last_buf = 1;
193         b->pos = (u_char *) CRLF "0" CRLF CRLF;
194         b->last = b->pos + 7;
195 
196         cl->buf->last_buf = 0;
197 
198         *ll = tl;
199 
200         if (size == 0) {
201             b->pos += 2;
202         }
203 
204     } else if (size > 0) {
205         tl = ngx_chain_get_free_buf(r->pool, &ctx->free);
206         if (tl == NULL) {
207             return NGX_ERROR;
208         }
209 
210         b = tl->buf;
211 
212         b->tag = (ngx_buf_tag_t) &ngx_http_chunked_filter_module;
213         b->temporary = 0;
214         b->memory = 1;
215         b->pos = (u_char *) CRLF;
216         b->last = b->pos + 2;
217 
218         *ll = tl;
219 
220     } else {
221         *ll = NULL;
222     }
223 
224     rc = ngx_http_next_body_filter(r, out);
225 
226     ngx_chain_update_chains(r->pool, &ctx->free, &ctx->busy, &out,
227                             (ngx_buf_tag_t) &ngx_http_chunked_filter_module);
228 
229     return rc;
230 }
231 
232 
233 static ngx_int_t
234 ngx_http_chunked_filter_init(ngx_conf_t *cf)
235 {
236     ngx_http_next_header_filter = ngx_http_top_header_filter;
237     ngx_http_top_header_filter = ngx_http_chunked_header_filter;
238 
239     ngx_http_next_body_filter = ngx_http_top_body_filter;
240     ngx_http_top_body_filter = ngx_http_chunked_body_filter;
241 
242     return NGX_OK;
243 }
```

One Level Up Top Level

src/http/modules/ngx_http_dav_module.c - nginx-1.7.10

Global variables defined

- ngx_http_dav_commands
- ngx_http_dav_methods_mask
- ngx_http_dav_module
- ngx_http_dav_module_ctx

Data types defined

- ngx_http_dav_copy_ctx_t
- ngx_http_dav_loc_conf_t

Functions defined

- ngx_http_dav_copy_dir
- ngx_http_dav_copy_dir_time
- ngx_http_dav_copy_move_handler
- ngx_http_dav_copy_tree_file
- ngx_http_dav_create_loc_conf
- ngx_http_dav_delete_dir
- ngx_http_dav_delete_file
- ngx_http_dav_delete_handler
- ngx_http_dav_delete_path
- ngx_http_dav_depth
- ngx_http_dav_error
- ngx_http_dav_handler
- ngx_http_dav_init
- ngx_http_dav_location
- ngx_http_dav_merge_loc_conf
- ngx_http_dav_mkcol_handler
- ngx_http_dav_noop
- ngx_http_dav_put_handler

Macros defined

- NGX_HTTP_DAV_INFINITY_DEPTH

- NGX_HTTP_DAV_INVALID_DEPTH
- NGX_HTTP_DAV_NO_DEPTH
- NGX_HTTP_DAV_OFF

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define NGX_HTTP_DAV_OFF             2
14
15
16  #define NGX_HTTP_DAV_NO_DEPTH        -3
17  #define NGX_HTTP_DAV_INVALID_DEPTH   -2
18  #define NGX_HTTP_DAV_INFINITY_DEPTH  -1
19
20
21  typedef struct {
22      ngx_uint_t  methods;
23      ngx_uint_t  access;
24      ngx_uint_t  min_delete_depth;
25      ngx_flag_t  create_full_put_path;
26  } ngx_http_dav_loc_conf_t;
27
28
29  typedef struct {
30      ngx_str_t   path;
31      size_t      len;
32  } ngx_http_dav_copy_ctx_t;
33
34
35  static ngx_int_t ngx_http_dav_handler(ngx_http_request_t *r);
36
37  static void ngx_http_dav_put_handler(ngx_http_request_t *r);
38
39  static ngx_int_t ngx_http_dav_delete_handler(ngx_http_request_t *r);
40  static ngx_int_t ngx_http_dav_delete_path(ngx_http_request_t *r,
41      ngx_str_t *path, ngx_uint_t dir);
42  static ngx_int_t ngx_http_dav_delete_dir(ngx_tree_ctx_t *ctx, ngx_str_t *path);
43  static ngx_int_t ngx_http_dav_delete_file(ngx_tree_ctx_t *ctx, ngx_str_t *path);
44  static ngx_int_t ngx_http_dav_noop(ngx_tree_ctx_t *ctx, ngx_str_t *path);
45
46  static ngx_int_t ngx_http_dav_mkcol_handler(ngx_http_request_t *r,
47      ngx_http_dav_loc_conf_t *dlcf);
48
49  static ngx_int_t ngx_http_dav_copy_move_handler(ngx_http_request_t *r);
50  static ngx_int_t ngx_http_dav_copy_dir(ngx_tree_ctx_t *ctx, ngx_str_t *path);
51  static ngx_int_t ngx_http_dav_copy_dir_time(ngx_tree_ctx_t *ctx,
52      ngx_str_t *path);
53  static ngx_int_t ngx_http_dav_copy_tree_file(ngx_tree_ctx_t *ctx,
54      ngx_str_t *path);
55
56  static ngx_int_t ngx_http_dav_depth(ngx_http_request_t *r, ngx_int_t dflt);
57  static ngx_int_t ngx_http_dav_error(ngx_log_t *log, ngx_err_t err,
58      ngx_int_t not_found, char *failed, u_char *path);
59  static ngx_int_t ngx_http_dav_location(ngx_http_request_t *r, u_char *path);
60  static void *ngx_http_dav_create_loc_conf(ngx_conf_t *cf);
61  static char *ngx_http_dav_merge_loc_conf(ngx_conf_t *cf,
62      void *parent, void *child);
63  static ngx_int_t ngx_http_dav_init(ngx_conf_t *cf);
64
65
```

```
static ngx_conf_bitmask_t  ngx_http_dav_methods_mask[] = {
    { ngx_string("off"), NGX_HTTP_DAV_OFF },
    { ngx_string("put"), NGX_HTTP_PUT },
    { ngx_string("delete"), NGX_HTTP_DELETE },
    { ngx_string("mkcol"), NGX_HTTP_MKCOL },
    { ngx_string("copy"), NGX_HTTP_COPY },
    { ngx_string("move"), NGX_HTTP_MOVE },
    { ngx_null_string, 0 }
};


static ngx_command_t  ngx_http_dav_commands[] = {

    { ngx_string("dav_methods"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_dav_loc_conf_t, methods),
      &ngx_http_dav_methods_mask },

    { ngx_string("create_full_put_path"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_dav_loc_conf_t, create_full_put_path),
      NULL },

    { ngx_string("min_delete_depth"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_dav_loc_conf_t, min_delete_depth),
      NULL },

     { ngx_string("dav_access"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
       ngx_conf_set_access_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_dav_loc_conf_t, access),
       NULL },

       ngx_null_command
};


static ngx_http_module_t  ngx_http_dav_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_dav_init,                     /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_dav_create_loc_conf,          /* create location configuration */
    ngx_http_dav_merge_loc_conf            /* merge location configuration */
};


ngx_module_t  ngx_http_dav_module = {
    NGX_MODULE_V1,
    &ngx_http_dav_module_ctx,              /* module context */
    ngx_http_dav_commands,                 /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};
```

```
static ngx_int_t
ngx_http_dav_handler(ngx_http_request_t *r)
{
    ngx_int_t                 rc;
    ngx_http_dav_loc_conf_t  *dlcf;

    dlcf = ngx_http_get_module_loc_conf(r, ngx_http_dav_module);

    if (!(r->method & dlcf->methods)) {
        return NGX_DECLINED;
    }

    switch (r->method) {

    case NGX_HTTP_PUT:

        if (r->uri.data[r->uri.len - 1] == '/') {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "cannot PUT to a collection");
            return NGX_HTTP_CONFLICT;
        }

        r->request_body_in_file_only = 1;
        r->request_body_in_persistent_file = 1;
        r->request_body_in_clean_file = 1;
        r->request_body_file_group_access = 1;
        r->request_body_file_log_level = 0;

        rc = ngx_http_read_client_request_body(r, ngx_http_dav_put_handler);

        if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
            return rc;
        }

        return NGX_DONE;

    case NGX_HTTP_DELETE:

        return ngx_http_dav_delete_handler(r);

    case NGX_HTTP_MKCOL:

        return ngx_http_dav_mkcol_handler(r, dlcf);

    case NGX_HTTP_COPY:

        return ngx_http_dav_copy_move_handler(r);

    case NGX_HTTP_MOVE:

        return ngx_http_dav_copy_move_handler(r);
    }

    return NGX_DECLINED;
}


static void
ngx_http_dav_put_handler(ngx_http_request_t *r)
{
    size_t                    root;
    time_t                    date;
    ngx_str_t                *temp, path;
    ngx_uint_t                status;
    ngx_file_info_t           fi;
    ngx_ext_rename_file_t     ext;
    ngx_http_dav_loc_conf_t  *dlcf;

    if (r->request_body == NULL || r->request_body->temp_file == NULL) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    if (ngx_http_map_uri_to_path(r, &path, &root, 0) == NULL) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
```

```
    }

    path.len--;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http put filename: \"%s\"", path.data);

    temp = &r->request_body->temp_file->file.name;

    if (ngx_file_info(path.data, &fi) == NGX_FILE_ERROR) {
        status = NGX_HTTP_CREATED;

    } else {
        status = NGX_HTTP_NO_CONTENT;

        if (ngx_is_dir(&fi)) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, NGX_EISDIR,
                          "\"%s\" could not be created", path.data);

            if (ngx_delete_file(temp->data) == NGX_FILE_ERROR) {
                ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
                              ngx_delete_file_n " \"%s\" failed",
                              temp->data);
            }

            ngx_http_finalize_request(r, NGX_HTTP_CONFLICT);
            return;
        }
    }

    dlcf = ngx_http_get_module_loc_conf(r, ngx_http_dav_module);

    ext.access = dlcf->access;
    ext.path_access = dlcf->access;
    ext.time = -1;
    ext.create_path = dlcf->create_full_put_path;
    ext.delete_file = 1;
    ext.log = r->connection->log;

    if (r->headers_in.date) {
        date = ngx_http_parse_time(r->headers_in.date->value.data,
                                   r->headers_in.date->value.len);

        if (date != NGX_ERROR) {
            ext.time = date;
            ext.fd = r->request_body->temp_file->file.fd;
        }
    }

    if (ngx_ext_rename_file(temp, &path, &ext) != NGX_OK) {
        ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
        return;
    }

    if (status == NGX_HTTP_CREATED) {
        if (ngx_http_dav_location(r, path.data) != NGX_OK) {
            ngx_http_finalize_request(r, NGX_HTTP_INTERNAL_SERVER_ERROR);
            return;
        }

        r->headers_out.content_length_n = 0;
    }

    r->headers_out.status = status;
    r->header_only = 1;

    ngx_http_finalize_request(r, ngx_http_send_header(r));
    return;
}


static ngx_int_t
ngx_http_dav_delete_handler(ngx_http_request_t *r)
{
    size_t                     root;
    ngx_err_t                  err;
```

```
    ngx_int_t                 rc, depth;
    ngx_uint_t                i, d, dir;
    ngx_str_t                 path;
    ngx_file_info_t           fi;
    ngx_http_dav_loc_conf_t  *dlcf;

    if (r->headers_in.content_length_n > 0) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "DELETE with body is unsupported");
        return NGX_HTTP_UNSUPPORTED_MEDIA_TYPE;
    }

    dlcf = ngx_http_get_module_loc_conf(r, ngx_http_dav_module);

    if (dlcf->min_delete_depth) {
        d = 0;

        for (i = 0; i < r->uri.len; /* void */) {
            if (r->uri.data[i++] == '/') {
                if (++d >= dlcf->min_delete_depth && i < r->uri.len) {
                    goto ok;
                }
            }
        }

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "insufficient URI depth:%i to DELETE", d);
        return NGX_HTTP_CONFLICT;
    }

ok:

    if (ngx_http_map_uri_to_path(r, &path, &root, 0) == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http delete filename: \"%s\"", path.data);

    if (ngx_link_info(path.data, &fi) == NGX_FILE_ERROR) {
        err = ngx_errno;

        rc = (err == NGX_ENOTDIR) ? NGX_HTTP_CONFLICT : NGX_HTTP_NOT_FOUND;

        return ngx_http_dav_error(r->connection->log, err,
                                  rc, ngx_link_info_n, path.data);
    }

    if (ngx_is_dir(&fi)) {

        if (r->uri.data[r->uri.len - 1] != '/') {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, NGX_EISDIR,
                          "DELETE \"%s\" failed", path.data);
            return NGX_HTTP_CONFLICT;
        }

        depth = ngx_http_dav_depth(r, NGX_HTTP_DAV_INFINITY_DEPTH);

        if (depth != NGX_HTTP_DAV_INFINITY_DEPTH) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "\"Depth\" header must be infinity");
            return NGX_HTTP_BAD_REQUEST;
        }

        path.len -= 2;  /* omit "/\0" */

        dir = 1;

    } else {

        /*
         * we do not need to test (r->uri.data[r->uri.len - 1] == '/')
         * because ngx_link_info("/file/") returned NGX_ENOTDIR above
         */

        depth = ngx_http_dav_depth(r, 0);
```

```
370
371         if (depth != 0 && depth != NGX_HTTP_DAV_INFINITY_DEPTH) {
372             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
373                           "\"Depth\" header must be 0 or infinity");
374             return NGX_HTTP_BAD_REQUEST;
375         }
376
377         dir = 0;
378     }
379
380     rc = ngx_http_dav_delete_path(r, &path, dir);
381
382     if (rc == NGX_OK) {
383         return NGX_HTTP_NO_CONTENT;
384     }
385
386     return rc;
387 }
388
389
390 static ngx_int_t
391 ngx_http_dav_delete_path(ngx_http_request_t *r, ngx_str_t *path, ngx_uint_t dir)
392 {
393     char            *failed;
394     ngx_tree_ctx_t   tree;
395
396     if (dir) {
397
398         tree.init_handler = NULL;
399         tree.file_handler = ngx_http_dav_delete_file;
400         tree.pre_tree_handler = ngx_http_dav_noop;
401         tree.post_tree_handler = ngx_http_dav_delete_dir;
402         tree.spec_handler = ngx_http_dav_delete_file;
403         tree.data = NULL;
404         tree.alloc = 0;
405         tree.log = r->connection->log;
406
407         /* TODO: 207 */
408
409         if (ngx_walk_tree(&tree, path) != NGX_OK) {
410             return NGX_HTTP_INTERNAL_SERVER_ERROR;
411         }
412
413         if (ngx_delete_dir(path->data) != NGX_FILE_ERROR) {
414             return NGX_OK;
415         }
416
417         failed = ngx_delete_dir_n;
418
419     } else {
420
421         if (ngx_delete_file(path->data) != NGX_FILE_ERROR) {
422             return NGX_OK;
423         }
424
425         failed = ngx_delete_file_n;
426     }
427
428     return ngx_http_dav_error(r->connection->log, ngx_errno,
429                               NGX_HTTP_NOT_FOUND, failed, path->data);
430 }
431
432
433 static ngx_int_t
434 ngx_http_dav_delete_dir(ngx_tree_ctx_t *ctx, ngx_str_t *path)
435 {
436     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
437                    "http delete dir: \"%s\"", path->data);
438
439     if (ngx_delete_dir(path->data) == NGX_FILE_ERROR) {
440
441         /* TODO: add to 207 */
442
443         (void) ngx_http_dav_error(ctx->log, ngx_errno, 0, ngx_delete_dir_n,
444                                   path->data);
445     }
```

```
446
447     return NGX_OK;
448 }
449
450
451 static ngx_int_t
452 ngx_http_dav_delete_file(ngx_tree_ctx_t *ctx, ngx_str_t *path)
453 {
454     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
455                    "http delete file: \"%s\"", path->data);
456
457     if (ngx_delete_file(path->data) == NGX_FILE_ERROR) {
458
459         /* TODO: add to 207 */
460
461         (void) ngx_http_dav_error(ctx->log, ngx_errno, 0, ngx_delete_file_n,
462                                   path->data);
463     }
464
465     return NGX_OK;
466 }
467
468
469 static ngx_int_t
470 ngx_http_dav_noop(ngx_tree_ctx_t *ctx, ngx_str_t *path)
471 {
472     return NGX_OK;
473 }
474
475
476 static ngx_int_t
477 ngx_http_dav_mkcol_handler(ngx_http_request_t *r, ngx_http_dav_loc_conf_t *dlcf)
478 {
479     u_char    *p;
480     size_t     root;
481     ngx_str_t  path;
482
483     if (r->headers_in.content_length_n > 0) {
484         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
485                       "MKCOL with body is unsupported");
486         return NGX_HTTP_UNSUPPORTED_MEDIA_TYPE;
487     }
488
489     if (r->uri.data[r->uri.len - 1] != '/') {
490         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
491                       "MKCOL can create a collection only");
492         return NGX_HTTP_CONFLICT;
493     }
494
495     p = ngx_http_map_uri_to_path(r, &path, &root, 0);
496     if (p == NULL) {
497         return NGX_HTTP_INTERNAL_SERVER_ERROR;
498     }
499
500     *(p - 1) = '\0';
501     r->uri.len--;
502
503     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
504                    "http mkcol path: \"%s\"", path.data);
505
506     if (ngx_create_dir(path.data, ngx_dir_access(dlcf->access))
507         != NGX_FILE_ERROR)
508     {
509         if (ngx_http_dav_location(r, path.data) != NGX_OK) {
510             return NGX_HTTP_INTERNAL_SERVER_ERROR;
511         }
512
513         return NGX_HTTP_CREATED;
514     }
515
516     return ngx_http_dav_error(r->connection->log, ngx_errno,
517                               NGX_HTTP_CONFLICT, ngx_create_dir_n, path.data);
518 }
519
520
521 static ngx_int_t
```

```
ngx_http_dav_copy_move_handler(ngx_http_request_t *r)
{
    u_char                   *p, *host, *last, ch;
    size_t                    len, root;
    ngx_err_t                 err;
    ngx_int_t                 rc, depth;
    ngx_uint_t                overwrite, slash, dir, flags;
    ngx_str_t                 path, uri, duri, args;
    ngx_tree_ctx_t            tree;
    ngx_copy_file_t           cf;
    ngx_file_info_t           fi;
    ngx_table_elt_t          *dest, *over;
    ngx_ext_rename_file_t     ext;
    ngx_http_dav_copy_ctx_t   copy;
    ngx_http_dav_loc_conf_t  *dlcf;

    if (r->headers_in.content_length_n > 0) {
        return NGX_HTTP_UNSUPPORTED_MEDIA_TYPE;
    }

    dest = r->headers_in.destination;

    if (dest == NULL) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "client sent no \"Destination\" header");
        return NGX_HTTP_BAD_REQUEST;
    }

    p = dest->value.data;
    /* there is always '\0' even after empty header value */
    if (p[0] == '/') {
        last = p + dest->value.len;
        goto destination_done;
    }

    len = r->headers_in.server.len;

    if (len == 0) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "client sent no \"Host\" header");
        return NGX_HTTP_BAD_REQUEST;
    }

#if (NGX_HTTP_SSL)

    if (r->connection->ssl) {
        if (ngx_strncmp(dest->value.data, "https://", sizeof("https://") - 1)
            != 0)
        {
            goto invalid_destination;
        }

        host = dest->value.data + sizeof("https://") - 1;

    } else
#endif
    {
        if (ngx_strncmp(dest->value.data, "http://", sizeof("http://") - 1)
            != 0)
        {
            goto invalid_destination;
        }

        host = dest->value.data + sizeof("http://") - 1;
    }

    if (ngx_strncmp(host, r->headers_in.server.data, len) != 0) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "\"Destination\" URI \"%V\" is handled by "
                      "different repository than the source URI",
                      &dest->value);
        return NGX_HTTP_BAD_REQUEST;
    }

    last = dest->value.data + dest->value.len;

```

```
    for (p = host + len; p < last; p++) {
        if (*p == '/') {
            goto destination_done;
        }
    }

invalid_destination:

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "client sent invalid \"Destination\" header: \"%V\"",
                  &dest->value);
    return NGX_HTTP_BAD_REQUEST;

destination_done:

    duri.len = last - p;
    duri.data = p;
    flags = NGX_HTTP_LOG_UNSAFE;

    if (ngx_http_parse_unsafe_uri(r, &duri, &args, &flags) != NGX_OK) {
        goto invalid_destination;
    }

    if ((r->uri.data[r->uri.len - 1] == '/' && *(last - 1) != '/')
        || (r->uri.data[r->uri.len - 1] != '/' && *(last - 1) == '/'))
    {
         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                       "both URI \"%V\" and \"Destination\" URI \"%V\" "
                       "should be either collections or non-collections",
                       &r->uri, &dest->value);
         return NGX_HTTP_CONFLICT;
    }

    depth = ngx_http_dav_depth(r, NGX_HTTP_DAV_INFINITY_DEPTH);

    if (depth != NGX_HTTP_DAV_INFINITY_DEPTH) {

        if (r->method == NGX_HTTP_COPY) {
            if (depth != 0) {
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "\"Depth\" header must be 0 or infinity");
                return NGX_HTTP_BAD_REQUEST;
            }

        } else {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "\"Depth\" header must be infinity");
            return NGX_HTTP_BAD_REQUEST;
        }
    }

    over = r->headers_in.overwrite;

    if (over) {
        if (over->value.len == 1) {
            ch = over->value.data[0];

            if (ch == 'T' || ch == 't') {
                overwrite = 1;
                goto overwrite_done;
            }

            if (ch == 'F' || ch == 'f') {
                overwrite = 0;
                goto overwrite_done;
            }

        }

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "client sent invalid \"Overwrite\" header: \"%V\"",
                      &over->value);
        return NGX_HTTP_BAD_REQUEST;
    }

    overwrite = 1;
```

```
674
675 overwrite_done:
676
677     if (ngx_http_map_uri_to_path(r, &path, &root, 0) == NULL) {
678         return NGX_HTTP_INTERNAL_SERVER_ERROR;
679     }
680
681     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
682                    "http copy from: \"%s\"", path.data);
683
684     uri = r->uri;
685     r->uri = duri;
686
687     if (ngx_http_map_uri_to_path(r, &copy.path, &root, 0) == NULL) {
688         return NGX_HTTP_INTERNAL_SERVER_ERROR;
689     }
690
691     r->uri = uri;
692
693     copy.path.len--;  /* omit "\0" */
694
695     if (copy.path.data[copy.path.len - 1] == '/') {
696         slash = 1;
697         copy.path.len--;
698         copy.path.data[copy.path.len] = '\0';
699
700     } else {
701         slash = 0;
702     }
703
704     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
705                    "http copy to: \"%s\"", copy.path.data);
706
707     if (ngx_link_info(copy.path.data, &fi) == NGX_FILE_ERROR) {
708         err = ngx_errno;
709
710         if (err != NGX_ENOENT) {
711             return ngx_http_dav_error(r->connection->log, err,
712                                       NGX_HTTP_NOT_FOUND, ngx_link_info_n,
713                                       copy.path.data);
714         }
715
716         /* destination does not exist */
717
718         overwrite = 0;
719         dir = 0;
720
721     } else {
722
723         /* destination exists */
724
725         if (ngx_is_dir(&fi) && !slash) {
726             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
727                           "\"%V\" could not be %Ved to collection \"%V\"",
728                           &r->uri, &r->method_name, &dest->value);
729             return NGX_HTTP_CONFLICT;
730         }
731
732         if (!overwrite) {
733             ngx_log_error(NGX_LOG_ERR, r->connection->log, NGX_EEXIST,
734                           "\"%s\" could not be created", copy.path.data);
735             return NGX_HTTP_PRECONDITION_FAILED;
736         }
737
738         dir = ngx_is_dir(&fi);
739     }
740
741     if (ngx_link_info(path.data, &fi) == NGX_FILE_ERROR) {
742         return ngx_http_dav_error(r->connection->log, ngx_errno,
743                                   NGX_HTTP_NOT_FOUND, ngx_link_info_n,
744                                   path.data);
745     }
746
747     if (ngx_is_dir(&fi)) {
748
749         if (r->uri.data[r->uri.len - 1] != '/') {
```

```
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "\"%V\" is collection", &r->uri);
            return NGX_HTTP_BAD_REQUEST;
        }

        if (overwrite) {
            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http delete: \"%s\"", copy.path.data);

            rc = ngx_http_dav_delete_path(r, &copy.path, dir);

            if (rc != NGX_OK) {
                return rc;
            }
        }
    }

    if (ngx_is_dir(&fi)) {

        path.len -= 2;  /* omit "/\0" */

        if (r->method == NGX_HTTP_MOVE) {
            if (ngx_rename_file(path.data, copy.path.data) != NGX_FILE_ERROR) {
                return NGX_HTTP_CREATED;
            }
        }

        if (ngx_create_dir(copy.path.data, ngx_file_access(&fi))
            == NGX_FILE_ERROR)
        {
            return ngx_http_dav_error(r->connection->log, ngx_errno,
                                      NGX_HTTP_NOT_FOUND,
                                      ngx_create_dir_n, copy.path.data);
        }

        copy.len = path.len;

        tree.init_handler = NULL;
        tree.file_handler = ngx_http_dav_copy_tree_file;
        tree.pre_tree_handler = ngx_http_dav_copy_dir;
        tree.post_tree_handler = ngx_http_dav_copy_dir_time;
        tree.spec_handler = ngx_http_dav_noop;
        tree.data = &copy;
        tree.alloc = 0;
        tree.log = r->connection->log;

        if (ngx_walk_tree(&tree, &path) == NGX_OK) {

            if (r->method == NGX_HTTP_MOVE) {
                rc = ngx_http_dav_delete_path(r, &path, 1);

                if (rc != NGX_OK) {
                    return rc;
                }
            }

            return NGX_HTTP_CREATED;
        }

    } else {

        if (r->method == NGX_HTTP_MOVE) {

            dlcf = ngx_http_get_module_loc_conf(r, ngx_http_dav_module);

            ext.access = 0;
            ext.path_access = dlcf->access;
            ext.time = -1;
            ext.create_path = 1;
            ext.delete_file = 0;
            ext.log = r->connection->log;

            if (ngx_ext_rename_file(&path, &copy.path, &ext) == NGX_OK) {
                return NGX_HTTP_NO_CONTENT;
            }

```

```
826             return NGX_HTTP_INTERNAL_SERVER_ERROR;
827         }
828
829         dlcf = ngx_http_get_module_loc_conf(r, ngx_http_dav_module);
830
831         cf.size = ngx_file_size(&fi);
832         cf.buf_size = 0;
833         cf.access = dlcf->access;
834         cf.time = ngx_file_mtime(&fi);
835         cf.log = r->connection->log;
836
837         if (ngx_copy_file(path.data, copy.path.data, &cf) == NGX_OK) {
838             return NGX_HTTP_NO_CONTENT;
839         }
840     }
841
842     return NGX_HTTP_INTERNAL_SERVER_ERROR;
843 }
844
845
846 static ngx_int_t
847 ngx_http_dav_copy_dir(ngx_tree_ctx_t *ctx, ngx_str_t *path)
848 {
849     u_char                   *p, *dir;
850     size_t                    len;
851     ngx_http_dav_copy_ctx_t  *copy;
852
853     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
854                    "http copy dir: \"%s\"", path->data);
855
856     copy = ctx->data;
857
858     len = copy->path.len + path->len;
859
860     dir = ngx_alloc(len + 1, ctx->log);
861     if (dir == NULL) {
862         return NGX_ABORT;
863     }
864
865     p = ngx_cpymem(dir, copy->path.data, copy->path.len);
866     (void) ngx_cpystrn(p, path->data + copy->len, path->len - copy->len + 1);
867
868     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
869                    "http copy dir to: \"%s\"", dir);
870
871     if (ngx_create_dir(dir, ngx_dir_access(ctx->access)) == NGX_FILE_ERROR) {
872         (void) ngx_http_dav_error(ctx->log, ngx_errno, 0, ngx_create_dir_n,
873                                   dir);
874     }
875
876     ngx_free(dir);
877
878     return NGX_OK;
879 }
880
881
882 static ngx_int_t
883 ngx_http_dav_copy_dir_time(ngx_tree_ctx_t *ctx, ngx_str_t *path)
884 {
885     u_char                   *p, *dir;
886     size_t                    len;
887     ngx_http_dav_copy_ctx_t  *copy;
888
889     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
890                    "http copy dir time: \"%s\"", path->data);
891
892     copy = ctx->data;
893
894     len = copy->path.len + path->len;
895
896     dir = ngx_alloc(len + 1, ctx->log);
897     if (dir == NULL) {
898         return NGX_ABORT;
899     }
900
901     p = ngx_cpymem(dir, copy->path.data, copy->path.len);
```

```
    (void) ngx_cpystrn(p, path->data + copy->len, path->len - copy->len + 1);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
                   "http copy dir time to: \"%s\"", dir);

#if (NGX_WIN32)
    {
    ngx_fd_t  fd;

    fd = ngx_open_file(dir, NGX_FILE_RDWR, NGX_FILE_OPEN, 0);

    if (fd == NGX_INVALID_FILE) {
        (void) ngx_http_dav_error(ctx->log, ngx_errno, 0, ngx_open_file_n, dir);
        goto failed;
    }

    if (ngx_set_file_time(NULL, fd, ctx->mtime) != NGX_OK) {
        ngx_log_error(NGX_LOG_ALERT, ctx->log, ngx_errno,
                      ngx_set_file_time_n " \"%s\" failed", dir);
    }

    if (ngx_close_file(fd) == NGX_FILE_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, ctx->log, ngx_errno,
                      ngx_close_file_n " \"%s\" failed", dir);
    }
    }

failed:

#else

    if (ngx_set_file_time(dir, 0, ctx->mtime) != NGX_OK) {
        ngx_log_error(NGX_LOG_ALERT, ctx->log, ngx_errno,
                      ngx_set_file_time_n " \"%s\" failed", dir);
    }

#endif

    ngx_free(dir);

    return NGX_OK;
}


static ngx_int_t
ngx_http_dav_copy_tree_file(ngx_tree_ctx_t *ctx, ngx_str_t *path)
{
    u_char                   *p, *file;
    size_t                    len;
    ngx_copy_file_t           cf;
    ngx_http_dav_copy_ctx_t  *copy;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
                   "http copy file: \"%s\"", path->data);

    copy = ctx->data;

    len = copy->path.len + path->len;

    file = ngx_alloc(len + 1, ctx->log);
    if (file == NULL) {
        return NGX_ABORT;
    }

    p = ngx_cpymem(file, copy->path.data, copy->path.len);
    (void) ngx_cpystrn(p, path->data + copy->len, path->len - copy->len + 1);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->log, 0,
                   "http copy file to: \"%s\"", file);

    cf.size = ctx->size;
    cf.buf_size = 0;
    cf.access = ctx->access;
    cf.time = ctx->mtime;
    cf.log = ctx->log;

```

```
978      (void) ngx_copy_file(path->data, file, &cf);
979
980      ngx_free(file);
981
982      return NGX_OK;
983  }
984
985
986  static ngx_int_t
987  ngx_http_dav_depth(ngx_http_request_t *r, ngx_int_t dflt)
988  {
989      ngx_table_elt_t  *depth;
990
991      depth = r->headers_in.depth;
992
993      if (depth == NULL) {
994          return dflt;
995      }
996
997      if (depth->value.len == 1) {
998
999          if (depth->value.data[0] == '0') {
1000             return 0;
1001         }
1002
1003         if (depth->value.data[0] == '1') {
1004             return 1;
1005         }
1006
1007     } else {
1008
1009         if (depth->value.len == sizeof("infinity") - 1
1010             && ngx_strcmp(depth->value.data, "infinity") == 0)
1011         {
1012             return NGX_HTTP_DAV_INFINITY_DEPTH;
1013         }
1014     }
1015
1016     ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1017                   "client sent invalid \"Depth\" header: \"%V\"",
1018                   &depth->value);
1019
1020     return NGX_HTTP_DAV_INVALID_DEPTH;
1021 }
1022
1023
1024 static ngx_int_t
1025 ngx_http_dav_error(ngx_log_t *log, ngx_err_t err, ngx_int_t not_found,
1026     char *failed, u_char *path)
1027 {
1028     ngx_int_t   rc;
1029     ngx_uint_t  level;
1030
1031     if (err == NGX_ENOENT || err == NGX_ENOTDIR || err == NGX_ENAMETOOLONG) {
1032         level = NGX_LOG_ERR;
1033         rc = not_found;
1034
1035     } else if (err == NGX_EACCES || err == NGX_EPERM) {
1036         level = NGX_LOG_ERR;
1037         rc = NGX_HTTP_FORBIDDEN;
1038
1039     } else if (err == NGX_EEXIST) {
1040         level = NGX_LOG_ERR;
1041         rc = NGX_HTTP_NOT_ALLOWED;
1042
1043     } else if (err == NGX_ENOSPC) {
1044         level = NGX_LOG_CRIT;
1045         rc = NGX_HTTP_INSUFFICIENT_STORAGE;
1046
1047     } else {
1048         level = NGX_LOG_CRIT;
1049         rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
1050     }
1051
1052     ngx_log_error(level, log, err, "%s \"%s\" failed", failed, path);
1053
```

```
    return rc;
}


static ngx_int_t
ngx_http_dav_location(ngx_http_request_t *r, u_char *path)
{
    u_char                    *location;
    ngx_http_core_loc_conf_t  *clcf;

    r->headers_out.location = ngx_palloc(r->pool, sizeof(ngx_table_elt_t));
    if (r->headers_out.location == NULL) {
        return NGX_ERROR;
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (!clcf->alias && clcf->root_lengths == NULL) {
        location = path + clcf->root.len;

    } else {
        location = ngx_pnalloc(r->pool, r->uri.len);
        if (location == NULL) {
            return NGX_ERROR;
        }

        ngx_memcpy(location, r->uri.data, r->uri.len);
    }

    /*
     * we do not need to set the r->headers_out.location->hash and
     * r->headers_out.location->key fields
     */

    r->headers_out.location->value.len = r->uri.len;
    r->headers_out.location->value.data = location;

    return NGX_OK;
}


static void *
ngx_http_dav_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_dav_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_dav_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->methods = 0;
     */

    conf->min_delete_depth = NGX_CONF_UNSET_UINT;
    conf->access = NGX_CONF_UNSET_UINT;
    conf->create_full_put_path = NGX_CONF_UNSET;

    return conf;
}


static char *
ngx_http_dav_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_dav_loc_conf_t  *prev = parent;
    ngx_http_dav_loc_conf_t  *conf = child;

    ngx_conf_merge_bitmask_value(conf->methods, prev->methods,
                         (NGX_CONF_BITMASK_SET|NGX_HTTP_DAV_OFF));

    ngx_conf_merge_uint_value(conf->min_delete_depth,
                         prev->min_delete_depth, 0);
```

```
1130
1131        ngx_conf_merge_uint_value(conf->access, prev->access, 0600);
1132
1133        ngx_conf_merge_value(conf->create_full_put_path,
1134                             prev->create_full_put_path, 0);
1135
1136        return NGX_CONF_OK;
1137    }
1138
1139
1140    static ngx_int_t
1141    ngx_http_dav_init(ngx_conf_t *cf)
1142    {
1143        ngx_http_handler_pt        *h;
1144        ngx_http_core_main_conf_t  *cmcf;
1145
1146        cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
1147
1148        h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
1149        if (h == NULL) {
1150            return NGX_ERROR;
1151        }
1152
1153        *h = ngx_http_dav_handler;
1154
1155        return NGX_OK;
1156    }
```

One Level Up Top Level

src/http/modules/ngx_http_empty_gif_module.c - nginx-1.7.10

Global variables defined

- ngx_empty_gif
- ngx_http_empty_gif_commands
- ngx_http_empty_gif_module
- ngx_http_empty_gif_module_ctx
- ngx_http_gif_type

Functions defined

- ngx_http_empty_gif
- ngx_http_empty_gif_handler

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9  #include <ngx_http.h>
10
11
12 static char *ngx_http_empty_gif(ngx_conf_t *cf, ngx_command_t *cmd,
13     void *conf);
14
15 static ngx_command_t  ngx_http_empty_gif_commands[] = {
16
17     { ngx_string("empty_gif"),
18       NGX_HTTP_LOC_CONF|NGX_CONF_NOARGS,
19       ngx_http_empty_gif,
20       0,
21       0,
22       NULL },
23
24       ngx_null_command
25 };
26
27
28 /* the minimal single pixel transparent GIF, 43 bytes */
29
30 static u_char  ngx_empty_gif[] = {
31
32     'G', 'I', 'F', '8', '9', 'a',  /* header                               */
33
34                                    /* logical screen descriptor            */
35     0x01, 0x00,                    /* logical screen width                 */
36     0x01, 0x00,                    /* logical screen height                */
37     0x80,                          /* global 1-bit color table             */
38     0x01,                          /* background color #1                  */
39     0x00,                          /* no aspect ratio                      */
40
41                                    /* global color table                   */
42     0x00, 0x00, 0x00,              /* #0: black                            */
43     0xff, 0xff, 0xff,              /* #1: white                            */
44
45                                    /* graphic control extension            */
```

```
    0x21,                           /* extension introducer                    */
    0xf9,                           /* graphic control label                   */
    0x04,                           /* block size                              */
    0x01,                           /* transparent color is given,             */
                                    /*     no disposal specified,              */
                                    /*     user input is not expected          */
    0x00, 0x00,                     /* delay time                              */
    0x01,                           /* transparent color #1                    */
    0x00,                           /* block terminator                        */

                                    /* image descriptor                        */
    0x2c,                           /* image separator                         */
    0x00, 0x00,                     /* image left position                     */
    0x00, 0x00,                     /* image top position                      */
    0x01, 0x00,                     /* image width                             */
    0x01, 0x00,                     /* image height                            */
    0x00,                           /* no local color table, no interlaced     */

                                    /* table based image data                  */
    0x02,                           /* LZW minimum code size,                  */
                                    /*     must be at least 2-bit              */
    0x02,                           /* block size                              */
    0x4c, 0x01,                     /* compressed bytes 01_001_100, 0000000_1  */
                                    /* 100: clear code                         */
                                    /* 001: 1                                  */
                                    /* 101: end of information code            */
    0x00,                           /* block terminator                        */

    0x3B                            /* trailer                                 */
};


static ngx_http_module_t  ngx_http_empty_gif_module_ctx = {
    NULL,                           /* preconfiguration */
    NULL,                           /* postconfiguration */

    NULL,                           /* create main configuration */
    NULL,                           /* init main configuration */

    NULL,                           /* create server configuration */
    NULL,                           /* merge server configuration */

    NULL,                           /* create location configuration */
    NULL                            /* merge location configuration */
};


ngx_module_t  ngx_http_empty_gif_module = {
    NGX_MODULE_V1,
    &ngx_http_empty_gif_module_ctx, /* module context */
    ngx_http_empty_gif_commands,    /* module directives */
    NGX_HTTP_MODULE,                /* module type */
    NULL,                           /* init master */
    NULL,                           /* init module */
     NULL,                           /* init process */
     NULL,                           /* init thread */
     NULL,                           /* exit thread */
     NULL,                           /* exit process */
     NULL,                           /* exit master */
     NGX_MODULE_V1_PADDING
};


static ngx_str_t  ngx_http_gif_type = ngx_string("image/gif");


static ngx_int_t
ngx_http_empty_gif_handler(ngx_http_request_t *r)
{
    ngx_http_complex_value_t  cv;

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD))) {
        return NGX_HTTP_NOT_ALLOWED;
    }

    ngx_memzero(&cv, sizeof(ngx_http_complex_value_t));
```

```
122
123     cv.value.len = sizeof(ngx_empty_gif);
124     cv.value.data = ngx_empty_gif;
125     r->headers_out.last_modified_time = 23349600;
126
127     return ngx_http_send_response(r, NGX_HTTP_OK, &ngx_http_gif_type, &cv);
128 }
129
130
131 static char *
132 ngx_http_empty_gif(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
133 {
134     ngx_http_core_loc_conf_t  *clcf;
135
136     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
137     clcf->handler = ngx_http_empty_gif_handler;
138
139     return NGX_CONF_OK;
140 }
```

One Level Up Top Level

src/http/modules/ngx_http_fastcgi_module.c - nginx-1.7.10

Global variables defined

- ngx_http_fastcgi_cache_headers
- ngx_http_fastcgi_commands
- ngx_http_fastcgi_hide_headers
- ngx_http_fastcgi_lowat_post
- ngx_http_fastcgi_module
- ngx_http_fastcgi_module
- ngx_http_fastcgi_module_ctx
- ngx_http_fastcgi_next_upstream_masks
- ngx_http_fastcgi_request_start
- ngx_http_fastcgi_temp_path
- ngx_http_fastcgi_vars

Data types defined

- ngx_http_fastcgi_begin_request_t
- ngx_http_fastcgi_ctx_t
- ngx_http_fastcgi_header_small_t
- ngx_http_fastcgi_header_t
- ngx_http_fastcgi_loc_conf_t
- ngx_http_fastcgi_main_conf_t
- ngx_http_fastcgi_params_t
- ngx_http_fastcgi_request_start_t
- ngx_http_fastcgi_split_part_t
- ngx_http_fastcgi_state_e

Functions defined

- ngx_http_fastcgi_abort_request
- ngx_http_fastcgi_add_variables
- ngx_http_fastcgi_cache
- ngx_http_fastcgi_cache_key
- ngx_http_fastcgi_create_key
- ngx_http_fastcgi_create_loc_conf

- ngx_http_fastcgi_create_main_conf
- ngx_http_fastcgi_create_request
- ngx_http_fastcgi_eval
- ngx_http_fastcgi_finalize_request
- ngx_http_fastcgi_handler
- ngx_http_fastcgi_init_params
- ngx_http_fastcgi_input_filter
- ngx_http_fastcgi_input_filter_init
- ngx_http_fastcgi_lowat_check
- ngx_http_fastcgi_merge_loc_conf
- ngx_http_fastcgi_non_buffered_filter
- ngx_http_fastcgi_pass
- ngx_http_fastcgi_path_info_variable
- ngx_http_fastcgi_process_header
- ngx_http_fastcgi_process_record
- ngx_http_fastcgi_reinit_request
- ngx_http_fastcgi_script_name_variable
- ngx_http_fastcgi_split
- ngx_http_fastcgi_split_path_info
- ngx_http_fastcgi_store

Macros defined

- NGX_HTTP_FASTCGI_ABORT_REQUEST
- NGX_HTTP_FASTCGI_BEGIN_REQUEST
- NGX_HTTP_FASTCGI_DATA
- NGX_HTTP_FASTCGI_END_REQUEST
- NGX_HTTP_FASTCGI_KEEP_CONN
- NGX_HTTP_FASTCGI_PARAMS
- NGX_HTTP_FASTCGI_RESPONDER
- NGX_HTTP_FASTCGI_STDERR
- NGX_HTTP_FASTCGI_STDIN
- NGX_HTTP_FASTCGI_STDOUT

Source code

```
1
```

```c
/*
 * Copyright (C) Igor Sysoev
 * Copyright (C) Nginx, Inc.
 */


#include <ngx_config.h>
#include <ngx_core.h>
 #include <ngx_http.h>


 typedef struct {
     ngx_array_t                    caches;  /* ngx_http_file_cache_t * */
 } ngx_http_fastcgi_main_conf_t;


 typedef struct {
     ngx_array_t                   *flushes;
     ngx_array_t                   *lengths;
     ngx_array_t                   *values;
     ngx_uint_t                     number;
     ngx_hash_t                     hash;
 } ngx_http_fastcgi_params_t;


 typedef struct {
     ngx_http_upstream_conf_t       upstream;

     ngx_str_t                      index;

     ngx_http_fastcgi_params_t      params;
 #if (NGX_HTTP_CACHE)
     ngx_http_fastcgi_params_t      params_cache;
 #endif

     ngx_array_t                   *params_source;
     ngx_array_t                   *catch_stderr;

     ngx_array_t                   *fastcgi_lengths;
     ngx_array_t                   *fastcgi_values;

     ngx_flag_t                     keep_conn;

 #if (NGX_HTTP_CACHE)
     ngx_http_complex_value_t       cache_key;
 #endif

 #if (NGX_PCRE)
     ngx_regex_t                   *split_regex;
     ngx_str_t                      split_name;
 #endif
 } ngx_http_fastcgi_loc_conf_t;


 typedef enum {
     ngx_http_fastcgi_st_version = 0,
     ngx_http_fastcgi_st_type,
     ngx_http_fastcgi_st_request_id_hi,
     ngx_http_fastcgi_st_request_id_lo,
     ngx_http_fastcgi_st_content_length_hi,
     ngx_http_fastcgi_st_content_length_lo,
     ngx_http_fastcgi_st_padding_length,
     ngx_http_fastcgi_st_reserved,
     ngx_http_fastcgi_st_data,
     ngx_http_fastcgi_st_padding
 } ngx_http_fastcgi_state_e;


 typedef struct {
     u_char                        *start;
     u_char                        *end;
 } ngx_http_fastcgi_split_part_t;


 typedef struct {
     ngx_http_fastcgi_state_e       state;
```

```
    u_char                       *pos;
    u_char                       *last;
    ngx_uint_t                    type;
    size_t                        length;
    size_t                        padding;

    unsigned                      fastcgi_stdout:1;
    unsigned                      large_stderr:1;

    ngx_array_t                  *split_parts;

    ngx_str_t                     script_name;
    ngx_str_t                     path_info;
} ngx_http_fastcgi_ctx_t;


#define NGX_HTTP_FASTCGI_RESPONDER      1

#define NGX_HTTP_FASTCGI_KEEP_CONN      1

#define NGX_HTTP_FASTCGI_BEGIN_REQUEST  1
#define NGX_HTTP_FASTCGI_ABORT_REQUEST  2
 #define NGX_HTTP_FASTCGI_END_REQUEST    3
 #define NGX_HTTP_FASTCGI_PARAMS         4
 #define NGX_HTTP_FASTCGI_STDIN          5
 #define NGX_HTTP_FASTCGI_STDOUT         6
 #define NGX_HTTP_FASTCGI_STDERR         7
 #define NGX_HTTP_FASTCGI_DATA           8


 typedef struct {
     u_char  version;
     u_char  type;
     u_char  request_id_hi;
     u_char  request_id_lo;
     u_char  content_length_hi;
     u_char  content_length_lo;
     u_char  padding_length;
     u_char  reserved;
 } ngx_http_fastcgi_header_t;


 typedef struct {
     u_char  role_hi;
     u_char  role_lo;
     u_char  flags;
     u_char  reserved[5];
 } ngx_http_fastcgi_begin_request_t;


 typedef struct {
     u_char  version;
     u_char  type;
     u_char  request_id_hi;
     u_char  request_id_lo;
 } ngx_http_fastcgi_header_small_t;


 typedef struct {
     ngx_http_fastcgi_header_t         h0;
     ngx_http_fastcgi_begin_request_t  br;
     ngx_http_fastcgi_header_small_t   h1;
 } ngx_http_fastcgi_request_start_t;


 static ngx_int_t ngx_http_fastcgi_eval(ngx_http_request_t *r,
     ngx_http_fastcgi_loc_conf_t *flcf);
 #if (NGX_HTTP_CACHE)
 static ngx_int_t ngx_http_fastcgi_create_key(ngx_http_request_t *r);
 #endif
 static ngx_int_t ngx_http_fastcgi_create_request(ngx_http_request_t *r);
 static ngx_int_t ngx_http_fastcgi_reinit_request(ngx_http_request_t *r);
 static ngx_int_t ngx_http_fastcgi_process_header(ngx_http_request_t *r);
 static ngx_int_t ngx_http_fastcgi_input_filter_init(void *data);
 static ngx_int_t ngx_http_fastcgi_input_filter(ngx_event_pipe_t *p,
     ngx_buf_t *buf);
```

```
static ngx_int_t ngx_http_fastcgi_non_buffered_filter(void *data,
    ssize_t bytes);
static ngx_int_t ngx_http_fastcgi_process_record(ngx_http_request_t *r,
    ngx_http_fastcgi_ctx_t *f);
static void ngx_http_fastcgi_abort_request(ngx_http_request_t *r);
static void ngx_http_fastcgi_finalize_request(ngx_http_request_t *r,
    ngx_int_t rc);

static ngx_int_t ngx_http_fastcgi_add_variables(ngx_conf_t *cf);
static void *ngx_http_fastcgi_create_main_conf(ngx_conf_t *cf);
static void *ngx_http_fastcgi_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_fastcgi_merge_loc_conf(ngx_conf_t *cf,
    void *parent, void *child);
static ngx_int_t ngx_http_fastcgi_init_params(ngx_conf_t *cf,
    ngx_http_fastcgi_loc_conf_t *conf, ngx_http_fastcgi_params_t *params,
    ngx_keyval_t *default_params);

static ngx_int_t ngx_http_fastcgi_script_name_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_fastcgi_path_info_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_http_fastcgi_ctx_t *ngx_http_fastcgi_split(ngx_http_request_t *r,
    ngx_http_fastcgi_loc_conf_t *flcf);

static char *ngx_http_fastcgi_pass(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_fastcgi_split_path_info(ngx_conf_t *cf,
    ngx_command_t *cmd, void *conf);
static char *ngx_http_fastcgi_store(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
#if (NGX_HTTP_CACHE)
static char *ngx_http_fastcgi_cache(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_fastcgi_cache_key(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
#endif

static char *ngx_http_fastcgi_lowat_check(ngx_conf_t *cf, void *post,
    void *data);


static ngx_conf_post_t  ngx_http_fastcgi_lowat_post =
    { ngx_http_fastcgi_lowat_check };


static ngx_conf_bitmask_t  ngx_http_fastcgi_next_upstream_masks[] = {
    { ngx_string("error"), NGX_HTTP_UPSTREAM_FT_ERROR },
    { ngx_string("timeout"), NGX_HTTP_UPSTREAM_FT_TIMEOUT },
    { ngx_string("invalid_header"), NGX_HTTP_UPSTREAM_FT_INVALID_HEADER },
    { ngx_string("http_500"), NGX_HTTP_UPSTREAM_FT_HTTP_500 },
    { ngx_string("http_503"), NGX_HTTP_UPSTREAM_FT_HTTP_503 },
    { ngx_string("http_403"), NGX_HTTP_UPSTREAM_FT_HTTP_403 },
    { ngx_string("http_404"), NGX_HTTP_UPSTREAM_FT_HTTP_404 },
    { ngx_string("updating"), NGX_HTTP_UPSTREAM_FT_UPDATING },
    { ngx_string("off"), NGX_HTTP_UPSTREAM_FT_OFF },
    { ngx_null_string, 0 }
};


ngx_module_t  ngx_http_fastcgi_module;


static ngx_command_t  ngx_http_fastcgi_commands[] = {

    { ngx_string("fastcgi_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_fastcgi_pass,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("fastcgi_index"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, index),
```

```
      NULL },

    { ngx_string("fastcgi_split_path_info"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_fastcgi_split_path_info,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("fastcgi_store"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_fastcgi_store,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("fastcgi_store_access"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
      ngx_conf_set_access_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.store_access),
      NULL },

    { ngx_string("fastcgi_buffering"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.buffering),
      NULL },

    { ngx_string("fastcgi_ignore_client_abort"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.ignore_client_abort),
      NULL },

    { ngx_string("fastcgi_bind"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_upstream_bind_set_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.local),
      NULL },

    { ngx_string("fastcgi_connect_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.connect_timeout),
      NULL },

    { ngx_string("fastcgi_send_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.send_timeout),
      NULL },

    { ngx_string("fastcgi_send_lowat"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.send_lowat),
      &ngx_http_fastcgi_lowat_post },

    { ngx_string("fastcgi_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_fastcgi_loc_conf_t, upstream.buffer_size),
      NULL },

    { ngx_string("fastcgi_pass_request_headers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
```

```
306         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.pass_request_headers),
307         NULL },
308
309     { ngx_string("fastcgi_pass_request_body"),
310         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
311         ngx_conf_set_flag_slot,
312         NGX_HTTP_LOC_CONF_OFFSET,
313         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.pass_request_body),
314         NULL },
315
316     { ngx_string("fastcgi_intercept_errors"),
317         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
318         ngx_conf_set_flag_slot,
319         NGX_HTTP_LOC_CONF_OFFSET,
320         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.intercept_errors),
321         NULL },
322
323     { ngx_string("fastcgi_read_timeout"),
324         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
325         ngx_conf_set_msec_slot,
326         NGX_HTTP_LOC_CONF_OFFSET,
327         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.read_timeout),
328         NULL },
329
330     { ngx_string("fastcgi_buffers"),
331         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
332         ngx_conf_set_bufs_slot,
333         NGX_HTTP_LOC_CONF_OFFSET,
334         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.bufs),
335         NULL },
336
337     { ngx_string("fastcgi_busy_buffers_size"),
338         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
339         ngx_conf_set_size_slot,
340         NGX_HTTP_LOC_CONF_OFFSET,
341         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.busy_buffers_size_conf),
342         NULL },
343
344     { ngx_string("fastcgi_force_ranges"),
345         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
346         ngx_conf_set_flag_slot,
347         NGX_HTTP_LOC_CONF_OFFSET,
348         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.force_ranges),
349         NULL },
350
351     { ngx_string("fastcgi_limit_rate"),
352         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
353         ngx_conf_set_size_slot,
354         NGX_HTTP_LOC_CONF_OFFSET,
355         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.limit_rate),
356         NULL },
357
358 #if (NGX_HTTP_CACHE)
359
360     { ngx_string("fastcgi_cache"),
361         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
362         ngx_http_fastcgi_cache,
363         NGX_HTTP_LOC_CONF_OFFSET,
364         0,
365         NULL },
366
367     { ngx_string("fastcgi_cache_key"),
368         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
369         ngx_http_fastcgi_cache_key,
370         NGX_HTTP_LOC_CONF_OFFSET,
371         0,
372         NULL },
373
374     { ngx_string("fastcgi_cache_path"),
375         NGX_HTTP_MAIN_CONF|NGX_CONF_2MORE,
376         ngx_http_file_cache_set_slot,
377         NGX_HTTP_MAIN_CONF_OFFSET,
378         offsetof(ngx_http_fastcgi_main_conf_t, caches),
379         &ngx_http_fastcgi_module },
380
381     { ngx_string("fastcgi_cache_bypass"),
```

```
382         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
383         ngx_http_set_predicate_slot,
384         NGX_HTTP_LOC_CONF_OFFSET,
385         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_bypass),
386         NULL },
387
388     { ngx_string("fastcgi_no_cache"),
389         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
390         ngx_http_set_predicate_slot,
391         NGX_HTTP_LOC_CONF_OFFSET,
392         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.no_cache),
393         NULL },
394
395     { ngx_string("fastcgi_cache_valid"),
396         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
397         ngx_http_file_cache_valid_set_slot,
398         NGX_HTTP_LOC_CONF_OFFSET,
399         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_valid),
400         NULL },
401
402     { ngx_string("fastcgi_cache_min_uses"),
403         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
404         ngx_conf_set_num_slot,
405         NGX_HTTP_LOC_CONF_OFFSET,
406         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_min_uses),
407         NULL },
408
409     { ngx_string("fastcgi_cache_use_stale"),
410         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
411         ngx_conf_set_bitmask_slot,
412         NGX_HTTP_LOC_CONF_OFFSET,
413         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_use_stale),
414         &ngx_http_fastcgi_next_upstream_masks },
415
416     { ngx_string("fastcgi_cache_methods"),
417         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
418         ngx_conf_set_bitmask_slot,
419         NGX_HTTP_LOC_CONF_OFFSET,
420         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_methods),
421         &ngx_http_upstream_cache_method_mask },
422
423     { ngx_string("fastcgi_cache_lock"),
424         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
425         ngx_conf_set_flag_slot,
426         NGX_HTTP_LOC_CONF_OFFSET,
427         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_lock),
428         NULL },
429
430     { ngx_string("fastcgi_cache_lock_timeout"),
431         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
432         ngx_conf_set_msec_slot,
433         NGX_HTTP_LOC_CONF_OFFSET,
434         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_lock_timeout),
435         NULL },
436
437     { ngx_string("fastcgi_cache_lock_age"),
438         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
439         ngx_conf_set_msec_slot,
440         NGX_HTTP_LOC_CONF_OFFSET,
441         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_lock_age),
442         NULL },
443
444     { ngx_string("fastcgi_cache_revalidate"),
445         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
446         ngx_conf_set_flag_slot,
447         NGX_HTTP_LOC_CONF_OFFSET,
448         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.cache_revalidate),
449         NULL },
450
451 #endif
452
453     { ngx_string("fastcgi_temp_path"),
454         NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1234,
455         ngx_conf_set_path_slot,
456         NGX_HTTP_LOC_CONF_OFFSET,
457         offsetof(ngx_http_fastcgi_loc_conf_t, upstream.temp_path),
```

```
458         NULL },
459
460     { ngx_string("fastcgi_max_temp_file_size"),
461       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
462       ngx_conf_set_size_slot,
463       NGX_HTTP_LOC_CONF_OFFSET,
464       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.max_temp_file_size_conf),
465       NULL },
466
467     { ngx_string("fastcgi_temp_file_write_size"),
468       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
469       ngx_conf_set_size_slot,
470       NGX_HTTP_LOC_CONF_OFFSET,
471       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.temp_file_write_size_conf),
472       NULL },
473
474     { ngx_string("fastcgi_next_upstream"),
475       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
476       ngx_conf_set_bitmask_slot,
477       NGX_HTTP_LOC_CONF_OFFSET,
478       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.next_upstream),
479       &ngx_http_fastcgi_next_upstream_masks },
480
481     { ngx_string("fastcgi_next_upstream_tries"),
482       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
483       ngx_conf_set_num_slot,
484       NGX_HTTP_LOC_CONF_OFFSET,
485       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.next_upstream_tries),
486       NULL },
487
488     { ngx_string("fastcgi_next_upstream_timeout"),
489       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
490       ngx_conf_set_msec_slot,
491       NGX_HTTP_LOC_CONF_OFFSET,
492       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.next_upstream_timeout),
493       NULL },
494
495     { ngx_string("fastcgi_param"),
496       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE23,
497       ngx_http_upstream_param_set_slot,
498       NGX_HTTP_LOC_CONF_OFFSET,
499       offsetof(ngx_http_fastcgi_loc_conf_t, params_source),
500       NULL },
501
502     { ngx_string("fastcgi_pass_header"),
503       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
504       ngx_conf_set_str_array_slot,
505       NGX_HTTP_LOC_CONF_OFFSET,
506       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.pass_headers),
507       NULL },
508
509     { ngx_string("fastcgi_hide_header"),
510       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
511       ngx_conf_set_str_array_slot,
512       NGX_HTTP_LOC_CONF_OFFSET,
513       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.hide_headers),
514       NULL },
515
516     { ngx_string("fastcgi_ignore_headers"),
517       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
518       ngx_conf_set_bitmask_slot,
519       NGX_HTTP_LOC_CONF_OFFSET,
520       offsetof(ngx_http_fastcgi_loc_conf_t, upstream.ignore_headers),
521       &ngx_http_upstream_ignore_headers_masks },
522
523     { ngx_string("fastcgi_catch_stderr"),
524       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
525       ngx_conf_set_str_array_slot,
526       NGX_HTTP_LOC_CONF_OFFSET,
527       offsetof(ngx_http_fastcgi_loc_conf_t, catch_stderr),
528       NULL },
529
530     { ngx_string("fastcgi_keep_conn"),
531       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
532       ngx_conf_set_flag_slot,
533       NGX_HTTP_LOC_CONF_OFFSET,
```

```
      offsetof(ngx_http_fastcgi_loc_conf_t, keep_conn),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_fastcgi_module_ctx = {
    ngx_http_fastcgi_add_variables,        /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_fastcgi_create_main_conf,     /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_fastcgi_create_loc_conf,      /* create location configuration */
    ngx_http_fastcgi_merge_loc_conf        /* merge location configuration */
};


ngx_module_t  ngx_http_fastcgi_module = {
    NGX_MODULE_V1,
    &ngx_http_fastcgi_module_ctx,          /* module context */
    ngx_http_fastcgi_commands,             /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_fastcgi_request_start_t  ngx_http_fastcgi_request_start = {
    { 1,                                               /* version */
      NGX_HTTP_FASTCGI_BEGIN_REQUEST,                  /* type */
      0,                                               /* request_id_hi */
      1,                                               /* request_id_lo */
      0,                                               /* content_length_hi */
      sizeof(ngx_http_fastcgi_begin_request_t),        /* content_length_lo */
      0,                                               /* padding_length */
      0 },                                             /* reserved */

    { 0,                                               /* role_hi */
      NGX_HTTP_FASTCGI_RESPONDER,                      /* role_lo */
      0, /* NGX_HTTP_FASTCGI_KEEP_CONN */              /* flags */
      { 0, 0, 0, 0, 0 } },                             /* reserved[5] */

    { 1,                                               /* version */
      NGX_HTTP_FASTCGI_PARAMS,                         /* type */
      0,                                               /* request_id_hi */
      1 },                                             /* request_id_lo */

};


static ngx_http_variable_t  ngx_http_fastcgi_vars[] = {

    { ngx_string("fastcgi_script_name"), NULL,
      ngx_http_fastcgi_script_name_variable, 0,
      NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

    { ngx_string("fastcgi_path_info"), NULL,
      ngx_http_fastcgi_path_info_variable, 0,
      NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static ngx_str_t  ngx_http_fastcgi_hide_headers[] = {
```

```
    ngx_string("Status"),
    ngx_string("X-Accel-Expires"),
    ngx_string("X-Accel-Redirect"),
    ngx_string("X-Accel-Limit-Rate"),
    ngx_string("X-Accel-Buffering"),
    ngx_string("X-Accel-Charset"),
    ngx_null_string
};


#if (NGX_HTTP_CACHE)

static ngx_keyval_t  ngx_http_fastcgi_cache_headers[] = {
    { ngx_string("HTTP_IF_MODIFIED_SINCE"),
      ngx_string("$upstream_cache_last_modified") },
    { ngx_string("HTTP_IF_UNMODIFIED_SINCE"), ngx_string("") },
    { ngx_string("HTTP_IF_NONE_MATCH"), ngx_string("$upstream_cache_etag") },
    { ngx_string("HTTP_IF_MATCH"), ngx_string("") },
    { ngx_string("HTTP_RANGE"), ngx_string("") },
    { ngx_string("HTTP_IF_RANGE"), ngx_string("") },
    { ngx_null_string, ngx_null_string }
};

#endif


static ngx_path_init_t  ngx_http_fastcgi_temp_path = {
    ngx_string(NGX_HTTP_FASTCGI_TEMP_PATH), { 1, 2, 0 }
};


static ngx_int_t
ngx_http_fastcgi_handler(ngx_http_request_t *r)
{
    ngx_int_t                     rc;
    ngx_http_upstream_t           *u;
    ngx_http_fastcgi_ctx_t        *f;
    ngx_http_fastcgi_loc_conf_t   *flcf;
#if (NGX_HTTP_CACHE)
    ngx_http_fastcgi_main_conf_t  *fmcf;
#endif

    if (ngx_http_upstream_create(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    f = ngx_pcalloc(r->pool, sizeof(ngx_http_fastcgi_ctx_t));
    if (f == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ngx_http_set_ctx(r, f, ngx_http_fastcgi_module);

    flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);

    if (flcf->fastcgi_lengths) {
        if (ngx_http_fastcgi_eval(r, flcf) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }
    }

    u = r->upstream;

    ngx_str_set(&u->schema, "fastcgi://");
    u->output.tag = (ngx_buf_tag_t) &ngx_http_fastcgi_module;

    u->conf = &flcf->upstream;

#if (NGX_HTTP_CACHE)
    fmcf = ngx_http_get_module_main_conf(r, ngx_http_fastcgi_module);

    u->caches = &fmcf->caches;
    u->create_key = ngx_http_fastcgi_create_key;
#endif

    u->create_request = ngx_http_fastcgi_create_request;
```

```
    u->reinit_request = ngx_http_fastcgi_reinit_request;
    u->process_header = ngx_http_fastcgi_process_header;
    u->abort_request = ngx_http_fastcgi_abort_request;
    u->finalize_request = ngx_http_fastcgi_finalize_request;
    r->state = 0;

    u->buffering = flcf->upstream.buffering;

    u->pipe = ngx_pcalloc(r->pool, sizeof(ngx_event_pipe_t));
    if (u->pipe == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    u->pipe->input_filter = ngx_http_fastcgi_input_filter;
    u->pipe->input_ctx = r;

    u->input_filter_init = ngx_http_fastcgi_input_filter_init;
    u->input_filter = ngx_http_fastcgi_non_buffered_filter;
    u->input_filter_ctx = r;

    rc = ngx_http_read_client_request_body(r, ngx_http_upstream_init);

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        return rc;
    }

    return NGX_DONE;
}


static ngx_int_t
ngx_http_fastcgi_eval(ngx_http_request_t *r, ngx_http_fastcgi_loc_conf_t *flcf)
{
    ngx_url_t             url;
    ngx_http_upstream_t  *u;

    ngx_memzero(&url, sizeof(ngx_url_t));

    if (ngx_http_script_run(r, &url.url, flcf->fastcgi_lengths->elts, 0,
                            flcf->fastcgi_values->elts)
        == NULL)
    {
        return NGX_ERROR;
    }

    url.no_resolve = 1;

    if (ngx_parse_url(r->pool, &url) != NGX_OK) {
         if (url.err) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "%s in upstream \"%V\"", url.err, &url.url);
        }

        return NGX_ERROR;
    }

    u = r->upstream;

    u->resolved = ngx_pcalloc(r->pool, sizeof(ngx_http_upstream_resolved_t));
    if (u->resolved == NULL) {
        return NGX_ERROR;
    }

    if (url.addrs && url.addrs[0].sockaddr) {
        u->resolved->sockaddr = url.addrs[0].sockaddr;
        u->resolved->socklen = url.addrs[0].socklen;
        u->resolved->naddrs = 1;
        u->resolved->host = url.addrs[0].name;

    } else {
        u->resolved->host = url.host;
        u->resolved->port = url.port;
        u->resolved->no_port = url.no_port;
    }

    return NGX_OK;
```

```
}


#if (NGX_HTTP_CACHE)

static ngx_int_t
ngx_http_fastcgi_create_key(ngx_http_request_t *r)
{
    ngx_str_t                    *key;
    ngx_http_fastcgi_loc_conf_t  *flcf;

    key = ngx_array_push(&r->cache->keys);
    if (key == NULL) {
        return NGX_ERROR;
    }

    flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);

    if (ngx_http_complex_value(r, &flcf->cache_key, key) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}

#endif


static ngx_int_t
ngx_http_fastcgi_create_request(ngx_http_request_t *r)
{
    off_t                        file_pos;
    u_char                       ch, *pos, *lowcase_key;
    size_t                       size, len, key_len, val_len, padding,
                                 allocated;
    ngx_uint_t                   i, n, next, hash, skip_empty, header_params;
    ngx_buf_t                   *b;
    ngx_chain_t                 *cl, *body;
    ngx_list_part_t             *part;
    ngx_table_elt_t             *header, **ignored;
    ngx_http_script_code_pt      code;
    ngx_http_script_engine_t     e, le;
    ngx_http_fastcgi_header_t   *h;
    ngx_http_fastcgi_params_t   *params;
    ngx_http_fastcgi_loc_conf_t *flcf;
    ngx_http_script_len_code_pt  lcode;

    len = 0;
    header_params = 0;
    ignored = NULL;

    flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);

#if (NGX_HTTP_CACHE)
    params = r->upstream->cacheable ? &flcf->params_cache : &flcf->params;
#else
    params = &flcf->params;
#endif

    if (params->lengths) {
        ngx_memzero(&le, sizeof(ngx_http_script_engine_t));

        ngx_http_script_flush_no_cacheable_variables(r, params->flushes);
        le.flushed = 1;

        le.ip = params->lengths->elts;
        le.request = r;

        while (*(uintptr_t *) le.ip) {

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            key_len = lcode(&le);

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            skip_empty = lcode(&le);

```

```
            for (val_len = 0; *(uintptr_t *) le.ip; val_len += lcode(&le)) {
                lcode = *(ngx_http_script_len_code_pt *) le.ip;
            }
            le.ip += sizeof(uintptr_t);

            if (skip_empty && val_len == 0) {
                continue;
            }

            len += 1 + key_len + ((val_len > 127) ? 4 : 1) + val_len;
        }
    }

    if (flcf->upstream.pass_request_headers) {

        allocated = 0;
        lowcase_key = NULL;

        if (params->number) {
            n = 0;
            part = &r->headers_in.headers.part;

            while (part) {
                n += part->nelts;
                part = part->next;
            }

            ignored = ngx_palloc(r->pool, n * sizeof(void *));
            if (ignored == NULL) {
                return NGX_ERROR;
            }
        }

        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            if (params->number) {
                if (allocated < header[i].key.len) {
                    allocated = header[i].key.len + 16;
                    lowcase_key = ngx_pnalloc(r->pool, allocated);
                    if (lowcase_key == NULL) {
                        return NGX_ERROR;
                    }
                }

                hash = 0;

                for (n = 0; n < header[i].key.len; n++) {
                    ch = header[i].key.data[n];

                    if (ch >= 'A' && ch <= 'Z') {
                        ch |= 0x20;

                    } else if (ch == '-') {
                        ch = '_';
                    }

                    hash = ngx_hash(hash, ch);
                    lowcase_key[n] = ch;
                }

                if (ngx_hash_find(&params->hash, hash, lowcase_key, n)) {
                    ignored[header_params++] = &header[i];
                    continue;
```

```
            }

            n += sizeof("HTTP_") - 1;

        } else {
            n = sizeof("HTTP_") - 1 + header[i].key.len;
        }

        len += ((n > 127) ? 4 : 1) + ((header[i].value.len > 127) ? 4 : 1)
            + n + header[i].value.len;
    }
}


if (len > 65535) {
    ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                  "fastcgi request record is too big: %uz", len);
    return NGX_ERROR;
}


padding = 8 - len % 8;
padding = (padding == 8) ? 0 : padding;


size = sizeof(ngx_http_fastcgi_header_t)
       + sizeof(ngx_http_fastcgi_begin_request_t)

       + sizeof(ngx_http_fastcgi_header_t)  /* NGX_HTTP_FASTCGI_PARAMS */
       + len + padding
       + sizeof(ngx_http_fastcgi_header_t)  /* NGX_HTTP_FASTCGI_PARAMS */

       + sizeof(ngx_http_fastcgi_header_t); /* NGX_HTTP_FASTCGI_STDIN */


b = ngx_create_temp_buf(r->pool, size);
if (b == NULL) {
    return NGX_ERROR;
}

cl = ngx_alloc_chain_link(r->pool);
if (cl == NULL) {
    return NGX_ERROR;
}

cl->buf = b;

ngx_http_fastcgi_request_start.br.flags =
    flcf->keep_conn ? NGX_HTTP_FASTCGI_KEEP_CONN : 0;

ngx_memcpy(b->pos, &ngx_http_fastcgi_request_start,
           sizeof(ngx_http_fastcgi_request_start_t));

h = (ngx_http_fastcgi_header_t *)
         (b->pos + sizeof(ngx_http_fastcgi_header_t)
                 + sizeof(ngx_http_fastcgi_begin_request_t));

h->content_length_hi = (u_char) ((len >> 8) & 0xff);
h->content_length_lo = (u_char) (len & 0xff);
h->padding_length = (u_char) padding;
h->reserved = 0;

b->last = b->pos + sizeof(ngx_http_fastcgi_header_t)
                 + sizeof(ngx_http_fastcgi_begin_request_t)
                 + sizeof(ngx_http_fastcgi_header_t);


if (params->lengths) {
    ngx_memzero(&e, sizeof(ngx_http_script_engine_t));

    e.ip = params->values->elts;
    e.pos = b->last;
    e.request = r;
    e.flushed = 1;

    le.ip = params->lengths->elts;
```

```
990
991         while (*(uintptr_t *) le.ip) {
992
993             lcode = *(ngx_http_script_len_code_pt *) le.ip;
994             key_len = (u_char) lcode(&le);
995
996             lcode = *(ngx_http_script_len_code_pt *) le.ip;
997             skip_empty = lcode(&le);
998
999             for (val_len = 0; *(uintptr_t *) le.ip; val_len += lcode(&le)) {
1000                 lcode = *(ngx_http_script_len_code_pt *) le.ip;
1001             }
1002             le.ip += sizeof(uintptr_t);
1003
1004             if (skip_empty && val_len == 0) {
1005                 e.skip = 1;
1006
1007                 while (*(uintptr_t *) e.ip) {
1008                     code = *(ngx_http_script_code_pt *) e.ip;
1009                     code((ngx_http_script_engine_t *) &e);
1010                 }
1011                 e.ip += sizeof(uintptr_t);
1012
1013                 e.skip = 0;
1014
1015                 continue;
1016             }
1017
1018             *e.pos++ = (u_char) key_len;
1019
1020             if (val_len > 127) {
1021                 *e.pos++ = (u_char) (((val_len >> 24) & 0x7f) | 0x80);
1022                 *e.pos++ = (u_char) ((val_len >> 16) & 0xff);
1023                 *e.pos++ = (u_char) ((val_len >> 8) & 0xff);
1024                 *e.pos++ = (u_char) (val_len & 0xff);
1025
1026             } else {
1027                 *e.pos++ = (u_char) val_len;
1028             }
1029
1030             while (*(uintptr_t *) e.ip) {
1031                 code = *(ngx_http_script_code_pt *) e.ip;
1032                 code((ngx_http_script_engine_t *) &e);
1033             }
1034             e.ip += sizeof(uintptr_t);
1035
1036             ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1037                            "fastcgi param: \"%*s: %*s\"",
1038                            key_len, e.pos - (key_len + val_len),
1039                            val_len, e.pos - val_len);
1040         }
1041
1042         b->last = e.pos;
1043     }
1044
1045
1046     if (flcf->upstream.pass_request_headers) {
1047
1048         part = &r->headers_in.headers.part;
1049         header = part->elts;
1050
1051         for (i = 0; /* void */; i++) {
1052
1053             if (i >= part->nelts) {
1054                 if (part->next == NULL) {
1055                     break;
1056                 }
1057
1058                 part = part->next;
1059                 header = part->elts;
1060                 i = 0;
1061             }
1062
1063             for (n = 0; n < header_params; n++) {
1064                 if (&header[i] == ignored[n]) {
1065                     goto next;
```

```
1066                 }
1067             }
1068
1069             key_len = sizeof("HTTP_") - 1 + header[i].key.len;
1070             if (key_len > 127) {
1071                 *b->last++ = (u_char) (((key_len >> 24) & 0x7f) | 0x80);
1072                 *b->last++ = (u_char) ((key_len >> 16) & 0xff);
1073                 *b->last++ = (u_char) ((key_len >> 8) & 0xff);
1074                 *b->last++ = (u_char) (key_len & 0xff);
1075
1076             } else {
1077                 *b->last++ = (u_char) key_len;
1078             }
1079
1080             val_len = header[i].value.len;
1081             if (val_len > 127) {
1082                 *b->last++ = (u_char) (((val_len >> 24) & 0x7f) | 0x80);
1083                 *b->last++ = (u_char) ((val_len >> 16) & 0xff);
1084                 *b->last++ = (u_char) ((val_len >> 8) & 0xff);
1085                 *b->last++ = (u_char) (val_len & 0xff);
1086
1087             } else {
1088                 *b->last++ = (u_char) val_len;
1089             }
1090
1091             b->last = ngx_cpymem(b->last, "HTTP_", sizeof("HTTP_") - 1);
1092
1093             for (n = 0; n < header[i].key.len; n++) {
1094                 ch = header[i].key.data[n];
1095
1096                 if (ch >= 'a' && ch <= 'z') {
1097                     ch &= ~0x20;
1098
1099                 } else if (ch == '-') {
1100                     ch = '_';
1101                 }
1102
1103                 *b->last++ = ch;
1104             }
1105
1106             b->last = ngx_copy(b->last, header[i].value.data, val_len);
1107
1108             ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1109                            "fastcgi param: \"%*s: %*s\"",
1110                            key_len, b->last - (key_len + val_len),
1111                            val_len, b->last - val_len);
1112         next:
1113
1114             continue;
1115         }
1116     }
1117
1118
1119     if (padding) {
1120         ngx_memzero(b->last, padding);
1121         b->last += padding;
1122     }
1123
1124
1125     h = (ngx_http_fastcgi_header_t *) b->last;
1126     b->last += sizeof(ngx_http_fastcgi_header_t);
1127
1128     h->version = 1;
1129     h->type = NGX_HTTP_FASTCGI_PARAMS;
1130     h->request_id_hi = 0;
1131     h->request_id_lo = 1;
1132     h->content_length_hi = 0;
1133     h->content_length_lo = 0;
1134     h->padding_length = 0;
1135     h->reserved = 0;
1136
1137     h = (ngx_http_fastcgi_header_t *) b->last;
1138     b->last += sizeof(ngx_http_fastcgi_header_t);
1139
1140     if (flcf->upstream.pass_request_body) {
1141         body = r->upstream->request_bufs;
```

```
        r->upstream->request_bufs = cl;

#if (NGX_SUPPRESS_WARN)
        file_pos = 0;
        pos = NULL;
#endif

        while (body) {

            if (body->buf->in_file) {
                file_pos = body->buf->file_pos;

            } else {
                pos = body->buf->pos;
            }

            next = 0;

            do {
                b = ngx_alloc_buf(r->pool);
                if (b == NULL) {
                    return NGX_ERROR;
                }

                ngx_memcpy(b, body->buf, sizeof(ngx_buf_t));

                if (body->buf->in_file) {
                    b->file_pos = file_pos;
                    file_pos += 32 * 1024;

                    if (file_pos >= body->buf->file_last) {
                        file_pos = body->buf->file_last;
                        next = 1;
                    }

                    b->file_last = file_pos;
                    len = (ngx_uint_t) (file_pos - b->file_pos);

                } else {
                    b->pos = pos;
                    b->start = pos;
                    pos += 32 * 1024;

                    if (pos >= body->buf->last) {
                        pos = body->buf->last;
                        next = 1;
                    }

                    b->last = pos;
                    len = (ngx_uint_t) (pos - b->pos);
                }

                padding = 8 - len % 8;
                padding = (padding == 8) ? 0 : padding;

                h->version = 1;
                h->type = NGX_HTTP_FASTCGI_STDIN;
                h->request_id_hi = 0;
                h->request_id_lo = 1;
                h->content_length_hi = (u_char) ((len >> 8) & 0xff);
                h->content_length_lo = (u_char) (len & 0xff);
                h->padding_length = (u_char) padding;
                h->reserved = 0;

                cl->next = ngx_alloc_chain_link(r->pool);
                if (cl->next == NULL) {
                    return NGX_ERROR;
                }

                cl = cl->next;
                cl->buf = b;

                b = ngx_create_temp_buf(r->pool,
                                        sizeof(ngx_http_fastcgi_header_t)
                                        + padding);
                if (b == NULL) {
```

```
                    return NGX_ERROR;
                }

                if (padding) {
                    ngx_memzero(b->last, padding);
                    b->last += padding;
                }

                h = (ngx_http_fastcgi_header_t *) b->last;
                b->last += sizeof(ngx_http_fastcgi_header_t);

                cl->next = ngx_alloc_chain_link(r->pool);
                if (cl->next == NULL) {
                    return NGX_ERROR;
                }

                cl = cl->next;
                cl->buf = b;

            } while (!next);

            body = body->next;
        }

    } else {
        r->upstream->request_bufs = cl;
    }

    h->version = 1;
    h->type = NGX_HTTP_FASTCGI_STDIN;
    h->request_id_hi = 0;
    h->request_id_lo = 1;
    h->content_length_hi = 0;
    h->content_length_lo = 0;
    h->padding_length = 0;
    h->reserved = 0;

    cl->next = NULL;

    return NGX_OK;
}


static ngx_int_t
ngx_http_fastcgi_reinit_request(ngx_http_request_t *r)
{
    ngx_http_fastcgi_ctx_t  *f;

    f = ngx_http_get_module_ctx(r, ngx_http_fastcgi_module);

    if (f == NULL) {
        return NGX_OK;
    }

    f->state = ngx_http_fastcgi_st_version;
    f->fastcgi_stdout = 0;
    f->large_stderr = 0;

    if (f->split_parts) {
        f->split_parts->nelts = 0;
    }

    r->state = 0;

    return NGX_OK;
}


static ngx_int_t
ngx_http_fastcgi_process_header(ngx_http_request_t *r)
{
    u_char                        *p, *msg, *start, *last,
                                  *part_start, *part_end;
    size_t                         size;
    ngx_str_t                     *status_line, *pattern;
    ngx_int_t                      rc, status;
```

```
    ngx_buf_t                     buf;
    ngx_uint_t                    i;
    ngx_table_elt_t              *h;
    ngx_http_upstream_t          *u;
    ngx_http_fastcgi_ctx_t       *f;
    ngx_http_upstream_header_t    *hh;
    ngx_http_fastcgi_loc_conf_t   *flcf;
    ngx_http_fastcgi_split_part_t  *part;
    ngx_http_upstream_main_conf_t  *umcf;

    f = ngx_http_get_module_ctx(r, ngx_http_fastcgi_module);

    umcf = ngx_http_get_module_main_conf(r, ngx_http_upstream_module);

    u = r->upstream;

    for ( ;; ) {

        if (f->state < ngx_http_fastcgi_st_data) {

            f->pos = u->buffer.pos;
            f->last = u->buffer.last;

            rc = ngx_http_fastcgi_process_record(r, f);

            u->buffer.pos = f->pos;
            u->buffer.last = f->last;

            if (rc == NGX_AGAIN) {
                return NGX_AGAIN;
            }

            if (rc == NGX_ERROR) {
                return NGX_HTTP_UPSTREAM_INVALID_HEADER;
            }

            if (f->type != NGX_HTTP_FASTCGI_STDOUT
                && f->type != NGX_HTTP_FASTCGI_STDERR)
            {
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "upstream sent unexpected FastCGI record: %d",
                              f->type);

                return NGX_HTTP_UPSTREAM_INVALID_HEADER;
            }

            if (f->type == NGX_HTTP_FASTCGI_STDOUT && f->length == 0) {
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "upstream prematurely closed FastCGI stdout");

                return NGX_HTTP_UPSTREAM_INVALID_HEADER;
            }
        }

        if (f->state == ngx_http_fastcgi_st_padding) {

            if (u->buffer.pos + f->padding < u->buffer.last) {
                f->state = ngx_http_fastcgi_st_version;
                u->buffer.pos += f->padding;

                continue;
            }

            if (u->buffer.pos + f->padding == u->buffer.last) {
                f->state = ngx_http_fastcgi_st_version;
                u->buffer.pos = u->buffer.last;

                return NGX_AGAIN;
            }

            f->padding -= u->buffer.last - u->buffer.pos;
            u->buffer.pos = u->buffer.last;

            return NGX_AGAIN;
        }

```

```
1370
1371        /* f->state == ngx_http_fastcgi_st_data */
1372
1373        if (f->type == NGX_HTTP_FASTCGI_STDERR) {
1374
1375            if (f->length) {
1376                msg = u->buffer.pos;
1377
1378                if (u->buffer.pos + f->length <= u->buffer.last) {
1379                    u->buffer.pos += f->length;
1380                    f->length = 0;
1381                    f->state = ngx_http_fastcgi_st_padding;
1382
1383                } else {
1384                    f->length -= u->buffer.last - u->buffer.pos;
1385                    u->buffer.pos = u->buffer.last;
1386                }
1387
1388                for (p = u->buffer.pos - 1; msg < p; p--) {
1389                    if (*p != LF && *p != CR && *p != '.' && *p != ' ') {
1390                        break;
1391                    }
1392                }
1393
1394                p++;
1395
1396                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1397                              "FastCGI sent in stderr: \"%*s\"", p - msg, msg);
1398
1399                flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);
1400
1401                if (flcf->catch_stderr) {
1402                    pattern = flcf->catch_stderr->elts;
1403
1404                    for (i = 0; i < flcf->catch_stderr->nelts; i++) {
1405                        if (ngx_strnstr(msg, (char *) pattern[i].data,
1406                                        p - msg)
1407                            != NULL)
1408                        {
1409                            return NGX_HTTP_UPSTREAM_INVALID_HEADER;
1410                        }
1411                    }
1412                }
1413
1414                if (u->buffer.pos == u->buffer.last) {
1415
1416                    if (!f->fastcgi_stdout) {
1417
1418                        /*
1419                         * the special handling the large number
1420                         * of the PHP warnings to not allocate memory
1421                         */
1422
1423 #if (NGX_HTTP_CACHE)
1424                        if (r->cache) {
1425                            u->buffer.pos = u->buffer.start
1426                                                + r->cache->header_start;
1427                        } else {
1428                            u->buffer.pos = u->buffer.start;
1429                        }
1430 #else
1431                        u->buffer.pos = u->buffer.start;
1432 #endif
1433                        u->buffer.last = u->buffer.pos;
1434                        f->large_stderr = 1;
1435                    }
1436
1437                    return NGX_AGAIN;
1438                }
1439
1440            } else {
1441                f->state = ngx_http_fastcgi_st_padding;
1442            }
1443
1444            continue;
1445        }
```

```
1446
1447
1448            /* f->type == NGX_HTTP_FASTCGI_STDOUT */
1449
1450    #if (NGX_HTTP_CACHE)
1451
1452            if (f->large_stderr && r->cache) {
1453                u_char                     *start;
1454                ssize_t                     len;
1455                ngx_http_fastcgi_header_t  *fh;
1456
1457                start = u->buffer.start + r->cache->header_start;
1458
1459                len = u->buffer.pos - start - 2 * sizeof(ngx_http_fastcgi_header_t);
1460
1461                /*
1462                 * A tail of large stderr output before HTTP header is placed
1463                 * in a cache file without a FastCGI record header.
1464                 * To workaround it we put a dummy FastCGI record header at the
1465                 * start of the stderr output or update r->cache_header_start,
1466                 * if there is no enough place for the record header.
1467                 */
1468
1469                if (len >= 0) {
1470                    fh = (ngx_http_fastcgi_header_t *) start;
1471                    fh->version = 1;
1472                    fh->type = NGX_HTTP_FASTCGI_STDERR;
1473                    fh->request_id_hi = 0;
1474                    fh->request_id_lo = 1;
1475                    fh->content_length_hi = (u_char) ((len >> 8) & 0xff);
1476                    fh->content_length_lo = (u_char) (len & 0xff);
1477                    fh->padding_length = 0;
1478                    fh->reserved = 0;
1479
1480                } else {
1481                    r->cache->header_start += u->buffer.pos - start
1482                                              - sizeof(ngx_http_fastcgi_header_t);
1483                }
1484
1485                f->large_stderr = 0;
1486            }
1487
1488    #endif
1489
1490            f->fastcgi_stdout = 1;
1491
1492            start = u->buffer.pos;
1493
1494            if (u->buffer.pos + f->length < u->buffer.last) {
1495
1496                /*
1497                 * set u->buffer.last to the end of the FastCGI record data
1498                 * for ngx_http_parse_header_line()
1499                 */
1500
1501                last = u->buffer.last;
1502                u->buffer.last = u->buffer.pos + f->length;
1503
1504            } else {
1505                last = NULL;
1506            }
1507
1508            for ( ;; ) {
1509
1510                part_start = u->buffer.pos;
1511                part_end = u->buffer.last;
1512
1513                rc = ngx_http_parse_header_line(r, &u->buffer, 1);
1514
1515                ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1516                               "http fastcgi parser: %d", rc);
1517
1518                if (rc == NGX_AGAIN) {
1519                    break;
1520                }
1521
```

```
1522            if (rc == NGX_OK) {
1523
1524                /* a header line has been parsed successfully */
1525
1526                h = ngx_list_push(&u->headers_in.headers);
1527                if (h == NULL) {
1528                    return NGX_ERROR;
1529                }
1530
1531                if (f->split_parts && f->split_parts->nelts) {
1532
1533                    part = f->split_parts->elts;
1534                    size = u->buffer.pos - part_start;
1535
1536                    for (i = 0; i < f->split_parts->nelts; i++) {
1537                        size += part[i].end - part[i].start;
1538                    }
1539
1540                    p = ngx_pnalloc(r->pool, size);
1541                    if (p == NULL) {
1542                        return NGX_ERROR;
1543                    }
1544
1545                    buf.pos = p;
1546
1547                    for (i = 0; i < f->split_parts->nelts; i++) {
1548                        p = ngx_cpymem(p, part[i].start,
1549                                       part[i].end - part[i].start);
1550                    }
1551
1552                    p = ngx_cpymem(p, part_start, u->buffer.pos - part_start);
1553
1554                    buf.last = p;
1555
1556                    f->split_parts->nelts = 0;
1557
1558                    rc = ngx_http_parse_header_line(r, &buf, 1);
1559
1560                    if (rc != NGX_OK) {
1561                        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
1562                                      "invalid header after joining "
1563                                      "FastCGI records");
1564                        return NGX_ERROR;
1565                    }
1566
1567                    h->key.len = r->header_name_end - r->header_name_start;
1568                    h->key.data = r->header_name_start;
1569                    h->key.data[h->key.len] = '\0';
1570
1571                    h->value.len = r->header_end - r->header_start;
1572                    h->value.data = r->header_start;
1573                    h->value.data[h->value.len] = '\0';
1574
1575                    h->lowcase_key = ngx_pnalloc(r->pool, h->key.len);
1576                    if (h->lowcase_key == NULL) {
1577                        return NGX_ERROR;
1578                    }
1579
1580                } else {
1581
1582                    h->key.len = r->header_name_end - r->header_name_start;
1583                    h->value.len = r->header_end - r->header_start;
1584
1585                    h->key.data = ngx_pnalloc(r->pool,
1586                                              h->key.len + 1 + h->value.len + 1
1587                                              + h->key.len);
1588                    if (h->key.data == NULL) {
1589                        return NGX_ERROR;
1590                    }
1591
1592                    h->value.data = h->key.data + h->key.len + 1;
1593                    h->lowcase_key = h->key.data + h->key.len + 1
1594                                     + h->value.len + 1;
1595
1596                    ngx_memcpy(h->key.data, r->header_name_start, h->key.len);
1597                    h->key.data[h->key.len] = '\0';
```

```
                ngx_memcpy(h->value.data, r->header_start, h->value.len);
                h->value.data[h->value.len] = '\0';
            }

            h->hash = r->header_hash;

            if (h->key.len == r->lowcase_index) {
                ngx_memcpy(h->lowcase_key, r->lowcase_header, h->key.len);

            } else {
                ngx_strlow(h->lowcase_key, h->key.data, h->key.len);
            }

            hh = ngx_hash_find(&umcf->headers_in_hash, h->hash,
                               h->lowcase_key, h->key.len);

            if (hh && hh->handler(r, h, hh->offset) != NGX_OK) {
                return NGX_ERROR;
            }

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http fastcgi header: \"%V: %V\"",
                           &h->key, &h->value);

            if (u->buffer.pos < u->buffer.last) {
                continue;
            }

            /* the end of the FastCGI record */

            break;
        }

        if (rc == NGX_HTTP_PARSE_HEADER_DONE) {

            /* a whole header has been parsed successfully */

            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http fastcgi header done");

            if (u->headers_in.status) {
                status_line = &u->headers_in.status->value;

                status = ngx_atoi(status_line->data, 3);

                if (status == NGX_ERROR) {
                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                                  "upstream sent invalid status \"%V\"",
                                  status_line);
                    return NGX_HTTP_UPSTREAM_INVALID_HEADER;
                }

                u->headers_in.status_n = status;
                u->headers_in.status_line = *status_line;

            } else if (u->headers_in.location) {
                u->headers_in.status_n = 302;
                ngx_str_set(&u->headers_in.status_line,
                            "302 Moved Temporarily");

            } else {
                u->headers_in.status_n = 200;
                ngx_str_set(&u->headers_in.status_line, "200 OK");
            }

            if (u->state && u->state->status == 0) {
                u->state->status = u->headers_in.status_n;
            }

            break;
        }

        /* there was error while a header line parsing */

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "upstream sent invalid header");
```

```
1674
1675            return NGX_HTTP_UPSTREAM_INVALID_HEADER;
1676        }
1677
1678        if (last) {
1679            u->buffer.last = last;
1680        }
1681
1682        f->length -= u->buffer.pos - start;
1683
1684        if (f->length == 0) {
1685            f->state = ngx_http_fastcgi_st_padding;
1686        }
1687
1688        if (rc == NGX_HTTP_PARSE_HEADER_DONE) {
1689            return NGX_OK;
1690        }
1691
1692        if (rc == NGX_OK) {
1693            continue;
1694        }
1695
1696        /* rc == NGX_AGAIN */
1697
1698        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1699                       "upstream split a header line in FastCGI records");
1700
1701        if (f->split_parts == NULL) {
1702            f->split_parts = ngx_array_create(r->pool, 1,
1703                                              sizeof(ngx_http_fastcgi_split_part_t));
1704            if (f->split_parts == NULL) {
1705                return NGX_ERROR;
1706            }
1707        }
1708
1709        part = ngx_array_push(f->split_parts);
1710        if (part == NULL) {
1711            return NGX_ERROR;
1712        }
1713
1714        part->start = part_start;
1715        part->end = part_end;
1716
1717        if (u->buffer.pos < u->buffer.last) {
1718            continue;
1719        }
1720
1721        return NGX_AGAIN;
1722    }
1723 }
1724
1725
1726 static ngx_int_t
1727 ngx_http_fastcgi_input_filter_init(void *data)
1728 {
1729     ngx_http_request_t           *r = data;
1730     ngx_http_fastcgi_loc_conf_t  *flcf;
1731
1732     flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);
1733
1734     r->upstream->pipe->length = flcf->keep_conn ?
1735                                 (off_t) sizeof(ngx_http_fastcgi_header_t) : -1;
1736
1737     return NGX_OK;
1738 }
1739
1740
1741 static ngx_int_t
1742 ngx_http_fastcgi_input_filter(ngx_event_pipe_t *p, ngx_buf_t *buf)
1743 {
1744     u_char                       *m, *msg;
1745     ngx_int_t                     rc;
1746     ngx_buf_t                    *b, **prev;
1747     ngx_chain_t                  *cl;
1748     ngx_http_request_t           *r;
1749     ngx_http_fastcgi_ctx_t       *f;
```

```
1750      ngx_http_fastcgi_loc_conf_t  *flcf;
1751
1752      if (buf->pos == buf->last) {
1753          return NGX_OK;
1754      }
1755
1756      r = p->input_ctx;
1757      f = ngx_http_get_module_ctx(r, ngx_http_fastcgi_module);
1758      flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);
1759
1760      b = NULL;
1761      prev = &buf->shadow;
1762
1763      f->pos = buf->pos;
1764      f->last = buf->last;
1765
1766      for ( ;; ) {
1767          if (f->state < ngx_http_fastcgi_st_data) {
1768
1769              rc = ngx_http_fastcgi_process_record(r, f);
1770
1771              if (rc == NGX_AGAIN) {
1772                  break;
1773              }
1774
1775              if (rc == NGX_ERROR) {
1776                  return NGX_ERROR;
1777              }
1778
1779              if (f->type == NGX_HTTP_FASTCGI_STDOUT && f->length == 0) {
1780                  f->state = ngx_http_fastcgi_st_padding;
1781
1782                  if (!flcf->keep_conn) {
1783                      p->upstream_done = 1;
1784                  }
1785
1786                  ngx_log_debug0(NGX_LOG_DEBUG_HTTP, p->log, 0,
1787                                 "http fastcgi closed stdout");
1788
1789                  continue;
1790              }
1791
1792              if (f->type == NGX_HTTP_FASTCGI_END_REQUEST) {
1793
1794                  ngx_log_debug0(NGX_LOG_DEBUG_HTTP, p->log, 0,
1795                                 "http fastcgi sent end request");
1796
1797                  if (!flcf->keep_conn) {
1798                      p->upstream_done = 1;
1799                      break;
1800                  }
1801
1802                  continue;
1803              }
1804          }
1805
1806
1807          if (f->state == ngx_http_fastcgi_st_padding) {
1808
1809              if (f->type == NGX_HTTP_FASTCGI_END_REQUEST) {
1810
1811                  if (f->pos + f->padding < f->last) {
1812                      p->upstream_done = 1;
1813                      break;
1814                  }
1815
1816                  if (f->pos + f->padding == f->last) {
1817                      p->upstream_done = 1;
1818                      r->upstream->keepalive = 1;
1819                      break;
1820                  }
1821
1822                  f->padding -= f->last - f->pos;
1823
1824                  break;
1825              }
```

```
            if (f->pos + f->padding < f->last) {
                f->state = ngx_http_fastcgi_st_version;
                f->pos += f->padding;

                continue;
            }

            if (f->pos + f->padding == f->last) {
                f->state = ngx_http_fastcgi_st_version;

                break;
            }

            f->padding -= f->last - f->pos;

            break;
        }


        /* f->state == ngx_http_fastcgi_st_data */

        if (f->type == NGX_HTTP_FASTCGI_STDERR) {

            if (f->length) {

                if (f->pos == f->last) {
                    break;
                }

                msg = f->pos;

                if (f->pos + f->length <= f->last) {
                    f->pos += f->length;
                    f->length = 0;
                    f->state = ngx_http_fastcgi_st_padding;

                } else {
                    f->length -= f->last - f->pos;
                    f->pos = f->last;
                }

                for (m = f->pos - 1; msg < m; m--) {
                    if (*m != LF && *m != CR && *m != '.' && *m != ' ') {
                        break;
                    }
                }

                ngx_log_error(NGX_LOG_ERR, p->log, 0,
                              "FastCGI sent in stderr: \"%*s\"",
                              m + 1 - msg, msg);

            } else {
                f->state = ngx_http_fastcgi_st_padding;
            }

            continue;
        }

        if (f->type == NGX_HTTP_FASTCGI_END_REQUEST) {

            if (f->pos + f->length <= f->last) {
                f->state = ngx_http_fastcgi_st_padding;
                f->pos += f->length;

                continue;
            }

            f->length -= f->last - f->pos;

            break;
        }


        /* f->type == NGX_HTTP_FASTCGI_STDOUT */
```

```
1902            if (f->pos == f->last) {
1903                break;
1904            }
1905
1906            cl = ngx_chain_get_free_buf(p->pool, &p->free);
1907            if (cl == NULL) {
1908                return NGX_ERROR;
1909            }
1910
1911            b = cl->buf;
1912
1913            ngx_memzero(b, sizeof(ngx_buf_t));
1914
1915            b->pos = f->pos;
1916            b->start = buf->start;
1917            b->end = buf->end;
1918            b->tag = p->tag;
1919            b->temporary = 1;
1920            b->recycled = 1;
1921
1922            *prev = b;
1923            prev = &b->shadow;
1924
1925            if (p->in) {
1926                *p->last_in = cl;
1927            } else {
1928                p->in = cl;
1929            }
1930            p->last_in = &cl->next;
1931
1932
1933            /* STUB */ b->num = buf->num;
1934
1935            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, p->log, 0,
1936                           "input buf #%d %p", b->num, b->pos);
1937
1938            if (f->pos + f->length <= f->last) {
1939                f->state = ngx_http_fastcgi_st_padding;
1940                f->pos += f->length;
1941                b->last = f->pos;
1942
1943                continue;
1944            }
1945
1946            f->length -= f->last - f->pos;
1947
1948            b->last = f->last;
1949
1950            break;
1951
1952        }
1953
1954        if (flcf->keep_conn) {
1955
1956            /* set p->length, minimal amount of data we want to see */
1957
1958            if (f->state < ngx_http_fastcgi_st_data) {
1959                p->length = 1;
1960
1961            } else if (f->state == ngx_http_fastcgi_st_padding) {
1962                p->length = f->padding;
1963
1964            } else {
1965                /* ngx_http_fastcgi_st_data */
1966
1967                p->length = f->length;
1968            }
1969        }
1970
1971        if (b) {
1972            b->shadow = buf;
1973            b->last_shadow = 1;
1974
1975            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, p->log, 0,
1976                           "input buf %p %z", b->pos, b->last - b->pos);
1977
```

```
        return NGX_OK;
    }

    /* there is no data record in the buf, add it to free chain */

    if (ngx_event_pipe_add_free_buf(p, buf) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_fastcgi_non_buffered_filter(void *data, ssize_t bytes)
{
    u_char                  *m, *msg;
    ngx_int_t                rc;
    ngx_buf_t               *b, *buf;
    ngx_chain_t             *cl, **ll;
    ngx_http_request_t      *r;
    ngx_http_upstream_t     *u;
    ngx_http_fastcgi_ctx_t  *f;

    r = data;
    f = ngx_http_get_module_ctx(r, ngx_http_fastcgi_module);

    u = r->upstream;
    buf = &u->buffer;

    buf->pos = buf->last;
    buf->last += bytes;

    for (cl = u->out_bufs, ll = &u->out_bufs; cl; cl = cl->next) {
        ll = &cl->next;
    }

    f->pos = buf->pos;
    f->last = buf->last;

    for ( ;; ) {
        if (f->state < ngx_http_fastcgi_st_data) {

            rc = ngx_http_fastcgi_process_record(r, f);

            if (rc == NGX_AGAIN) {
                break;
            }

            if (rc == NGX_ERROR) {
                return NGX_ERROR;
            }

            if (f->type == NGX_HTTP_FASTCGI_STDOUT && f->length == 0) {
                f->state = ngx_http_fastcgi_st_padding;

                ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                               "http fastcgi closed stdout");

                continue;
            }
        }

        if (f->state == ngx_http_fastcgi_st_padding) {

            if (f->type == NGX_HTTP_FASTCGI_END_REQUEST) {

                if (f->pos + f->padding < f->last) {
                    u->length = 0;
                    break;
                }

                if (f->pos + f->padding == f->last) {
                    u->length = 0;
                    u->keepalive = 1;
                    break;
```

```
            }

            f->padding -= f->last - f->pos;

            break;
        }

        if (f->pos + f->padding < f->last) {
            f->state = ngx_http_fastcgi_st_version;
            f->pos += f->padding;

            continue;
        }

        if (f->pos + f->padding == f->last) {
            f->state = ngx_http_fastcgi_st_version;

            break;
        }

        f->padding -= f->last - f->pos;

        break;
    }


    /* f->state == ngx_http_fastcgi_st_data */

    if (f->type == NGX_HTTP_FASTCGI_STDERR) {

        if (f->length) {

            if (f->pos == f->last) {
                break;
            }

            msg = f->pos;

            if (f->pos + f->length <= f->last) {
                f->pos += f->length;
                f->length = 0;
                f->state = ngx_http_fastcgi_st_padding;

            } else {
                f->length -= f->last - f->pos;
                f->pos = f->last;
            }

            for (m = f->pos - 1; msg < m; m--) {
                if (*m != LF && *m != CR && *m != '.' && *m != ' ') {
                    break;
                }
            }

            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "FastCGI sent in stderr: \"%*s\"",
                          m + 1 - msg, msg);

        } else {
            f->state = ngx_http_fastcgi_st_padding;
        }

        continue;
    }

    if (f->type == NGX_HTTP_FASTCGI_END_REQUEST) {

        if (f->pos + f->length <= f->last) {
            f->state = ngx_http_fastcgi_st_padding;
            f->pos += f->length;

            continue;
        }

        f->length -= f->last - f->pos;

```

```
            break;
        }


        /* f->type == NGX_HTTP_FASTCGI_STDOUT */

        if (f->pos == f->last) {
            break;
        }

        cl = ngx_chain_get_free_buf(r->pool, &u->free_bufs);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        *ll = cl;
        ll = &cl->next;

        b = cl->buf;

        b->flush = 1;
        b->memory = 1;

        b->pos = f->pos;
        b->tag = u->output.tag;

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http fastcgi output buf %p", b->pos);

        if (f->pos + f->length <= f->last) {
            f->state = ngx_http_fastcgi_st_padding;
            f->pos += f->length;
            b->last = f->pos;

            continue;
        }

        f->length -= f->last - f->pos;
        b->last = f->last;

        break;
    }

    /* provide continuous buffer for subrequests in memory */

    if (r->subrequest_in_memory) {

        cl = u->out_bufs;

        if (cl) {
            buf->pos = cl->buf->pos;
        }

        buf->last = buf->pos;

        for (cl = u->out_bufs; cl; cl = cl->next) {
            ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http fastcgi in memory %p-%p %uz",
                           cl->buf->pos, cl->buf->last, ngx_buf_size(cl->buf));

            if (buf->last == cl->buf->pos) {
                buf->last = cl->buf->last;
                continue;
            }

            buf->last = ngx_movemem(buf->last, cl->buf->pos,
                                    cl->buf->last - cl->buf->pos);

            cl->buf->pos = buf->last - (cl->buf->last - cl->buf->pos);
            cl->buf->last = buf->last;
        }
    }

    return NGX_OK;
}
```

```
2206
2207 static ngx_int_t
2208 ngx_http_fastcgi_process_record(ngx_http_request_t *r,
2209     ngx_http_fastcgi_ctx_t *f)
2210 {
2211     u_char                     ch, *p;
2212     ngx_http_fastcgi_state_e   state;
2213
2214     state = f->state;
2215
2216     for (p = f->pos; p < f->last; p++) {
2217
2218         ch = *p;
2219
2220         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2221                        "http fastcgi record byte: %02Xd", ch);
2222
2223         switch (state) {
2224
2225         case ngx_http_fastcgi_st_version:
2226             if (ch != 1) {
2227                 ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2228                               "upstream sent unsupported FastCGI "
2229                               "protocol version: %d", ch);
2230                 return NGX_ERROR;
2231             }
2232             state = ngx_http_fastcgi_st_type;
2233             break;
2234
2235         case ngx_http_fastcgi_st_type:
2236             switch (ch) {
2237             case NGX_HTTP_FASTCGI_STDOUT:
2238             case NGX_HTTP_FASTCGI_STDERR:
2239             case NGX_HTTP_FASTCGI_END_REQUEST:
2240                  f->type = (ngx_uint_t) ch;
2241                  break;
2242             default:
2243                 ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2244                               "upstream sent invalid FastCGI "
2245                               "record type: %d", ch);
2246                 return NGX_ERROR;
2247
2248             }
2249             state = ngx_http_fastcgi_st_request_id_hi;
2250             break;
2251
2252         /* we support the single request per connection */
2253
2254         case ngx_http_fastcgi_st_request_id_hi:
2255             if (ch != 0) {
2256                 ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2257                               "upstream sent unexpected FastCGI "
2258                               "request id high byte: %d", ch);
2259                 return NGX_ERROR;
2260             }
2261             state = ngx_http_fastcgi_st_request_id_lo;
2262             break;
2263
2264         case ngx_http_fastcgi_st_request_id_lo:
2265             if (ch != 1) {
2266                 ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2267                               "upstream sent unexpected FastCGI "
2268                               "request id low byte: %d", ch);
2269                 return NGX_ERROR;
2270             }
2271             state = ngx_http_fastcgi_st_content_length_hi;
2272             break;
2273
2274         case ngx_http_fastcgi_st_content_length_hi:
2275             f->length = ch << 8;
2276             state = ngx_http_fastcgi_st_content_length_lo;
2277             break;
2278
2279         case ngx_http_fastcgi_st_content_length_lo:
2280             f->length |= (size_t) ch;
2281             state = ngx_http_fastcgi_st_padding_length;
```

```
2282            break;
2283
2284        case ngx_http_fastcgi_st_padding_length:
2285            f->padding = (size_t) ch;
2286            state = ngx_http_fastcgi_st_reserved;
2287            break;
2288
2289        case ngx_http_fastcgi_st_reserved:
2290            state = ngx_http_fastcgi_st_data;
2291
2292            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2293                           "http fastcgi record length: %z", f->length);
2294
2295            f->pos = p + 1;
2296            f->state = state;
2297
2298            return NGX_OK;
2299
2300        /* suppress warning */
2301        case ngx_http_fastcgi_st_data:
2302        case ngx_http_fastcgi_st_padding:
2303            break;
2304        }
2305    }
2306
2307    f->state = state;
2308
2309    return NGX_AGAIN;
2310 }
2311
2312
2313 static void
2314 ngx_http_fastcgi_abort_request(ngx_http_request_t *r)
2315 {
2316    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2317                   "abort http fastcgi request");
2318
2319    return;
2320 }
2321
2322
2323 static void
2324 ngx_http_fastcgi_finalize_request(ngx_http_request_t *r, ngx_int_t rc)
2325 {
2326    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2327                   "finalize http fastcgi request");
2328
2329    return;
2330 }
2331
2332
2333 static ngx_int_t
2334 ngx_http_fastcgi_add_variables(ngx_conf_t *cf)
2335 {
2336    ngx_http_variable_t  *var, *v;
2337
2338    for (v = ngx_http_fastcgi_vars; v->name.len; v++) {
2339        var = ngx_http_add_variable(cf, &v->name, v->flags);
2340        if (var == NULL) {
2341            return NGX_ERROR;
2342        }
2343
2344        var->get_handler = v->get_handler;
2345        var->data = v->data;
2346    }
2347
2348    return NGX_OK;
2349 }
2350
2351
2352 static void *
2353 ngx_http_fastcgi_create_main_conf(ngx_conf_t *cf)
2354 {
2355    ngx_http_fastcgi_main_conf_t  *conf;
2356
2357    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_fastcgi_main_conf_t));
```

```
2358     if (conf == NULL) {
2359         return NULL;
2360     }
2361 
2362 #if (NGX_HTTP_CACHE)
2363     if (ngx_array_init(&conf->caches, cf->pool, 4,
2364                        sizeof(ngx_http_file_cache_t *))
2365         != NGX_OK)
2366     {
2367         return NULL;
2368     }
2369 #endif
2370 
2371     return conf;
2372 }
2373 
2374 
2375 static void *
2376 ngx_http_fastcgi_create_loc_conf(ngx_conf_t *cf)
2377 {
2378     ngx_http_fastcgi_loc_conf_t  *conf;
2379 
2380     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_fastcgi_loc_conf_t));
2381     if (conf == NULL) {
2382         return NULL;
2383     }
2384 
2385     /*
2386      * set by ngx_pcalloc():
2387      *
2388      *     conf->upstream.bufs.num = 0;
2389      *     conf->upstream.ignore_headers = 0;
2390      *     conf->upstream.next_upstream = 0;
2391      *     conf->upstream.cache_zone = NULL;
2392      *     conf->upstream.cache_use_stale = 0;
2393      *     conf->upstream.cache_methods = 0;
2394      *     conf->upstream.temp_path = NULL;
2395      *     conf->upstream.hide_headers_hash = { NULL, 0 };
2396      *     conf->upstream.uri = { 0, NULL };
2397      *     conf->upstream.location = NULL;
2398      *     conf->upstream.store_lengths = NULL;
2399      *     conf->upstream.store_values = NULL;
2400      *
2401      *     conf->index.len = { 0, NULL };
2402      */
2403 
2404     conf->upstream.store = NGX_CONF_UNSET;
2405     conf->upstream.store_access = NGX_CONF_UNSET_UINT;
2406     conf->upstream.next_upstream_tries = NGX_CONF_UNSET_UINT;
2407     conf->upstream.buffering = NGX_CONF_UNSET;
2408     conf->upstream.ignore_client_abort = NGX_CONF_UNSET;
2409     conf->upstream.force_ranges = NGX_CONF_UNSET;
2410 
2411     conf->upstream.local = NGX_CONF_UNSET_PTR;
2412 
2413     conf->upstream.connect_timeout = NGX_CONF_UNSET_MSEC;
2414     conf->upstream.send_timeout = NGX_CONF_UNSET_MSEC;
2415     conf->upstream.read_timeout = NGX_CONF_UNSET_MSEC;
2416     conf->upstream.next_upstream_timeout = NGX_CONF_UNSET_MSEC;
2417 
2418     conf->upstream.send_lowat = NGX_CONF_UNSET_SIZE;
2419     conf->upstream.buffer_size = NGX_CONF_UNSET_SIZE;
2420     conf->upstream.limit_rate = NGX_CONF_UNSET_SIZE;
2421 
2422     conf->upstream.busy_buffers_size_conf = NGX_CONF_UNSET_SIZE;
2423     conf->upstream.max_temp_file_size_conf = NGX_CONF_UNSET_SIZE;
2424     conf->upstream.temp_file_write_size_conf = NGX_CONF_UNSET_SIZE;
2425 
2426     conf->upstream.pass_request_headers = NGX_CONF_UNSET;
2427     conf->upstream.pass_request_body = NGX_CONF_UNSET;
2428 
2429 #if (NGX_HTTP_CACHE)
2430     conf->upstream.cache = NGX_CONF_UNSET;
2431     conf->upstream.cache_min_uses = NGX_CONF_UNSET_UINT;
2432     conf->upstream.cache_bypass = NGX_CONF_UNSET_PTR;
2433     conf->upstream.no_cache = NGX_CONF_UNSET_PTR;
```

```
2434      conf->upstream.cache_valid = NGX_CONF_UNSET_PTR;
2435      conf->upstream.cache_lock = NGX_CONF_UNSET;
2436      conf->upstream.cache_lock_timeout = NGX_CONF_UNSET_MSEC;
2437      conf->upstream.cache_lock_age = NGX_CONF_UNSET_MSEC;
2438      conf->upstream.cache_revalidate = NGX_CONF_UNSET;
2439  #endif
2440
2441      conf->upstream.hide_headers = NGX_CONF_UNSET_PTR;
2442      conf->upstream.pass_headers = NGX_CONF_UNSET_PTR;
2443
2444      conf->upstream.intercept_errors = NGX_CONF_UNSET;
2445
2446      /* "fastcgi_cyclic_temp_file" is disabled */
2447      conf->upstream.cyclic_temp_file = 0;
2448
2449      conf->upstream.change_buffering = 1;
2450
2451      conf->catch_stderr = NGX_CONF_UNSET_PTR;
2452
2453      conf->keep_conn = NGX_CONF_UNSET;
2454
2455      ngx_str_set(&conf->upstream.module, "fastcgi");
2456
2457      return conf;
2458  }
2459
2460
2461  static char *
2462  ngx_http_fastcgi_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
2463  {
2464      ngx_http_fastcgi_loc_conf_t *prev = parent;
2465      ngx_http_fastcgi_loc_conf_t *conf = child;
2466
2467      size_t                       size;
2468      ngx_int_t                    rc;
2469      ngx_hash_init_t              hash;
2470      ngx_http_core_loc_conf_t     *clcf;
2471
2472  #if (NGX_HTTP_CACHE)
2473
2474      if (conf->upstream.store > 0) {
2475          conf->upstream.cache = 0;
2476      }
2477
2478      if (conf->upstream.cache > 0) {
2479          conf->upstream.store = 0;
2480      }
2481
2482  #endif
2483
2484      if (conf->upstream.store == NGX_CONF_UNSET) {
2485          ngx_conf_merge_value(conf->upstream.store,
2486                               prev->upstream.store, 0);
2487
2488          conf->upstream.store_lengths = prev->upstream.store_lengths;
2489          conf->upstream.store_values = prev->upstream.store_values;
2490      }
2491
2492      ngx_conf_merge_uint_value(conf->upstream.store_access,
2493                                prev->upstream.store_access, 0600);
2494
2495      ngx_conf_merge_uint_value(conf->upstream.next_upstream_tries,
2496                                prev->upstream.next_upstream_tries, 0);
2497
2498      ngx_conf_merge_value(conf->upstream.buffering,
2499                                prev->upstream.buffering, 1);
2500
2501      ngx_conf_merge_value(conf->upstream.ignore_client_abort,
2502                                prev->upstream.ignore_client_abort, 0);
2503
2504      ngx_conf_merge_value(conf->upstream.force_ranges,
2505                                prev->upstream.force_ranges, 0);
2506
2507      ngx_conf_merge_ptr_value(conf->upstream.local,
2508                                prev->upstream.local, NULL);
2509
```

```
    ngx_conf_merge_msec_value(conf->upstream.connect_timeout,
                              prev->upstream.connect_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.send_timeout,
                              prev->upstream.send_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.read_timeout,
                              prev->upstream.read_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.next_upstream_timeout,
                              prev->upstream.next_upstream_timeout, 0);

    ngx_conf_merge_size_value(conf->upstream.send_lowat,
                              prev->upstream.send_lowat, 0);

    ngx_conf_merge_size_value(conf->upstream.buffer_size,
                              prev->upstream.buffer_size,
                              (size_t) ngx_pagesize);

    ngx_conf_merge_size_value(conf->upstream.limit_rate,
                              prev->upstream.limit_rate, 0);


    ngx_conf_merge_bufs_value(conf->upstream.bufs, prev->upstream.bufs,
                              8, ngx_pagesize);

    if (conf->upstream.bufs.num < 2) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "there must be at least 2 \"fastcgi_buffers\"");
        return NGX_CONF_ERROR;
    }


    size = conf->upstream.buffer_size;
    if (size < conf->upstream.bufs.size) {
        size = conf->upstream.bufs.size;
    }


    ngx_conf_merge_size_value(conf->upstream.busy_buffers_size_conf,
                              prev->upstream.busy_buffers_size_conf,
                              NGX_CONF_UNSET_SIZE);

    if (conf->upstream.busy_buffers_size_conf == NGX_CONF_UNSET_SIZE) {
        conf->upstream.busy_buffers_size = 2 * size;
    } else {
        conf->upstream.busy_buffers_size =
                                        conf->upstream.busy_buffers_size_conf;
    }

    if (conf->upstream.busy_buffers_size < size) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
             "\"fastcgi_busy_buffers_size\" must be equal to or greater than "
             "the maximum of the value of \"fastcgi_buffer_size\" and "
             "one of the \"fastcgi_buffers\"");

        return NGX_CONF_ERROR;
    }

    if (conf->upstream.busy_buffers_size
        > (conf->upstream.bufs.num - 1) * conf->upstream.bufs.size)
    {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
             "\"fastcgi_busy_buffers_size\" must be less than "
             "the size of all \"fastcgi_buffers\" minus one buffer");

        return NGX_CONF_ERROR;
    }


    ngx_conf_merge_size_value(conf->upstream.temp_file_write_size_conf,
                              prev->upstream.temp_file_write_size_conf,
                              NGX_CONF_UNSET_SIZE);

    if (conf->upstream.temp_file_write_size_conf == NGX_CONF_UNSET_SIZE) {
        conf->upstream.temp_file_write_size = 2 * size;
```

```
2586      } else {
2587          conf->upstream.temp_file_write_size =
2588                                           conf->upstream.temp_file_write_size_conf;
2589      }
2590
2591      if (conf->upstream.temp_file_write_size < size) {
2592          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
2593               "\"fastcgi_temp_file_write_size\" must be equal to or greater "
2594               "than the maximum of the value of \"fastcgi_buffer_size\" and "
2595               "one of the \"fastcgi_buffers\"");
2596
2597          return NGX_CONF_ERROR;
2598      }
2599
2600
2601      ngx_conf_merge_size_value(conf->upstream.max_temp_file_size_conf,
2602                                prev->upstream.max_temp_file_size_conf,
2603                                NGX_CONF_UNSET_SIZE);
2604
2605      if (conf->upstream.max_temp_file_size_conf == NGX_CONF_UNSET_SIZE) {
2606          conf->upstream.max_temp_file_size = 1024 * 1024 * 1024;
2607      } else {
2608          conf->upstream.max_temp_file_size =
2609                                            conf->upstream.max_temp_file_size_conf;
2610      }
2611
2612      if (conf->upstream.max_temp_file_size != 0
2613          && conf->upstream.max_temp_file_size < size)
2614      {
2615          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
2616               "\"fastcgi_max_temp_file_size\" must be equal to zero to disable "
2617               "temporary files usage or must be equal to or greater than "
2618               "the maximum of the value of \"fastcgi_buffer_size\" and "
2619               "one of the \"fastcgi_buffers\"");
2620
2621          return NGX_CONF_ERROR;
2622      }
2623
2624
2625      ngx_conf_merge_bitmask_value(conf->upstream.ignore_headers,
2626                                   prev->upstream.ignore_headers,
2627                                   NGX_CONF_BITMASK_SET);
2628
2629
2630      ngx_conf_merge_bitmask_value(conf->upstream.next_upstream,
2631                                   prev->upstream.next_upstream,
2632                                   (NGX_CONF_BITMASK_SET
2633                                    |NGX_HTTP_UPSTREAM_FT_ERROR
2634                                    |NGX_HTTP_UPSTREAM_FT_TIMEOUT));
2635
2636      if (conf->upstream.next_upstream & NGX_HTTP_UPSTREAM_FT_OFF) {
2637          conf->upstream.next_upstream = NGX_CONF_BITMASK_SET
2638                                         |NGX_HTTP_UPSTREAM_FT_OFF;
2639      }
2640
2641      if (ngx_conf_merge_path_value(cf, &conf->upstream.temp_path,
2642                                    prev->upstream.temp_path,
2643                                    &ngx_http_fastcgi_temp_path)
2644          != NGX_OK)
2645      {
2646          return NGX_CONF_ERROR;
2647      }
2648
2649  #if (NGX_HTTP_CACHE)
2650
2651      if (conf->upstream.cache == NGX_CONF_UNSET) {
2652          ngx_conf_merge_value(conf->upstream.cache,
2653                               prev->upstream.cache, 0);
2654
2655          conf->upstream.cache_zone = prev->upstream.cache_zone;
2656          conf->upstream.cache_value = prev->upstream.cache_value;
2657      }
2658
2659      if (conf->upstream.cache_zone && conf->upstream.cache_zone->data == NULL) {
2660          ngx_shm_zone_t  *shm_zone;
2661
```

```
        shm_zone = conf->upstream.cache_zone;

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"fastcgi_cache\" zone \"%V\" is unknown",
                           &shm_zone->shm.name);

        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_uint_value(conf->upstream.cache_min_uses,
                              prev->upstream.cache_min_uses, 1);

    ngx_conf_merge_bitmask_value(conf->upstream.cache_use_stale,
                              prev->upstream.cache_use_stale,
                              (NGX_CONF_BITMASK_SET
                               |NGX_HTTP_UPSTREAM_FT_OFF));

    if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_OFF) {
        conf->upstream.cache_use_stale = NGX_CONF_BITMASK_SET
                                         |NGX_HTTP_UPSTREAM_FT_OFF;
    }

    if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_ERROR) {
        conf->upstream.cache_use_stale |= NGX_HTTP_UPSTREAM_FT_NOLIVE;
    }

    if (conf->upstream.cache_methods == 0) {
        conf->upstream.cache_methods = prev->upstream.cache_methods;
    }

    conf->upstream.cache_methods |= NGX_HTTP_GET|NGX_HTTP_HEAD;

    ngx_conf_merge_ptr_value(conf->upstream.cache_bypass,
                             prev->upstream.cache_bypass, NULL);

    ngx_conf_merge_ptr_value(conf->upstream.no_cache,
                             prev->upstream.no_cache, NULL);

    ngx_conf_merge_ptr_value(conf->upstream.cache_valid,
                             prev->upstream.cache_valid, NULL);

    if (conf->cache_key.value.data == NULL) {
        conf->cache_key = prev->cache_key;
    }

    if (conf->upstream.cache && conf->cache_key.value.data == NULL) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "no \"fastcgi_cache_key\" for \"fastcgi_cache\"");
    }

    ngx_conf_merge_value(conf->upstream.cache_lock,
                              prev->upstream.cache_lock, 0);

    ngx_conf_merge_msec_value(conf->upstream.cache_lock_timeout,
                              prev->upstream.cache_lock_timeout, 5000);

    ngx_conf_merge_msec_value(conf->upstream.cache_lock_age,
                              prev->upstream.cache_lock_age, 5000);

    ngx_conf_merge_value(conf->upstream.cache_revalidate,
                              prev->upstream.cache_revalidate, 0);

#endif

    ngx_conf_merge_value(conf->upstream.pass_request_headers,
                              prev->upstream.pass_request_headers, 1);
    ngx_conf_merge_value(conf->upstream.pass_request_body,
                              prev->upstream.pass_request_body, 1);

    ngx_conf_merge_value(conf->upstream.intercept_errors,
                              prev->upstream.intercept_errors, 0);

    ngx_conf_merge_ptr_value(conf->catch_stderr, prev->catch_stderr, NULL);

    ngx_conf_merge_value(conf->keep_conn, prev->keep_conn, 0);

```

```
2738
2739        ngx_conf_merge_str_value(conf->index, prev->index, "");
2740
2741        hash.max_size = 512;
2742        hash.bucket_size = ngx_align(64, ngx_cacheline_size);
2743        hash.name = "fastcgi_hide_headers_hash";
2744
2745        if (ngx_http_upstream_hide_headers_hash(cf, &conf->upstream,
2746                 &prev->upstream, ngx_http_fastcgi_hide_headers, &hash)
2747            != NGX_OK)
2748        {
2749            return NGX_CONF_ERROR;
2750        }
2751
2752        clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
2753
2754        if (clcf->noname
2755            && conf->upstream.upstream == NULL && conf->fastcgi_lengths == NULL)
2756        {
2757            conf->upstream.upstream = prev->upstream.upstream;
2758            conf->fastcgi_lengths = prev->fastcgi_lengths;
2759            conf->fastcgi_values = prev->fastcgi_values;
2760        }
2761
2762        if (clcf->lmt_excpt && clcf->handler == NULL
2763            && (conf->upstream.upstream || conf->fastcgi_lengths))
2764        {
2765            clcf->handler = ngx_http_fastcgi_handler;
2766        }
2767
2768    #if (NGX_PCRE)
2769        if (conf->split_regex == NULL) {
2770            conf->split_regex = prev->split_regex;
2771            conf->split_name = prev->split_name;
2772        }
2773    #endif
2774
2775        if (conf->params_source == NULL) {
2776            conf->params = prev->params;
2777    #if (NGX_HTTP_CACHE)
2778            conf->params_cache = prev->params_cache;
2779    #endif
2780            conf->params_source = prev->params_source;
2781        }
2782
2783        rc = ngx_http_fastcgi_init_params(cf, conf, &conf->params, NULL);
2784        if (rc != NGX_OK) {
2785            return NGX_CONF_ERROR;
2786        }
2787
2788    #if (NGX_HTTP_CACHE)
2789
2790        if (conf->upstream.cache) {
2791            rc = ngx_http_fastcgi_init_params(cf, conf, &conf->params_cache,
2792                                              ngx_http_fastcgi_cache_headers);
2793            if (rc != NGX_OK) {
2794                return NGX_CONF_ERROR;
2795            }
2796        }
2797
2798    #endif
2799
2800        return NGX_CONF_OK;
2801    }
2802
2803
2804    static ngx_int_t
2805    ngx_http_fastcgi_init_params(ngx_conf_t *cf, ngx_http_fastcgi_loc_conf_t *conf,
2806        ngx_http_fastcgi_params_t *params, ngx_keyval_t *default_params)
2807    {
2808        u_char                     *p;
2809        size_t                      size;
2810        uintptr_t                  *code;
2811        ngx_uint_t                  i, nsrc;
2812        ngx_array_t                 headers_names, params_merged;
2813        ngx_keyval_t               *h;
```

```
    ngx_hash_key_t               *hk;
    ngx_hash_init_t               hash;
    ngx_http_upstream_param_t    *src, *s;
    ngx_http_script_compile_t     sc;
    ngx_http_script_copy_code_t  *copy;

    if (params->hash.buckets) {
        return NGX_OK;
    }

    if (conf->params_source == NULL && default_params == NULL) {
        params->hash.buckets = (void *) 1;
        return NGX_OK;
    }

    params->lengths = ngx_array_create(cf->pool, 64, 1);
    if (params->lengths == NULL) {
        return NGX_ERROR;
    }

    params->values = ngx_array_create(cf->pool, 512, 1);
    if (params->values == NULL) {
        return NGX_ERROR;
    }

    if (ngx_array_init(&headers_names, cf->temp_pool, 4, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (conf->params_source) {
        src = conf->params_source->elts;
        nsrc = conf->params_source->nelts;

    } else {
        src = NULL;
        nsrc = 0;
    }

    if (default_params) {
        if (ngx_array_init(&params_merged, cf->temp_pool, 4,
                           sizeof(ngx_http_upstream_param_t))
            != NGX_OK)
        {
            return NGX_ERROR;
        }

        for (i = 0; i < nsrc; i++) {

            s = ngx_array_push(&params_merged);
            if (s == NULL) {
                return NGX_ERROR;
            }

            *s = src[i];
        }

        h = default_params;

        while (h->key.len) {

            src = params_merged.elts;
            nsrc = params_merged.nelts;

            for (i = 0; i < nsrc; i++) {
                if (ngx_strcasecmp(h->key.data, src[i].key.data) == 0) {
                    goto next;
                }
            }

            s = ngx_array_push(&params_merged);
            if (s == NULL) {
                return NGX_ERROR;
            }

```

```
            s->key = h->key;
            s->value = h->value;
            s->skip_empty = 1;

        next:

            h++;
        }

        src = params_merged.elts;
        nsrc = params_merged.nelts;
    }

    for (i = 0; i < nsrc; i++) {

        if (src[i].key.len > sizeof("HTTP_") - 1
            && ngx_strncmp(src[i].key.data, "HTTP_", sizeof("HTTP_") - 1) == 0)
        {
            hk = ngx_array_push(&headers_names);
            if (hk == NULL) {
                return NGX_ERROR;
            }

            hk->key.len = src[i].key.len - 5;
            hk->key.data = src[i].key.data + 5;
            hk->key_hash = ngx_hash_key_lc(hk->key.data, hk->key.len);
            hk->value = (void *) 1;

            if (src[i].value.len == 0) {
                continue;
            }
        }

        copy = ngx_array_push_n(params->lengths,
                                sizeof(ngx_http_script_copy_code_t));
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
        copy->len = src[i].key.len;

        copy = ngx_array_push_n(params->lengths,
                                sizeof(ngx_http_script_copy_code_t));
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
        copy->len = src[i].skip_empty;


        size = (sizeof(ngx_http_script_copy_code_t)
                + src[i].key.len + sizeof(uintptr_t) - 1)
               & ~(sizeof(uintptr_t) - 1);

        copy = ngx_array_push_n(params->values, size);
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = ngx_http_script_copy_code;
        copy->len = src[i].key.len;

        p = (u_char *) copy + sizeof(ngx_http_script_copy_code_t);
        ngx_memcpy(p, src[i].key.data, src[i].key.len);


        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

        sc.cf = cf;
        sc.source = &src[i].value;
        sc.flushes = &params->flushes;
        sc.lengths = &params->lengths;
        sc.values = &params->values;

```

```
        if (ngx_http_script_compile(&sc) != NGX_OK) {
            return NGX_ERROR;
        }

        code = ngx_array_push_n(params->lengths, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;


        code = ngx_array_push_n(params->values, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;
    }

    code = ngx_array_push_n(params->lengths, sizeof(uintptr_t));
    if (code == NULL) {
        return NGX_ERROR;
    }

    *code = (uintptr_t) NULL;

    params->number = headers_names.nelts;

    hash.hash = &params->hash;
    hash.key = ngx_hash_key_lc;
    hash.max_size = 512;
    hash.bucket_size = 64;
    hash.name = "fastcgi_params_hash";
    hash.pool = cf->pool;
    hash.temp_pool = NULL;

    return ngx_hash_init(&hash, headers_names.elts, headers_names.nelts);
}


static ngx_int_t
ngx_http_fastcgi_script_name_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    u_char                       *p;
    ngx_http_fastcgi_ctx_t       *f;
    ngx_http_fastcgi_loc_conf_t  *flcf;

    flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);

    f = ngx_http_fastcgi_split(r, flcf);

    if (f == NULL) {
        return NGX_ERROR;
    }

    if (f->script_name.len == 0
        || f->script_name.data[f->script_name.len - 1] != '/')
    {
        v->len = f->script_name.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = f->script_name.data;

        return NGX_OK;
    }

    v->len = f->script_name.len + flcf->index.len;

    v->data = ngx_pnalloc(r->pool, v->len);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

```

```
3042      p = ngx_copy(v->data, f->script_name.data, f->script_name.len);
3043      ngx_memcpy(p, flcf->index.data, flcf->index.len);
3044
3045      return NGX_OK;
3046  }
3047
3048
3049  static ngx_int_t
3050  ngx_http_fastcgi_path_info_variable(ngx_http_request_t *r,
3051      ngx_http_variable_value_t *v, uintptr_t data)
3052  {
3053      ngx_http_fastcgi_ctx_t       *f;
3054      ngx_http_fastcgi_loc_conf_t  *flcf;
3055
3056      flcf = ngx_http_get_module_loc_conf(r, ngx_http_fastcgi_module);
3057
3058      f = ngx_http_fastcgi_split(r, flcf);
3059
3060      if (f == NULL) {
3061          return NGX_ERROR;
3062      }
3063
3064      v->len = f->path_info.len;
3065      v->valid = 1;
3066      v->no_cacheable = 0;
3067      v->not_found = 0;
3068      v->data = f->path_info.data;
3069
3070      return NGX_OK;
3071  }
3072
3073
3074  static ngx_http_fastcgi_ctx_t *
3075  ngx_http_fastcgi_split(ngx_http_request_t *r, ngx_http_fastcgi_loc_conf_t *flcf)
3076  {
3077      ngx_http_fastcgi_ctx_t       *f;
3078  #if (NGX_PCRE)
3079      ngx_int_t                     n;
3080      int                           captures[(1 + 2) * 3];
3081
3082      f = ngx_http_get_module_ctx(r, ngx_http_fastcgi_module);
3083
3084      if (f == NULL) {
3085          f = ngx_pcalloc(r->pool, sizeof(ngx_http_fastcgi_ctx_t));
3086          if (f == NULL) {
3087              return NULL;
3088          }
3089
3090          ngx_http_set_ctx(r, f, ngx_http_fastcgi_module);
3091      }
3092
3093      if (f->script_name.len) {
3094          return f;
3095      }
3096
3097      if (flcf->split_regex == NULL) {
3098          f->script_name = r->uri;
3099          return f;
3100      }
3101
3102      n = ngx_regex_exec(flcf->split_regex, &r->uri, captures, (1 + 2) * 3);
3103
3104      if (n >= 0) { /* match */
3105          f->script_name.len = captures[3] - captures[2];
3106          f->script_name.data = r->uri.data + captures[2];
3107
3108          f->path_info.len = captures[5] - captures[4];
3109          f->path_info.data = r->uri.data + captures[4];
3110
3111          return f;
3112      }
3113
3114      if (n == NGX_REGEX_NO_MATCHED) {
3115          f->script_name = r->uri;
3116          return f;
3117      }
```

```
3118
3119        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
3120                      ngx_regex_exec_n " failed: %i on \"%V\" using \"%V\"",
3121                      n, &r->uri, &flcf->split_name);
3122        return NULL;
3123
3124    #else
3125
3126        f = ngx_http_get_module_ctx(r, ngx_http_fastcgi_module);
3127
3128        if (f == NULL) {
3129            f = ngx_pcalloc(r->pool, sizeof(ngx_http_fastcgi_ctx_t));
3130            if (f == NULL) {
3131                return NULL;
3132            }
3133
3134            ngx_http_set_ctx(r, f, ngx_http_fastcgi_module);
3135        }
3136
3137        f->script_name = r->uri;
3138
3139        return f;
3140
3141    #endif
3142    }
3143
3144
3145    static char *
3146    ngx_http_fastcgi_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3147    {
3148        ngx_http_fastcgi_loc_conf_t *flcf = conf;
3149
3150        ngx_url_t                   u;
3151        ngx_str_t                  *value, *url;
3152        ngx_uint_t                  n;
3153        ngx_http_core_loc_conf_t   *clcf;
3154        ngx_http_script_compile_t   sc;
3155
3156        if (flcf->upstream.upstream || flcf->fastcgi_lengths) {
3157            return "is duplicate";
3158        }
3159
3160        clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
3161
3162        clcf->handler = ngx_http_fastcgi_handler;
3163
3164        if (clcf->name.data[clcf->name.len - 1] == '/') {
3165            clcf->auto_redirect = 1;
3166        }
3167
3168        value = cf->args->elts;
3169
3170        url = &value[1];
3171
3172        n = ngx_http_script_variables_count(url);
3173
3174        if (n) {
3175
3176            ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
3177
3178            sc.cf = cf;
3179            sc.source = url;
3180            sc.lengths = &flcf->fastcgi_lengths;
3181            sc.values = &flcf->fastcgi_values;
3182            sc.variables = n;
3183            sc.complete_lengths = 1;
3184            sc.complete_values = 1;
3185
3186            if (ngx_http_script_compile(&sc) != NGX_OK) {
3187                return NGX_CONF_ERROR;
3188            }
3189
3190            return NGX_CONF_OK;
3191        }
3192
3193        ngx_memzero(&u, sizeof(ngx_url_t));
```

```
3194
3195        u.url = value[1];
3196        u.no_resolve = 1;
3197
3198        flcf->upstream.upstream = ngx_http_upstream_add(cf, &u, 0);
3199        if (flcf->upstream.upstream == NULL) {
3200            return NGX_CONF_ERROR;
3201        }
3202
3203        return NGX_CONF_OK;
3204    }
3205
3206
3207    static char *
3208    ngx_http_fastcgi_split_path_info(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3209    {
3210    #if (NGX_PCRE)
3211        ngx_http_fastcgi_loc_conf_t *flcf = conf;
3212
3213        ngx_str_t            *value;
3214        ngx_regex_compile_t   rc;
3215        u_char                errstr[NGX_MAX_CONF_ERRSTR];
3216
3217        value = cf->args->elts;
3218
3219        flcf->split_name = value[1];
3220
3221        ngx_memzero(&rc, sizeof(ngx_regex_compile_t));
3222
3223        rc.pattern = value[1];
3224        rc.pool = cf->pool;
3225        rc.err.len = NGX_MAX_CONF_ERRSTR;
3226        rc.err.data = errstr;
3227
3228        if (ngx_regex_compile(&rc) != NGX_OK) {
3229            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "%V", &rc.err);
3230            return NGX_CONF_ERROR;
3231        }
3232
3233        if (rc.captures != 2) {
3234            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3235                               "pattern \"%V\" must have 2 captures", &value[1]);
3236            return NGX_CONF_ERROR;
3237        }
3238
3239        flcf->split_regex = rc.regex;
3240
3241        return NGX_CONF_OK;
3242
3243    #else
3244
3245        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3246                           "\"%V\" requires PCRE library", &cmd->name);
3247        return NGX_CONF_ERROR;
3248
3249    #endif
3250    }
3251
3252
3253    static char *
3254    ngx_http_fastcgi_store(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3255    {
3256        ngx_http_fastcgi_loc_conf_t *flcf = conf;
3257
3258        ngx_str_t                  *value;
3259        ngx_http_script_compile_t   sc;
3260
3261        if (flcf->upstream.store != NGX_CONF_UNSET) {
3262            return "is duplicate";
3263        }
3264
3265        value = cf->args->elts;
3266
3267        if (ngx_strcmp(value[1].data, "off") == 0) {
3268            flcf->upstream.store = 0;
3269            return NGX_CONF_OK;
```

```
    }

#if (NGX_HTTP_CACHE)
    if (flcf->upstream.cache > 0) {
        return "is incompatible with \"fastcgi_cache\"";
    }
#endif

    flcf->upstream.store = 1;

    if (ngx_strcmp(value[1].data, "on") == 0) {
        return NGX_CONF_OK;
    }

    /* include the terminating '\0' into script */
    value[1].len++;

    ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

    sc.cf = cf;
    sc.source = &value[1];
    sc.lengths = &flcf->upstream.store_lengths;
    sc.values = &flcf->upstream.store_values;
    sc.variables = ngx_http_script_variables_count(&value[1]);
    sc.complete_lengths = 1;
    sc.complete_values = 1;

    if (ngx_http_script_compile(&sc) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


#if (NGX_HTTP_CACHE)

static char *
ngx_http_fastcgi_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_fastcgi_loc_conf_t *flcf = conf;

    ngx_str_t                         *value;
    ngx_http_complex_value_t           cv;
    ngx_http_compile_complex_value_t   ccv;

    value = cf->args->elts;

    if (flcf->upstream.cache != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    if (ngx_strcmp(value[1].data, "off") == 0) {
        flcf->upstream.cache = 0;
        return NGX_CONF_OK;
    }

    if (flcf->upstream.store > 0) {
        return "is incompatible with \"fastcgi_store\"";
    }

    flcf->upstream.cache = 1;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &cv;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (cv.lengths != NULL) {

        flcf->upstream.cache_value = ngx_palloc(cf->pool,
```

```
                                           sizeof(ngx_http_complex_value_t));
        if (flcf->upstream.cache_value == NULL) {
            return NGX_CONF_ERROR;
        }

        *flcf->upstream.cache_value = cv;

        return NGX_CONF_OK;
    }

    flcf->upstream.cache_zone = ngx_shared_memory_add(cf, &value[1], 0,
                                                      &ngx_http_fastcgi_module);
    if (flcf->upstream.cache_zone == NULL) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_fastcgi_cache_key(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_fastcgi_loc_conf_t *flcf = conf;

    ngx_str_t                         *value;
    ngx_http_compile_complex_value_t   ccv;

    value = cf->args->elts;

    if (flcf->cache_key.value.data) {
        return "is duplicate";
    }

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &flcf->cache_key;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}

#endif


static char *
ngx_http_fastcgi_lowat_check(ngx_conf_t *cf, void *post, void *data)
{
#if (NGX_FREEBSD)
    ssize_t *np = data;

    if ((u_long) *np >= ngx_freebsd_net_inet_tcp_sendspace) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"fastcgi_send_lowat\" must be less than %d "
                           "(sysctl net.inet.tcp.sendspace)",
                           ngx_freebsd_net_inet_tcp_sendspace);

        return NGX_CONF_ERROR;
    }

#elif !(NGX_HAVE_SO_SNDLOWAT)
    ssize_t *np = data;

    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                       "\"fastcgi_send_lowat\" is not supported, ignored");

    *np = 0;

#endif

    return NGX_CONF_OK;
```

```
3422 }
```

One Level Up Top Level

src/http/modules/ngx_http_flv_module.c - nginx-1.7.10

Global variables defined

- ngx_flv_header
- ngx_http_flv_commands
- ngx_http_flv_module
- ngx_http_flv_module_ctx

Functions defined

- ngx_http_flv
- ngx_http_flv_handler

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9  #include <ngx_http.h>
10
11
12 static char *ngx_http_flv(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
13
14 static ngx_command_t  ngx_http_flv_commands[] = {
15
16     { ngx_string("flv"),
17       NGX_HTTP_LOC_CONF|NGX_CONF_NOARGS,
18       ngx_http_flv,
19       0,
20       0,
21       NULL },
22
23       ngx_null_command
24 };
25
26
27 static u_char  ngx_flv_header[] = "FLV\x1\x5\0\0\0\x9\0\0\0\0";
28
29
30 static ngx_http_module_t  ngx_http_flv_module_ctx = {
31     NULL,                          /* preconfiguration */
32     NULL,                          /* postconfiguration */
33
34     NULL,                          /* create main configuration */
35     NULL,                          /* init main configuration */
36
37     NULL,                          /* create server configuration */
38     NULL,                          /* merge server configuration */
39
40     NULL,                          /* create location configuration */
41     NULL                           /* merge location configuration */
42 };
43
44
45 ngx_module_t  ngx_http_flv_module = {
46     NGX_MODULE_V1,
47     &ngx_http_flv_module_ctx,      /* module context */
48     ngx_http_flv_commands,         /* module directives */
```

```
    NGX_HTTP_MODULE,               /* module type */
    NULL,                          /* init master */
    NULL,                          /* init module */
    NULL,                          /* init process */
    NULL,                          /* init thread */
    NULL,                          /* exit thread */
    NULL,                          /* exit process */
    NULL,                          /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_flv_handler(ngx_http_request_t *r)
{
    u_char                    *last;
    off_t                      start, len;
    size_t                     root;
    ngx_int_t                  rc;
    ngx_uint_t                 level, i;
    ngx_str_t                  path, value;
    ngx_log_t                 *log;
    ngx_buf_t                 *b;
    ngx_chain_t                out[2];
    ngx_open_file_info_t       of;
    ngx_http_core_loc_conf_t  *clcf;

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD))) {
        return NGX_HTTP_NOT_ALLOWED;
    }

    if (r->uri.data[r->uri.len - 1] == '/') {
        return NGX_DECLINED;
    }

    rc = ngx_http_discard_request_body(r);

    if (rc != NGX_OK) {
        return rc;
    }

    last = ngx_http_map_uri_to_path(r, &path, &root, 0);
    if (last == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    log = r->connection->log;

    path.len = last - path.data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, log, 0,
                   "http flv filename: \"%V\"", &path);

     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

     ngx_memzero(&of, sizeof(ngx_open_file_info_t));

     of.read_ahead = clcf->read_ahead;
     of.directio = clcf->directio;
     of.valid = clcf->open_file_cache_valid;
     of.min_uses = clcf->open_file_cache_min_uses;
     of.errors = clcf->open_file_cache_errors;
     of.events = clcf->open_file_cache_events;

     if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
         return NGX_HTTP_INTERNAL_SERVER_ERROR;
     }

     if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
         != NGX_OK)
     {
         switch (of.err) {

         case 0:
             return NGX_HTTP_INTERNAL_SERVER_ERROR;

```

```
        case NGX_ENOENT:
        case NGX_ENOTDIR:
        case NGX_ENAMETOOLONG:

            level = NGX_LOG_ERR;
            rc = NGX_HTTP_NOT_FOUND;
            break;

        case NGX_EACCES:
#if (NGX_HAVE_OPENAT)
        case NGX_EMLINK:
        case NGX_ELOOP:
#endif

            level = NGX_LOG_ERR;
            rc = NGX_HTTP_FORBIDDEN;
            break;

        default:

            level = NGX_LOG_CRIT;
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
            break;
        }

        if (rc != NGX_HTTP_NOT_FOUND || clcf->log_not_found) {
            ngx_log_error(level, log, of.err,
                          "%s \"%s\" failed", of.failed, path.data);
        }

        return rc;
    }

    if (!of.is_file) {

        if (ngx_close_file(of.fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                          ngx_close_file_n " \"%s\" failed", path.data);
        }

        return NGX_DECLINED;
    }

    r->root_tested = !r->error_page;

    start = 0;
    len = of.size;
    i = 1;

    if (r->args.len) {

        if (ngx_http_arg(r, (u_char *) "start", 5, &value) == NGX_OK) {

            start = ngx_atoof(value.data, value.len);

            if (start == NGX_ERROR || start >= len) {
                start = 0;
            }

            if (start) {
                len = sizeof(ngx_flv_header) - 1 + len - start;
                i = 0;
            }
        }
    }

    log->action = "sending flv to client";

    r->headers_out.status = NGX_HTTP_OK;
    r->headers_out.content_length_n = len;
    r->headers_out.last_modified_time = of.mtime;

    if (ngx_http_set_etag(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

```

```
201     if (ngx_http_set_content_type(r) != NGX_OK) {
202         return NGX_HTTP_INTERNAL_SERVER_ERROR;
203     }
204 
205     if (i == 0) {
206         b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
207         if (b == NULL) {
208             return NGX_HTTP_INTERNAL_SERVER_ERROR;
209         }
210 
211         b->pos = ngx_flv_header;
212         b->last = ngx_flv_header + sizeof(ngx_flv_header) - 1;
213         b->memory = 1;
214 
215         out[0].buf = b;
216         out[0].next = &out[1];
217     }
218 
219 
220     b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
221     if (b == NULL) {
222         return NGX_HTTP_INTERNAL_SERVER_ERROR;
223     }
224 
225     b->file = ngx_pcalloc(r->pool, sizeof(ngx_file_t));
226     if (b->file == NULL) {
227         return NGX_HTTP_INTERNAL_SERVER_ERROR;
228     }
229 
230     r->allow_ranges = 1;
231 
232     rc = ngx_http_send_header(r);
233 
234     if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
235         return rc;
236     }
237 
238     b->file_pos = start;
239     b->file_last = of.size;
240 
241     b->in_file = b->file_last ? 1: 0;
242     b->last_buf = (r == r->main) ? 1 : 0;
243     b->last_in_chain = 1;
244 
245     b->file->fd = of.fd;
246     b->file->name = path;
247     b->file->log = log;
248     b->file->directio = of.is_directio;
249 
250     out[1].buf = b;
251     out[1].next = NULL;
252 
253     return ngx_http_output_filter(r, &out[i]);
254 }
255 
256 
257 static char *
258 ngx_http_flv(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
259 {
260     ngx_http_core_loc_conf_t  *clcf;
261 
262     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
263     clcf->handler = ngx_http_flv_handler;
264 
265     return NGX_CONF_OK;
266 }
```

One Level Up Top Level

src/http/modules/ngx_http_geo_module.c - nginx-1.7.10

Global variables defined

- ngx_http_geo_commands
- ngx_http_geo_header
- ngx_http_geo_module
- ngx_http_geo_module_ctx

Data types defined

- ngx_http_geo_conf_ctx_t
- ngx_http_geo_ctx_t
- ngx_http_geo_header_t
- ngx_http_geo_high_ranges_t
- ngx_http_geo_range_t
- ngx_http_geo_trees_t
- ngx_http_geo_variable_value_node_t

Functions defined

- ngx_http_geo
- ngx_http_geo_add_proxy
- ngx_http_geo_add_range
- ngx_http_geo_addr
- ngx_http_geo_block
- ngx_http_geo_cidr
- ngx_http_geo_cidr_add
- ngx_http_geo_cidr_value
- ngx_http_geo_cidr_variable
- ngx_http_geo_copy_values
- ngx_http_geo_create_binary_base
- ngx_http_geo_delete_range
- ngx_http_geo_include
- ngx_http_geo_include_binary_base
- ngx_http_geo_range
- ngx_http_geo_range_variable

- ngx_http_geo_real_addr
- ngx_http_geo_value

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_http_variable_value_t       *value;
15      u_short                          start;
16      u_short                          end;
17  } ngx_http_geo_range_t;
18
19
20  typedef struct {
21      ngx_radix_tree_t                *tree;
22  #if (NGX_HAVE_INET6)
23      ngx_radix_tree_t                *tree6;
24  #endif
25  } ngx_http_geo_trees_t;
26
27
28  typedef struct {
29      ngx_http_geo_range_t           **low;
30      ngx_http_variable_value_t       *default_value;
31  } ngx_http_geo_high_ranges_t;
32
33
34  typedef struct {
35      ngx_str_node_t                   sn;
36      ngx_http_variable_value_t       *value;
37      size_t                           offset;
38  } ngx_http_geo_variable_value_node_t;
39
40
41  typedef struct {
42      ngx_http_variable_value_t       *value;
43      ngx_str_t                       *net;
44      ngx_http_geo_high_ranges_t       high;
45      ngx_radix_tree_t                *tree;
46  #if (NGX_HAVE_INET6)
47      ngx_radix_tree_t                *tree6;
48  #endif
49      ngx_rbtree_t                     rbtree;
50      ngx_rbtree_node_t                sentinel;
51      ngx_array_t                     *proxies;
52      ngx_pool_t                      *pool;
53      ngx_pool_t                      *temp_pool;
54
55      size_t                           data_size;
56
57      ngx_str_t                        include_name;
58      ngx_uint_t                       includes;
59      ngx_uint_t                       entries;
60
61      unsigned                         ranges:1;
62      unsigned                         outside_entries:1;
63      unsigned                         allow_binary_include:1;
64      unsigned                         binary_include:1;
65      unsigned                         proxy_recursive:1;
66  } ngx_http_geo_conf_ctx_t;
67
68
```

```
typedef struct {
    union {
        ngx_http_geo_trees_t         trees;
        ngx_http_geo_high_ranges_t   high;
    } u;

    ngx_array_t                     *proxies;
    unsigned                         proxy_recursive:1;

    ngx_int_t                        index;
} ngx_http_geo_ctx_t;


static ngx_int_t ngx_http_geo_addr(ngx_http_request_t *r,
    ngx_http_geo_ctx_t *ctx, ngx_addr_t *addr);
static ngx_int_t ngx_http_geo_real_addr(ngx_http_request_t *r,
    ngx_http_geo_ctx_t *ctx, ngx_addr_t *addr);
static char *ngx_http_geo_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static char *ngx_http_geo(ngx_conf_t *cf, ngx_command_t *dummy, void *conf);
static char *ngx_http_geo_range(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_str_t *value);
static char *ngx_http_geo_add_range(ngx_conf_t *cf,
    ngx_http_geo_conf_ctx_t *ctx, in_addr_t start, in_addr_t end);
static ngx_uint_t ngx_http_geo_delete_range(ngx_conf_t *cf,
    ngx_http_geo_conf_ctx_t *ctx, in_addr_t start, in_addr_t end);
static char *ngx_http_geo_cidr(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_str_t *value);
static char *ngx_http_geo_cidr_add(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_cidr_t *cidr, ngx_str_t *value, ngx_str_t *net);
static ngx_http_variable_value_t *ngx_http_geo_value(ngx_conf_t *cf,
    ngx_http_geo_conf_ctx_t *ctx, ngx_str_t *value);
static char *ngx_http_geo_add_proxy(ngx_conf_t *cf,
    ngx_http_geo_conf_ctx_t *ctx, ngx_cidr_t *cidr);
static ngx_int_t ngx_http_geo_cidr_value(ngx_conf_t *cf, ngx_str_t *net,
    ngx_cidr_t *cidr);
static char *ngx_http_geo_include(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_str_t *name);
static ngx_int_t ngx_http_geo_include_binary_base(ngx_conf_t *cf,
    ngx_http_geo_conf_ctx_t *ctx, ngx_str_t *name);
static void ngx_http_geo_create_binary_base(ngx_http_geo_conf_ctx_t *ctx);
static u_char *ngx_http_geo_copy_values(u_char *base, u_char *p,
    ngx_rbtree_node_t *node, ngx_rbtree_node_t *sentinel);


static ngx_command_t  ngx_http_geo_commands[] = {

    { ngx_string("geo"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_TAKE12,
      ngx_http_geo_block,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_geo_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_geo_module = {
    NGX_MODULE_V1,
    &ngx_http_geo_module_ctx,              /* module context */
    ngx_http_geo_commands,                 /* module directives */
```

```
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


typedef struct {
    u_char    GEORNG[6];
    u_char    version;
    u_char    ptr_size;
    uint32_t  endianness;
    uint32_t  crc32;
} ngx_http_geo_header_t;


static ngx_http_geo_header_t  ngx_http_geo_header = {
    { 'G', 'E', 'O', 'R', 'N', 'G' }, 0, sizeof(void *), 0x12345678, 0
};


/* geo range is AF_INET only */

static ngx_int_t
ngx_http_geo_cidr_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    ngx_http_geo_ctx_t *ctx = (ngx_http_geo_ctx_t *) data;

    in_addr_t                   inaddr;
    ngx_addr_t                  addr;
    struct sockaddr_in         *sin;
    ngx_http_variable_value_t  *vv;
#if (NGX_HAVE_INET6)
    u_char                     *p;
    struct in6_addr            *inaddr6;
#endif

    if (ngx_http_geo_addr(r, ctx, &addr) != NGX_OK) {
        vv = (ngx_http_variable_value_t *)
                  ngx_radix32tree_find(ctx->u.trees.tree, INADDR_NONE);
        goto done;
    }

    switch (addr.sockaddr->sa_family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        inaddr6 = &((struct sockaddr_in6 *) addr.sockaddr)->sin6_addr;
        p = inaddr6->s6_addr;

        if (IN6_IS_ADDR_V4MAPPED(inaddr6)) {
            inaddr = p[12] << 24;
            inaddr += p[13] << 16;
            inaddr += p[14] << 8;
            inaddr += p[15];

            vv = (ngx_http_variable_value_t *)
                      ngx_radix32tree_find(ctx->u.trees.tree, inaddr);

        } else {
            vv = (ngx_http_variable_value_t *)
                      ngx_radix128tree_find(ctx->u.trees.tree6, p);
        }

        break;
#endif

    default: /* AF_INET */
        sin = (struct sockaddr_in *) addr.sockaddr;
        inaddr = ntohl(sin->sin_addr.s_addr);
```

```
221
222         vv = (ngx_http_variable_value_t *)
223                   ngx_radix32tree_find(ctx->u.trees.tree, inaddr);
224
225         break;
226     }
227
228 done:
229
230     *v = *vv;
231
232     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
233                    "http geo: %v", v);
234
235     return NGX_OK;
236 }
237
238
239 static ngx_int_t
240 ngx_http_geo_range_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
241     uintptr_t data)
242 {
243     ngx_http_geo_ctx_t *ctx = (ngx_http_geo_ctx_t *) data;
244
245     in_addr_t              inaddr;
246     ngx_addr_t             addr;
247     ngx_uint_t             n;
248     struct sockaddr_in    *sin;
249     ngx_http_geo_range_t  *range;
250 #if (NGX_HAVE_INET6)
251     u_char                *p;
252     struct in6_addr       *inaddr6;
253 #endif
254
255     *v = *ctx->u.high.default_value;
256
257     if (ngx_http_geo_addr(r, ctx, &addr) == NGX_OK) {
258
259         switch (addr.sockaddr->sa_family) {
260
261 #if (NGX_HAVE_INET6)
262         case AF_INET6:
263             inaddr6 = &((struct sockaddr_in6 *) addr.sockaddr)->sin6_addr;
264
265             if (IN6_IS_ADDR_V4MAPPED(inaddr6)) {
266                 p = inaddr6->s6_addr;
267
268                 inaddr = p[12] << 24;
269                 inaddr += p[13] << 16;
270                 inaddr += p[14] << 8;
271                 inaddr += p[15];
272
273             } else {
274                 inaddr = INADDR_NONE;
275             }
276
277             break;
278 #endif
279
280         default: /* AF_INET */
281             sin = (struct sockaddr_in *) addr.sockaddr;
282             inaddr = ntohl(sin->sin_addr.s_addr);
283             break;
284         }
285
286     } else {
287         inaddr = INADDR_NONE;
288     }
289
290     if (ctx->u.high.low) {
291         range = ctx->u.high.low[inaddr >> 16];
292
293         if (range) {
294             n = inaddr & 0xffff;
295             do {
296                 if (n >= (ngx_uint_t) range->start
```

```
                    && n <= (ngx_uint_t) range->end)
                {
                    *v = *range->value;
                    break;
                }
            } while ((++range)->value);
        }
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http geo: %v", v);

    return NGX_OK;
}


static ngx_int_t
ngx_http_geo_addr(ngx_http_request_t *r, ngx_http_geo_ctx_t *ctx,
    ngx_addr_t *addr)
{
    ngx_array_t  *xfwd;

    if (ngx_http_geo_real_addr(r, ctx, addr) != NGX_OK) {
        return NGX_ERROR;
    }

    xfwd = &r->headers_in.x_forwarded_for;

    if (xfwd->nelts > 0 && ctx->proxies != NULL) {
        (void) ngx_http_get_forwarded_addr(r, addr, xfwd, NULL,
                                           ctx->proxies, ctx->proxy_recursive);
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_geo_real_addr(ngx_http_request_t *r, ngx_http_geo_ctx_t *ctx,
    ngx_addr_t *addr)
{
    ngx_http_variable_value_t  *v;

    if (ctx->index == -1) {
        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http geo started: %V", &r->connection->addr_text);

        addr->sockaddr = r->connection->sockaddr;
        addr->socklen = r->connection->socklen;
        /* addr->name = r->connection->addr_text; */

        return NGX_OK;
    }

    v = ngx_http_get_flushed_variable(r, ctx->index);

    if (v == NULL || v->not_found) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http geo not found");

        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http geo started: %v", v);

    if (ngx_parse_addr(r->pool, addr, v->data, v->len) == NGX_OK) {
        return NGX_OK;
    }

    return NGX_ERROR;
}


static char *
ngx_http_geo_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
```

```
{
    char                    *rv;
    size_t                   len;
    ngx_str_t               *value, name;
    ngx_uint_t               i;
    ngx_conf_t               save;
    ngx_pool_t              *pool;
    ngx_array_t             *a;
    ngx_http_variable_t     *var;
    ngx_http_geo_ctx_t      *geo;
    ngx_http_geo_conf_ctx_t  ctx;
#if (NGX_HAVE_INET6)
    static struct in6_addr   zero;
#endif

    value = cf->args->elts;

    geo = ngx_palloc(cf->pool, sizeof(ngx_http_geo_ctx_t));
    if (geo == NULL) {
        return NGX_CONF_ERROR;
    }

    name = value[1];

    if (name.data[0] != '$') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid variable name \"%V\"", &name);
        return NGX_CONF_ERROR;
    }

    name.len--;
    name.data++;

    if (cf->args->nelts == 3) {

        geo->index = ngx_http_get_variable_index(cf, &name);
        if (geo->index == NGX_ERROR) {
            return NGX_CONF_ERROR;
        }

        name = value[2];

        if (name.data[0] != '$') {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid variable name \"%V\"", &name);
            return NGX_CONF_ERROR;
        }

        name.len--;
        name.data++;

    } else {
        geo->index = -1;
    }

    var = ngx_http_add_variable(cf, &name, NGX_HTTP_VAR_CHANGEABLE);
    if (var == NULL) {
        return NGX_CONF_ERROR;
    }

    pool = ngx_create_pool(NGX_DEFAULT_POOL_SIZE, cf->log);
    if (pool == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memzero(&ctx, sizeof(ngx_http_geo_conf_ctx_t));

    ctx.temp_pool = ngx_create_pool(NGX_DEFAULT_POOL_SIZE, cf->log);
    if (ctx.temp_pool == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_rbtree_init(&ctx.rbtree, &ctx.sentinel, ngx_str_rbtree_insert_value);

    ctx.pool = cf->pool;
    ctx.data_size = sizeof(ngx_http_geo_header_t)
```

```
                + sizeof(ngx_http_variable_value_t)
                + 0x10000 * sizeof(ngx_http_geo_range_t *);
    ctx.allow_binary_include = 1;

    save = *cf;
    cf->pool = pool;
    cf->ctx = &ctx;
    cf->handler = ngx_http_geo;
    cf->handler_conf = conf;

    rv = ngx_conf_parse(cf, NULL);

    *cf = save;

    geo->proxies = ctx.proxies;
    geo->proxy_recursive = ctx.proxy_recursive;

    if (ctx.ranges) {

        if (ctx.high.low && !ctx.binary_include) {
            for (i = 0; i < 0x10000; i++) {
                a = (ngx_array_t *) ctx.high.low[i];

                if (a == NULL || a->nelts == 0) {
                    continue;
                }

                len = a->nelts * sizeof(ngx_http_geo_range_t);

                ctx.high.low[i] = ngx_palloc(cf->pool, len + sizeof(void *));
                if (ctx.high.low[i] == NULL) {
                    return NGX_CONF_ERROR;
                }

                ngx_memcpy(ctx.high.low[i], a->elts, len);
                ctx.high.low[i][a->nelts].value = NULL;
                ctx.data_size += len + sizeof(void *);
            }

            if (ctx.allow_binary_include
                && !ctx.outside_entries
                && ctx.entries > 100000
                && ctx.includes == 1)
            {
                ngx_http_geo_create_binary_base(&ctx);
            }
        }

        if (ctx.high.default_value == NULL) {
            ctx.high.default_value = &ngx_http_variable_null_value;
        }

        geo->u.high = ctx.high;

        var->get_handler = ngx_http_geo_range_variable;
        var->data = (uintptr_t) geo;

        ngx_destroy_pool(ctx.temp_pool);
        ngx_destroy_pool(pool);

    } else {
        if (ctx.tree == NULL) {
            ctx.tree = ngx_radix_tree_create(cf->pool, -1);
            if (ctx.tree == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        geo->u.trees.tree = ctx.tree;

#if (NGX_HAVE_INET6)
        if (ctx.tree6 == NULL) {
            ctx.tree6 = ngx_radix_tree_create(cf->pool, -1);
            if (ctx.tree6 == NULL) {
                return NGX_CONF_ERROR;
            }
```

```
        }

        geo->u.trees.tree6 = ctx.tree6;
#endif

        var->get_handler = ngx_http_geo_cidr_variable;
        var->data = (uintptr_t) geo;

        ngx_destroy_pool(ctx.temp_pool);
        ngx_destroy_pool(pool);

        if (ngx_radix32tree_insert(ctx.tree, 0, 0,
                                   (uintptr_t) &ngx_http_variable_null_value)
            == NGX_ERROR)
        {
            return NGX_CONF_ERROR;
        }

        /* NGX_BUSY is okay (default was set explicitly) */

#if (NGX_HAVE_INET6)
        if (ngx_radix128tree_insert(ctx.tree6, zero.s6_addr, zero.s6_addr,
                                    (uintptr_t) &ngx_http_variable_null_value)
            == NGX_ERROR)
        {
            return NGX_CONF_ERROR;
        }
#endif
    }

    return rv;
}


static char *
ngx_http_geo(ngx_conf_t *cf, ngx_command_t *dummy, void *conf)
{
    char                      *rv;
    ngx_str_t                 *value;
    ngx_cidr_t                 cidr;
    ngx_http_geo_conf_ctx_t  *ctx;

    ctx = cf->ctx;

    value = cf->args->elts;

    if (cf->args->nelts == 1) {

        if (ngx_strcmp(value[0].data, "ranges") == 0) {

            if (ctx->tree
#if (NGX_HAVE_INET6)
                || ctx->tree6
#endif
               )
            {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "the \"ranges\" directive must be "
                                   "the first directive inside \"geo\" block");
                goto failed;
            }

            ctx->ranges = 1;

            rv = NGX_CONF_OK;

            goto done;
        }

        else if (ngx_strcmp(value[0].data, "proxy_recursive") == 0) {
            ctx->proxy_recursive = 1;
            rv = NGX_CONF_OK;
            goto done;
        }
    }

```

```
    if (cf->args->nelts != 2) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid number of the geo parameters");
        goto failed;
    }

    if (ngx_strcmp(value[0].data, "include") == 0) {

        rv = ngx_http_geo_include(cf, ctx, &value[1]);

        goto done;

    } else if (ngx_strcmp(value[0].data, "proxy") == 0) {

        if (ngx_http_geo_cidr_value(cf, &value[1], &cidr) != NGX_OK) {
            goto failed;
        }

        rv = ngx_http_geo_add_proxy(cf, ctx, &cidr);

        goto done;
    }

    if (ctx->ranges) {
        rv = ngx_http_geo_range(cf, ctx, value);

    } else {
        rv = ngx_http_geo_cidr(cf, ctx, value);
    }

done:

    ngx_reset_pool(cf->pool);

    return rv;

failed:

    ngx_reset_pool(cf->pool);

    return NGX_CONF_ERROR;
}


static char *
ngx_http_geo_range(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_str_t *value)
{
    u_char      *p, *last;
    in_addr_t    start, end;
    ngx_str_t   *net;
    ngx_uint_t   del;

    if (ngx_strcmp(value[0].data, "default") == 0) {

        if (ctx->high.default_value) {
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                "duplicate default geo range value: \"%V\", old value: \"%v\"",
                &value[1], ctx->high.default_value);
        }

        ctx->high.default_value = ngx_http_geo_value(cf, ctx, &value[1]);
        if (ctx->high.default_value == NULL) {
            return NGX_CONF_ERROR;
        }

        return NGX_CONF_OK;
    }

    if (ctx->binary_include) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
            "binary geo range base \"%s\" cannot be mixed with usual entries",
            ctx->include_name.data);
        return NGX_CONF_ERROR;
    }

```

```
    if (ctx->high.low == NULL) {
        ctx->high.low = ngx_pcalloc(ctx->pool,
                                    0x10000 * sizeof(ngx_http_geo_range_t *));
        if (ctx->high.low == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    ctx->entries++;
    ctx->outside_entries = 1;

    if (ngx_strcmp(value[0].data, "delete") == 0) {
        net = &value[1];
        del = 1;

    } else {
        net = &value[0];
        del = 0;
    }

    last = net->data + net->len;

    p = ngx_strlchr(net->data, last, '-');

    if (p == NULL) {
        goto invalid;
    }

    start = ngx_inet_addr(net->data, p - net->data);

    if (start == INADDR_NONE) {
        goto invalid;
    }

    start = ntohl(start);

    p++;

    end = ngx_inet_addr(p, last - p);

    if (end == INADDR_NONE) {
        goto invalid;
    }

    end = ntohl(end);

    if (start > end) {
        goto invalid;
    }

    if (del) {
        if (ngx_http_geo_delete_range(cf, ctx, start, end)) {
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                               "no address range \"%V\" to delete", net);
        }

        return NGX_CONF_OK;
    }

    ctx->value = ngx_http_geo_value(cf, ctx, &value[1]);

    if (ctx->value == NULL) {
        return NGX_CONF_ERROR;
    }

    ctx->net = net;

    return ngx_http_geo_add_range(cf, ctx, start, end);

invalid:

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid range \"%V\"", net);

    return NGX_CONF_ERROR;
}
```

```
753
754 /* the add procedure is optimized to add a growing up sequence */
755
756 static char *
757 ngx_http_geo_add_range(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
758     in_addr_t start, in_addr_t end)
759 {
760     in_addr_t              n;
761     ngx_uint_t             h, i, s, e;
762     ngx_array_t           *a;
763     ngx_http_geo_range_t  *range;
764
765     for (n = start; n <= end; n = (n + 0x10000) & 0xffff0000) {
766
767         h = n >> 16;
768
769         if (n == start) {
770             s = n & 0xffff;
771         } else {
772             s = 0;
773         }
774
775         if ((n | 0xffff) > end) {
776             e = end & 0xffff;
777
778         } else {
779             e = 0xffff;
780         }
781
782         a = (ngx_array_t *) ctx->high.low[h];
783
784         if (a == NULL) {
785             a = ngx_array_create(ctx->temp_pool, 64,
786                                  sizeof(ngx_http_geo_range_t));
787             if (a == NULL) {
788                 return NGX_CONF_ERROR;
789             }
790
791             ctx->high.low[h] = (ngx_http_geo_range_t *) a;
792         }
793
794         i = a->nelts;
795         range = a->elts;
796
797         while (i) {
798
799             i--;
800
801             if (e < (ngx_uint_t) range[i].start) {
802                 continue;
803             }
804
805             if (s > (ngx_uint_t) range[i].end) {
806
807                 /* add after the range */
808
809                 range = ngx_array_push(a);
810                 if (range == NULL) {
811                     return NGX_CONF_ERROR;
812                 }
813
814                 range = a->elts;
815
816                 ngx_memmove(&range[i + 2], &range[i + 1],
817                             (a->nelts - 2 - i) * sizeof(ngx_http_geo_range_t));
818
819                 range[i + 1].start = (u_short) s;
820                 range[i + 1].end = (u_short) e;
821                 range[i + 1].value = ctx->value;
822
823                 goto next;
824             }
825
826             if (s == (ngx_uint_t) range[i].start
827                 && e == (ngx_uint_t) range[i].end)
828             {
```

```
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                "duplicate range \"%V\", value: \"%v\", old value: \"%v\"",
                ctx->net, ctx->value, range[i].value);

            range[i].value = ctx->value;

            goto next;
        }

        if (s > (ngx_uint_t) range[i].start
            && e < (ngx_uint_t) range[i].end)
        {
            /* split the range and insert the new one */

            range = ngx_array_push(a);
            if (range == NULL) {
                return NGX_CONF_ERROR;
            }

            range = ngx_array_push(a);
            if (range == NULL) {
                return NGX_CONF_ERROR;
            }

            range = a->elts;

            ngx_memmove(&range[i + 3], &range[i + 1],
                        (a->nelts - 3 - i) * sizeof(ngx_http_geo_range_t));

            range[i + 2].start = (u_short) (e + 1);
            range[i + 2].end = range[i].end;
            range[i + 2].value = range[i].value;

            range[i + 1].start = (u_short) s;
            range[i + 1].end = (u_short) e;
            range[i + 1].value = ctx->value;

            range[i].end = (u_short) (s - 1);

            goto next;
        }

        if (s == (ngx_uint_t) range[i].start
            && e < (ngx_uint_t) range[i].end)
        {
            /* shift the range start and insert the new range */

            range = ngx_array_push(a);
            if (range == NULL) {
                return NGX_CONF_ERROR;
            }

            range = a->elts;

            ngx_memmove(&range[i + 1], &range[i],
                        (a->nelts - 1 - i) * sizeof(ngx_http_geo_range_t));

            range[i + 1].start = (u_short) (e + 1);

            range[i].start = (u_short) s;
            range[i].end = (u_short) e;
            range[i].value = ctx->value;

            goto next;
        }

        if (s > (ngx_uint_t) range[i].start
            && e == (ngx_uint_t) range[i].end)
        {
            /* shift the range end and insert the new range */

            range = ngx_array_push(a);
            if (range == NULL) {
                return NGX_CONF_ERROR;
            }

```

```
                range = a->elts;

                ngx_memmove(&range[i + 2], &range[i + 1],
                            (a->nelts - 2 - i) * sizeof(ngx_http_geo_range_t));

                range[i + 1].start = (u_short) s;
                range[i + 1].end = (u_short) e;
                range[i + 1].value = ctx->value;

                range[i].end = (u_short) (s - 1);

                goto next;
            }

            s = (ngx_uint_t) range[i].start;
            e = (ngx_uint_t) range[i].end;

            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "range \"%V\" overlaps \"%d.%d.%d.%d-%d.%d.%d.%d\"",
                               ctx->net,
                               h >> 8, h & 0xff, s >> 8, s & 0xff,
                               h >> 8, h & 0xff, e >> 8, e & 0xff);

            return NGX_CONF_ERROR;
        }

        /* add the first range */

        range = ngx_array_push(a);
        if (range == NULL) {
            return NGX_CONF_ERROR;
        }

        range->start = (u_short) s;
        range->end = (u_short) e;
        range->value = ctx->value;

    next:

        continue;
    }

    return NGX_CONF_OK;
}


static ngx_uint_t
ngx_http_geo_delete_range(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    in_addr_t start, in_addr_t end)
{
    in_addr_t              n;
    ngx_uint_t             h, i, s, e, warn;
    ngx_array_t           *a;
    ngx_http_geo_range_t  *range;

    warn = 0;

    for (n = start; n <= end; n += 0x10000) {

        h = n >> 16;

        if (n == start) {
            s = n & 0xffff;
        } else {
            s = 0;
        }

        if ((n | 0xffff) > end) {
            e = end & 0xffff;

        } else {
            e = 0xffff;
        }

        a = (ngx_array_t *) ctx->high.low[h];

```

```
        if (a == NULL) {
            warn = 1;
            continue;
        }

        range = a->elts;
        for (i = 0; i < a->nelts; i++) {

            if (s == (ngx_uint_t) range[i].start
                && e == (ngx_uint_t) range[i].end)
            {
                ngx_memmove(&range[i], &range[i + 1],
                            (a->nelts - 1 - i) * sizeof(ngx_http_geo_range_t));

                a->nelts--;

                break;
            }

             if (s != (ngx_uint_t) range[i].start
                 && e != (ngx_uint_t) range[i].end)
             {
                 continue;
             }

             warn = 1;
         }
    }

    return warn;
}


static char *
ngx_http_geo_cidr(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_str_t *value)
{
    char        *rv;
    ngx_int_t    rc, del;
    ngx_str_t   *net;
    ngx_cidr_t   cidr;

    if (ctx->tree == NULL) {
        ctx->tree = ngx_radix_tree_create(ctx->pool, -1);
        if (ctx->tree == NULL) {
            return NGX_CONF_ERROR;
        }
    }

#if (NGX_HAVE_INET6)
    if (ctx->tree6 == NULL) {
        ctx->tree6 = ngx_radix_tree_create(ctx->pool, -1);
        if (ctx->tree6 == NULL) {
            return NGX_CONF_ERROR;
        }
    }
#endif

    if (ngx_strcmp(value[0].data, "default") == 0) {
        cidr.family = AF_INET;
        cidr.u.in.addr = 0;
        cidr.u.in.mask = 0;

        rv = ngx_http_geo_cidr_add(cf, ctx, &cidr, &value[1], &value[0]);

        if (rv != NGX_CONF_OK) {
            return rv;
        }

#if (NGX_HAVE_INET6)
        cidr.family = AF_INET6;
        ngx_memzero(&cidr.u.in6, sizeof(ngx_in6_cidr_t));

        rv = ngx_http_geo_cidr_add(cf, ctx, &cidr, &value[1], &value[0]);

        if (rv != NGX_CONF_OK) {
```

```
            return rv;
        }
#endif

        return NGX_CONF_OK;
    }

    if (ngx_strcmp(value[0].data, "delete") == 0) {
        net = &value[1];
        del = 1;

    } else {
        net = &value[0];
        del = 0;
    }

    if (ngx_http_geo_cidr_value(cf, net, &cidr) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (cidr.family == AF_INET) {
        cidr.u.in.addr = ntohl(cidr.u.in.addr);
        cidr.u.in.mask = ntohl(cidr.u.in.mask);
    }

    if (del) {
        switch (cidr.family) {

#if (NGX_HAVE_INET6)
        case AF_INET6:
            rc = ngx_radix128tree_delete(ctx->tree6,
                                         cidr.u.in6.addr.s6_addr,
                                         cidr.u.in6.mask.s6_addr);
            break;
#endif

        default: /* AF_INET */
            rc = ngx_radix32tree_delete(ctx->tree, cidr.u.in.addr,
                                        cidr.u.in.mask);
            break;
        }

        if (rc != NGX_OK) {
            ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                               "no network \"%V\" to delete", net);
        }

        return NGX_CONF_OK;
    }

    return ngx_http_geo_cidr_add(cf, ctx, &cidr, &value[1], net);
}


static char *
ngx_http_geo_cidr_add(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_cidr_t *cidr, ngx_str_t *value, ngx_str_t *net)
{
    ngx_int_t                   rc;
    ngx_http_variable_value_t  *val, *old;

    val = ngx_http_geo_value(cf, ctx, value);

    if (val == NULL) {
        return NGX_CONF_ERROR;
    }

    switch (cidr->family) {

#if (NGX_HAVE_INET6)
    case AF_INET6:
        rc = ngx_radix128tree_insert(ctx->tree6, cidr->u.in6.addr.s6_addr,
                                     cidr->u.in6.mask.s6_addr,
                                     (uintptr_t) val);

        if (rc == NGX_OK) {
```

```
            return NGX_CONF_OK;
        }

        if (rc == NGX_ERROR) {
            return NGX_CONF_ERROR;
        }

        /* rc == NGX_BUSY */

        old = (ngx_http_variable_value_t *)
                  ngx_radix128tree_find(ctx->tree6,
                                        cidr->u.in6.addr.s6_addr);

        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
              "duplicate network \"%V\", value: \"%v\", old value: \"%v\"",
              net, val, old);

        rc = ngx_radix128tree_delete(ctx->tree6,
                                     cidr->u.in6.addr.s6_addr,
                                     cidr->u.in6.mask.s6_addr);

        if (rc == NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid radix tree");
            return NGX_CONF_ERROR;
        }

        rc = ngx_radix128tree_insert(ctx->tree6, cidr->u.in6.addr.s6_addr,
                                     cidr->u.in6.mask.s6_addr,
                                     (uintptr_t) val);

        break;
#endif

    default: /* AF_INET */
        rc = ngx_radix32tree_insert(ctx->tree, cidr->u.in.addr,
                                    cidr->u.in.mask, (uintptr_t) val);

        if (rc == NGX_OK) {
            return NGX_CONF_OK;
        }

        if (rc == NGX_ERROR) {
            return NGX_CONF_ERROR;
        }

        /* rc == NGX_BUSY */

        old = (ngx_http_variable_value_t *)
                  ngx_radix32tree_find(ctx->tree, cidr->u.in.addr);

        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
              "duplicate network \"%V\", value: \"%v\", old value: \"%v\"",
              net, val, old);

        rc = ngx_radix32tree_delete(ctx->tree,
                                    cidr->u.in.addr, cidr->u.in.mask);

        if (rc == NGX_ERROR) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid radix tree");
            return NGX_CONF_ERROR;
        }

        rc = ngx_radix32tree_insert(ctx->tree, cidr->u.in.addr,
                                    cidr->u.in.mask, (uintptr_t) val);

        break;
    }

    if (rc == NGX_OK) {
        return NGX_CONF_OK;
    }

    return NGX_CONF_ERROR;
}
```

```
1209 static ngx_http_variable_value_t *
1210 ngx_http_geo_value(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
1211     ngx_str_t *value)
1212 {
1213     uint32_t                             hash;
1214     ngx_http_variable_value_t           *val;
1215     ngx_http_geo_variable_value_node_t  *gvvn;
1216 
1217     hash = ngx_crc32_long(value->data, value->len);
1218 
1219     gvvn = (ngx_http_geo_variable_value_node_t *)
1220                 ngx_str_rbtree_lookup(&ctx->rbtree, value, hash);
1221 
1222     if (gvvn) {
1223         return gvvn->value;
1224     }
1225 
1226     val = ngx_palloc(ctx->pool, sizeof(ngx_http_variable_value_t));
1227     if (val == NULL) {
1228         return NULL;
1229     }
1230 
1231     val->len = value->len;
1232     val->data = ngx_pstrdup(ctx->pool, value);
1233     if (val->data == NULL) {
1234         return NULL;
1235     }
1236 
1237     val->valid = 1;
1238     val->no_cacheable = 0;
1239     val->not_found = 0;
1240 
1241     gvvn = ngx_palloc(ctx->temp_pool,
1242                       sizeof(ngx_http_geo_variable_value_node_t));
1243     if (gvvn == NULL) {
1244         return NULL;
1245     }
1246 
1247     gvvn->sn.node.key = hash;
1248     gvvn->sn.str.len = val->len;
1249     gvvn->sn.str.data = val->data;
1250     gvvn->value = val;
1251     gvvn->offset = 0;
1252 
1253     ngx_rbtree_insert(&ctx->rbtree, &gvvn->sn.node);
1254 
1255     ctx->data_size += ngx_align(sizeof(ngx_http_variable_value_t) + value->len,
1256                                 sizeof(void *));
1257 
1258     return val;
1259 }
1260 
1261 
1262 static char *
1263 ngx_http_geo_add_proxy(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
1264     ngx_cidr_t *cidr)
1265 {
1266     ngx_cidr_t  *c;
1267 
1268     if (ctx->proxies == NULL) {
1269         ctx->proxies = ngx_array_create(ctx->pool, 4, sizeof(ngx_cidr_t));
1270         if (ctx->proxies == NULL) {
1271             return NGX_CONF_ERROR;
1272         }
1273     }
1274 
1275     c = ngx_array_push(ctx->proxies);
1276     if (c == NULL) {
1277         return NGX_CONF_ERROR;
1278     }
1279 
1280     *c = *cidr;
1281 
1282     return NGX_CONF_OK;
1283 }
1284 
```

```
1285
1286 static ngx_int_t
1287 ngx_http_geo_cidr_value(ngx_conf_t *cf, ngx_str_t *net, ngx_cidr_t *cidr)
1288 {
1289     ngx_int_t  rc;
1290
1291     if (ngx_strcmp(net->data, "255.255.255.255") == 0) {
1292         cidr->family = AF_INET;
1293         cidr->u.in.addr = 0xffffffff;
1294         cidr->u.in.mask = 0xffffffff;
1295
1296         return NGX_OK;
1297     }
1298
1299     rc = ngx_ptocidr(net, cidr);
1300
1301     if (rc == NGX_ERROR) {
1302         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid network \"%V\"", net);
1303         return NGX_ERROR;
1304     }
1305
1306     if (rc == NGX_DONE) {
1307         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1308                            "low address bits of %V are meaningless", net);
1309     }
1310
1311     return NGX_OK;
1312 }
1313
1314
1315 static char *
1316 ngx_http_geo_include(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
1317     ngx_str_t *name)
1318 {
1319     char       *rv;
1320     ngx_str_t   file;
1321
1322     file.len = name->len + 4;
1323     file.data = ngx_pnalloc(ctx->temp_pool, name->len + 5);
1324     if (file.data == NULL) {
1325         return NGX_CONF_ERROR;
1326     }
1327
1328     ngx_sprintf(file.data, "%V.bin%Z", name);
1329
1330     if (ngx_conf_full_name(cf->cycle, &file, 1) != NGX_OK) {
1331         return NGX_CONF_ERROR;
1332     }
1333
1334     if (ctx->ranges) {
1335         ngx_log_debug1(NGX_LOG_DEBUG_CORE, cf->log, 0, "include %s", file.data);
1336
1337         switch (ngx_http_geo_include_binary_base(cf, ctx, &file)) {
1338         case NGX_OK:
1339             return NGX_CONF_OK;
1340         case NGX_ERROR:
1341             return NGX_CONF_ERROR;
1342         default:
1343             break;
1344         }
1345     }
1346
1347     file.len -= 4;
1348     file.data[file.len] = '\0';
1349
1350     ctx->include_name = file;
1351
1352     if (ctx->outside_entries) {
1353         ctx->allow_binary_include = 0;
1354     }
1355
1356     ngx_log_debug1(NGX_LOG_DEBUG_CORE, cf->log, 0, "include %s", file.data);
1357
1358     rv = ngx_conf_parse(cf, &file);
1359
1360     ctx->includes++;
```

```
    ctx->outside_entries = 0;

    return rv;
}


static ngx_int_t
ngx_http_geo_include_binary_base(ngx_conf_t *cf, ngx_http_geo_conf_ctx_t *ctx,
    ngx_str_t *name)
{
    u_char                     *base, ch;
    time_t                      mtime;
    size_t                      size, len;
    ssize_t                     n;
    uint32_t                    crc32;
    ngx_err_t                   err;
    ngx_int_t                   rc;
    ngx_uint_t                  i;
    ngx_file_t                  file;
    ngx_file_info_t             fi;
    ngx_http_geo_range_t       *range, **ranges;
    ngx_http_geo_header_t      *header;
    ngx_http_variable_value_t  *vv;

    ngx_memzero(&file, sizeof(ngx_file_t));
    file.name = *name;
    file.log = cf->log;

    file.fd = ngx_open_file(name->data, NGX_FILE_RDONLY, 0, 0);
    if (file.fd == NGX_INVALID_FILE) {
        err = ngx_errno;
        if (err != NGX_ENOENT) {
            ngx_conf_log_error(NGX_LOG_CRIT, cf, err,
                               ngx_open_file_n " \"%s\" failed", name->data);
        }
        return NGX_DECLINED;
    }

    if (ctx->outside_entries) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
            "binary geo range base \"%s\" cannot be mixed with usual entries",
            name->data);
        rc = NGX_ERROR;
        goto done;
    }

    if (ctx->binary_include) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
            "second binary geo range base \"%s\" cannot be mixed with \"%s\"",
            name->data, ctx->include_name.data);
        rc = NGX_ERROR;
        goto done;
    }

    if (ngx_fd_info(file.fd, &fi) == NGX_FILE_ERROR) {
        ngx_conf_log_error(NGX_LOG_CRIT, cf, ngx_errno,
                           ngx_fd_info_n " \"%s\" failed", name->data);
        goto failed;
    }

    size = (size_t) ngx_file_size(&fi);
    mtime = ngx_file_mtime(&fi);

    ch = name->data[name->len - 4];
    name->data[name->len - 4] = '\0';

    if (ngx_file_info(name->data, &fi) == NGX_FILE_ERROR) {
        ngx_conf_log_error(NGX_LOG_CRIT, cf, ngx_errno,
                           ngx_file_info_n " \"%s\" failed", name->data);
        goto failed;
    }

    name->data[name->len - 4] = ch;

    if (mtime < ngx_file_mtime(&fi)) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
```

```
                          "stale binary geo range base \"%s\"", name->data);
        goto failed;
    }

    base = ngx_palloc(ctx->pool, size);
    if (base == NULL) {
        goto failed;
    }

    n = ngx_read_file(&file, base, size, 0);

    if (n == NGX_ERROR) {
        ngx_conf_log_error(NGX_LOG_CRIT, cf, ngx_errno,
                           ngx_read_file_n " \"%s\" failed", name->data);
        goto failed;
    }

    if ((size_t) n != size) {
        ngx_conf_log_error(NGX_LOG_CRIT, cf, 0,
            ngx_read_file_n " \"%s\" returned only %z bytes instead of %z",
            name->data, n, size);
        goto failed;
    }

    header = (ngx_http_geo_header_t *) base;

    if (size < 16 || ngx_memcmp(&ngx_http_geo_header, header, 12) != 0) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
             "incompatible binary geo range base \"%s\"", name->data);
        goto failed;
    }

    ngx_crc32_init(crc32);

    vv = (ngx_http_variable_value_t *) (base + sizeof(ngx_http_geo_header_t));

    while (vv->data) {
        len = ngx_align(sizeof(ngx_http_variable_value_t) + vv->len,
                        sizeof(void *));
        ngx_crc32_update(&crc32, (u_char *) vv, len);
        vv->data += (size_t) base;
        vv = (ngx_http_variable_value_t *) ((u_char *) vv + len);
    }
    ngx_crc32_update(&crc32, (u_char *) vv, sizeof(ngx_http_variable_value_t));
    vv++;

    ranges = (ngx_http_geo_range_t **) vv;

    for (i = 0; i < 0x10000; i++) {
        ngx_crc32_update(&crc32, (u_char *) &ranges[i], sizeof(void *));
        if (ranges[i]) {
            ranges[i] = (ngx_http_geo_range_t *)
                            ((u_char *) ranges[i] + (size_t) base);
        }
    }

    range = (ngx_http_geo_range_t *) &ranges[0x10000];

    while ((u_char *) range < base + size) {
        while (range->value) {
            ngx_crc32_update(&crc32, (u_char *) range,
                             sizeof(ngx_http_geo_range_t));
            range->value = (ngx_http_variable_value_t *)
                               ((u_char *) range->value + (size_t) base);
            range++;
        }
        ngx_crc32_update(&crc32, (u_char *) range, sizeof(void *));
        range = (ngx_http_geo_range_t *) ((u_char *) range + sizeof(void *));
    }

    ngx_crc32_final(crc32);

    if (crc32 != header->crc32) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                  "CRC32 mismatch in binary geo range base \"%s\"", name->data);
        goto failed;
```

```
1513      }
1514
1515      ngx_conf_log_error(NGX_LOG_NOTICE, cf, 0,
1516                         "using binary geo range base \"%s\"", name->data);
1517
1518      ctx->include_name = *name;
1519      ctx->binary_include = 1;
1520      ctx->high.low = ranges;
1521      rc = NGX_OK;
1522
1523      goto done;
1524
1525  failed:
1526
1527      rc = NGX_DECLINED;
1528
1529  done:
1530
1531      if (ngx_close_file(file.fd) == NGX_FILE_ERROR) {
1532          ngx_log_error(NGX_LOG_ALERT, cf->log, ngx_errno,
1533                        ngx_close_file_n " \"%s\" failed", name->data);
1534      }
1535
1536      return rc;
1537  }
1538
1539
1540  static void
1541  ngx_http_geo_create_binary_base(ngx_http_geo_conf_ctx_t *ctx)
1542  {
1543      u_char                              *p;
1544      uint32_t                             hash;
1545      ngx_str_t                            s;
1546      ngx_uint_t                           i;
1547      ngx_file_mapping_t                   fm;
1548      ngx_http_geo_range_t                *r, *range, **ranges;
1549      ngx_http_geo_header_t               *header;
1550      ngx_http_geo_variable_value_node_t  *gvvn;
1551
1552      fm.name = ngx_pnalloc(ctx->temp_pool, ctx->include_name.len + 5);
1553      if (fm.name == NULL) {
1554          return;
1555      }
1556
1557      ngx_sprintf(fm.name, "%V.bin%Z", &ctx->include_name);
1558
1559      fm.size = ctx->data_size;
1560      fm.log = ctx->pool->log;
1561
1562      ngx_log_error(NGX_LOG_NOTICE, fm.log, 0,
1563                    "creating binary geo range base \"%s\"", fm.name);
1564
1565      if (ngx_create_file_mapping(&fm) != NGX_OK) {
1566          return;
1567      }
1568
1569      p = ngx_cpymem(fm.addr, &ngx_http_geo_header,
1570                     sizeof(ngx_http_geo_header_t));
1571
1572      p = ngx_http_geo_copy_values(fm.addr, p, ctx->rbtree.root,
1573                                   ctx->rbtree.sentinel);
1574
1575      p += sizeof(ngx_http_variable_value_t);
1576
1577      ranges = (ngx_http_geo_range_t **) p;
1578
1579      p += 0x10000 * sizeof(ngx_http_geo_range_t *);
1580
1581      for (i = 0; i < 0x10000; i++) {
1582          r = ctx->high.low[i];
1583          if (r == NULL) {
1584              continue;
1585          }
1586
1587          range = (ngx_http_geo_range_t *) p;
1588          ranges[i] = (ngx_http_geo_range_t *) (p - (u_char *) fm.addr);
```

```
1589
1590            do {
1591                s.len = r->value->len;
1592                s.data = r->value->data;
1593                hash = ngx_crc32_long(s.data, s.len);
1594                gvvn = (ngx_http_geo_variable_value_node_t *)
1595                            ngx_str_rbtree_lookup(&ctx->rbtree, &s, hash);
1596
1597                range->value = (ngx_http_variable_value_t *) gvvn->offset;
1598                range->start = r->start;
1599                range->end = r->end;
1600                range++;
1601
1602            } while ((++r)->value);
1603
1604            range->value = NULL;
1605
1606            p = (u_char *) range + sizeof(void *);
1607        }
1608
1609        header = fm.addr;
1610        header->crc32 = ngx_crc32_long((u_char *) fm.addr
1611                                       + sizeof(ngx_http_geo_header_t),
1612                                       fm.size - sizeof(ngx_http_geo_header_t));
1613
1614        ngx_close_file_mapping(&fm);
1615    }
1616
1617
1618    static u_char *
1619    ngx_http_geo_copy_values(u_char *base, u_char *p, ngx_rbtree_node_t *node,
1620        ngx_rbtree_node_t *sentinel)
1621    {
1622        ngx_http_variable_value_t           *vv;
1623        ngx_http_geo_variable_value_node_t  *gvvn;
1624
1625        if (node == sentinel) {
1626            return p;
1627        }
1628
1629        gvvn = (ngx_http_geo_variable_value_node_t *) node;
1630        gvvn->offset = p - base;
1631
1632        vv = (ngx_http_variable_value_t *) p;
1633        *vv = *gvvn->value;
1634        p += sizeof(ngx_http_variable_value_t);
1635        vv->data = (u_char *) (p - base);
1636
1637        p = ngx_cpymem(p, gvvn->sn.str.data, gvvn->sn.str.len);
1638
1639        p = ngx_align_ptr(p, sizeof(void *));
1640
1641        p = ngx_http_geo_copy_values(base, p, node->left, sentinel);
1642
1643        return ngx_http_geo_copy_values(base, p, node->right, sentinel);
1644    }
```

One Level Up Top Level

src/core/ngx_radix_tree.h - nginx-1.7.10

Data types defined

- ngx_radix_node_s
- ngx_radix_node_t
- ngx_radix_tree_t

Macros defined

- NGX_RADIX_NO_VALUE
- _NGX_RADIX_TREE_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_RADIX_TREE_H_INCLUDED_
9  #define _NGX_RADIX_TREE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  #define NGX_RADIX_NO_VALUE   (uintptr_t) -1
17
18  typedef struct ngx_radix_node_s  ngx_radix_node_t;
19
20  struct ngx_radix_node_s {
21      ngx_radix_node_t  *right;
22      ngx_radix_node_t  *left;
23      ngx_radix_node_t  *parent;
24      uintptr_t          value;
25  };
26
27
28  typedef struct {
29      ngx_radix_node_t  *root;
30      ngx_pool_t        *pool;
31      ngx_radix_node_t  *free;
32      char              *start;
33      size_t             size;
34  } ngx_radix_tree_t;
35
36
37  ngx_radix_tree_t *ngx_radix_tree_create(ngx_pool_t *pool,
38      ngx_int_t preallocate);
39
40  ngx_int_t ngx_radix32tree_insert(ngx_radix_tree_t *tree,
41      uint32_t key, uint32_t mask, uintptr_t value);
42  ngx_int_t ngx_radix32tree_delete(ngx_radix_tree_t *tree,
43      uint32_t key, uint32_t mask);
44  uintptr_t ngx_radix32tree_find(ngx_radix_tree_t *tree, uint32_t key);
45
46  #if (NGX_HAVE_INET6)
47  ngx_int_t ngx_radix128tree_insert(ngx_radix_tree_t *tree,
48      u_char *key, u_char *mask, uintptr_t value);
49  ngx_int_t ngx_radix128tree_delete(ngx_radix_tree_t *tree,
50      u_char *key, u_char *mask);
```

```
51 uintptr_t ngx_radix128tree_find(ngx_radix_tree_t *tree, u_char *key);
52 #endif
53
54
55 #endif /* _NGX_RADIX_TREE_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_radix_tree.c - nginx-1.7.10

Functions defined

- ngx_radix128tree_delete
- ngx_radix128tree_find
- ngx_radix128tree_insert
- ngx_radix32tree_delete
- ngx_radix32tree_find
- ngx_radix32tree_insert
- ngx_radix_alloc
- ngx_radix_tree_create

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11
12 static ngx_radix_node_t *ngx_radix_alloc(ngx_radix_tree_t *tree);
13
14
15 ngx_radix_tree_t *
16 ngx_radix_tree_create(ngx_pool_t *pool, ngx_int_t preallocate)
17 {
18     uint32_t           key, mask, inc;
19     ngx_radix_tree_t  *tree;
20
21     tree = ngx_palloc(pool, sizeof(ngx_radix_tree_t));
22     if (tree == NULL) {
23         return NULL;
24     }
25
26     tree->pool = pool;
27     tree->free = NULL;
28     tree->start = NULL;
29     tree->size = 0;
30
31     tree->root = ngx_radix_alloc(tree);
32     if (tree->root == NULL) {
33         return NULL;
34     }
35
36     tree->root->right = NULL;
37     tree->root->left = NULL;
38     tree->root->parent = NULL;
39     tree->root->value = NGX_RADIX_NO_VALUE;
40
41     if (preallocate == 0) {
42         return tree;
43     }
44
45     /*
46      * Preallocation of first nodes : 0, 1, 00, 01, 10, 11, 000, 001, etc.
47      * increases TLB hits even if for first lookup iterations.
```

```
     * On 32-bit platforms the 7 preallocated bits takes continuous 4K,
     * 8 - 8K, 9 - 16K, etc.  On 64-bit platforms the 6 preallocated bits
     * takes continuous 4K, 7 - 8K, 8 - 16K, etc.  There is no sense to
     * to preallocate more than one page, because further preallocation
     * distributes the only bit per page.  Instead, a random insertion
     * may distribute several bits per page.
     *
     * Thus, by default we preallocate maximum
     *     6 bits on amd64 (64-bit platform and 4K pages)
     *     7 bits on i386 (32-bit platform and 4K pages)
     *     7 bits on sparc64 in 64-bit mode (8K pages)
     *     8 bits on sparc64 in 32-bit mode (8K pages)
     */

    if (preallocate == -1) {
        switch (ngx_pagesize / sizeof(ngx_radix_node_t)) {

        /* amd64 */
        case 128:
            preallocate = 6;
            break;

        /* i386, sparc64 */
        case 256:
            preallocate = 7;
            break;

        /* sparc64 in 32-bit mode */
        default:
            preallocate = 8;
        }
    }

    mask = 0;
    inc = 0x80000000;

    while (preallocate--) {

        key = 0;
        mask >>= 1;
        mask |= 0x80000000;

        do {
            if (ngx_radix32tree_insert(tree, key, mask, NGX_RADIX_NO_VALUE)
                != NGX_OK)
            {
                return NULL;
            }

            key += inc;

        } while (key);

         inc >>= 1;
    }

    return tree;
}


ngx_int_t
ngx_radix32tree_insert(ngx_radix_tree_t *tree, uint32_t key, uint32_t mask,
    uintptr_t value)
{
    uint32_t           bit;
    ngx_radix_node_t  *node, *next;

    bit = 0x80000000;

    node = tree->root;
    next = tree->root;

    while (bit & mask) {
        if (key & bit) {
            next = node->right;

```

```
        } else {
            next = node->left;
        }

        if (next == NULL) {
            break;
        }

        bit >>= 1;
        node = next;
    }

    if (next) {
        if (node->value != NGX_RADIX_NO_VALUE) {
            return NGX_BUSY;
        }

        node->value = value;
        return NGX_OK;
    }

    while (bit & mask) {
        next = ngx_radix_alloc(tree);
        if (next == NULL) {
            return NGX_ERROR;
        }

        next->right = NULL;
        next->left = NULL;
        next->parent = node;
        next->value = NGX_RADIX_NO_VALUE;

        if (key & bit) {
            node->right = next;

        } else {
            node->left = next;
        }

        bit >>= 1;
        node = next;
    }

    node->value = value;

    return NGX_OK;
}


ngx_int_t
ngx_radix32tree_delete(ngx_radix_tree_t *tree, uint32_t key, uint32_t mask)
{
    uint32_t           bit;
    ngx_radix_node_t  *node;

    bit = 0x80000000;
    node = tree->root;

    while (node && (bit & mask)) {
        if (key & bit) {
            node = node->right;

        } else {
            node = node->left;
        }

        bit >>= 1;
    }

    if (node == NULL) {
        return NGX_ERROR;
    }

    if (node->right || node->left) {
        if (node->value != NGX_RADIX_NO_VALUE) {
            node->value = NGX_RADIX_NO_VALUE;
```

```
            return NGX_OK;
        }

        return NGX_ERROR;
    }

    for ( ;; ) {
        if (node->parent->right == node) {
            node->parent->right = NULL;

        } else {
            node->parent->left = NULL;
        }

        node->right = tree->free;
        tree->free = node;

        node = node->parent;

        if (node->right || node->left) {
            break;
        }

        if (node->value != NGX_RADIX_NO_VALUE) {
            break;
        }

        if (node->parent == NULL) {
            break;
        }
    }

    return NGX_OK;
}


uintptr_t
ngx_radix32tree_find(ngx_radix_tree_t *tree, uint32_t key)
{
    uint32_t           bit;
    uintptr_t          value;
    ngx_radix_node_t  *node;

    bit = 0x80000000;
    value = NGX_RADIX_NO_VALUE;
    node = tree->root;

    while (node) {
        if (node->value != NGX_RADIX_NO_VALUE) {
            value = node->value;
        }

        if (key & bit) {
            node = node->right;

        } else {
            node = node->left;
        }

        bit >>= 1;
    }

    return value;
}


#if (NGX_HAVE_INET6)

ngx_int_t
ngx_radix128tree_insert(ngx_radix_tree_t *tree, u_char *key, u_char *mask,
    uintptr_t value)
{
    u_char             bit;
    ngx_uint_t         i;
    ngx_radix_node_t  *node, *next;

```

```
    i = 0;
    bit = 0x80;

    node = tree->root;
    next = tree->root;

    while (bit & mask[i]) {
        if (key[i] & bit) {
            next = node->right;

        } else {
            next = node->left;
        }

        if (next == NULL) {
            break;
        }

        bit >>= 1;
        node = next;

        if (bit == 0) {
            if (++i == 16) {
                break;
            }

            bit = 0x80;
        }
    }

    if (next) {
        if (node->value != NGX_RADIX_NO_VALUE) {
            return NGX_BUSY;
        }

        node->value = value;
        return NGX_OK;
    }

    while (bit & mask[i]) {
        next = ngx_radix_alloc(tree);
        if (next == NULL) {
            return NGX_ERROR;
        }

        next->right = NULL;
        next->left = NULL;
        next->parent = node;
        next->value = NGX_RADIX_NO_VALUE;

        if (key[i] & bit) {
            node->right = next;

        } else {
            node->left = next;
        }

        bit >>= 1;
        node = next;

        if (bit == 0) {
            if (++i == 16) {
                break;
            }

            bit = 0x80;
        }
    }

    node->value = value;

    return NGX_OK;
}


ngx_int_t
```

```
ngx_radix128tree_delete(ngx_radix_tree_t *tree, u_char *key, u_char *mask)
{
    u_char             bit;
    ngx_uint_t         i;
    ngx_radix_node_t  *node;

    i = 0;
    bit = 0x80;
    node = tree->root;

    while (node && (bit & mask[i])) {
        if (key[i] & bit) {
            node = node->right;

        } else {
            node = node->left;
        }

        bit >>= 1;

        if (bit == 0) {
            if (++i == 16) {
                break;
            }

            bit = 0x80;
        }
    }

    if (node == NULL) {
        return NGX_ERROR;
    }

    if (node->right || node->left) {
        if (node->value != NGX_RADIX_NO_VALUE) {
            node->value = NGX_RADIX_NO_VALUE;
            return NGX_OK;
        }

        return NGX_ERROR;
    }

    for ( ;; ) {
        if (node->parent->right == node) {
            node->parent->right = NULL;

        } else {
            node->parent->left = NULL;
        }

        node->right = tree->free;
        tree->free = node;

        node = node->parent;

        if (node->right || node->left) {
            break;
        }

        if (node->value != NGX_RADIX_NO_VALUE) {
            break;
        }

        if (node->parent == NULL) {
            break;
        }
    }

    return NGX_OK;
}


uintptr_t
ngx_radix128tree_find(ngx_radix_tree_t *tree, u_char *key)
{
    u_char             bit;
```

```
    uintptr_t          value;
    ngx_uint_t         i;
    ngx_radix_node_t  *node;

    i = 0;
    bit = 0x80;
    value = NGX_RADIX_NO_VALUE;
    node = tree->root;

    while (node) {
        if (node->value != NGX_RADIX_NO_VALUE) {
            value = node->value;
        }

        if (key[i] & bit) {
            node = node->right;

        } else {
            node = node->left;
        }

        bit >>= 1;

        if (bit == 0) {
            i++;
            bit = 0x80;
        }
    }

    return value;
}

#endif


static ngx_radix_node_t *
ngx_radix_alloc(ngx_radix_tree_t *tree)
{
    ngx_radix_node_t  *p;

    if (tree->free) {
        p = tree->free;
        tree->free = tree->free->right;
        return p;
    }

    if (tree->size < sizeof(ngx_radix_node_t)) {
        tree->start = ngx_pmemalign(tree->pool, ngx_pagesize, ngx_pagesize);
        if (tree->start == NULL) {
            return NULL;
        }

        tree->size = ngx_pagesize;
    }

    p = (ngx_radix_node_t *) tree->start;
    tree->start += sizeof(ngx_radix_node_t);
    tree->size -= sizeof(ngx_radix_node_t);

    return p;
}
```

One Level Up Top Level

src/http/modules/ngx_http_geoip_module.c - nginx-1.7.10

Global variables defined

- ngx_http_geoip_commands
- ngx_http_geoip_country_functions
- ngx_http_geoip_country_v6_functions
- ngx_http_geoip_module
- ngx_http_geoip_module_ctx
- ngx_http_geoip_vars

Data types defined

- ngx_http_geoip_conf_t
- ngx_http_geoip_var_t
- ngx_http_geoip_variable_handler_pt
- ngx_http_geoip_variable_handler_v6_pt

Functions defined

- ngx_http_geoip_add_variables
- ngx_http_geoip_addr
- ngx_http_geoip_addr_v6
- ngx_http_geoip_cidr_value
- ngx_http_geoip_city
- ngx_http_geoip_city_float_variable
- ngx_http_geoip_city_int_variable
- ngx_http_geoip_city_variable
- ngx_http_geoip_cleanup
- ngx_http_geoip_country
- ngx_http_geoip_country_variable
- ngx_http_geoip_create_conf
- ngx_http_geoip_get_city_record
- ngx_http_geoip_init_conf
- ngx_http_geoip_org
- ngx_http_geoip_org_variable
- ngx_http_geoip_proxy

- ngx_http_geoip_region_name_variable

Macros defined

- NGX_GEOIP_COUNTRY_CODE
- NGX_GEOIP_COUNTRY_CODE3
- NGX_GEOIP_COUNTRY_NAME

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12  #include <GeoIP.h>
13  #include <GeoIPCity.h>
14
15
16  #define NGX_GEOIP_COUNTRY_CODE   0
17  #define NGX_GEOIP_COUNTRY_CODE3  1
18  #define NGX_GEOIP_COUNTRY_NAME   2
19
20
21  typedef struct {
22      GeoIP        *country;
23      GeoIP        *org;
24      GeoIP        *city;
25      ngx_array_t  *proxies;    /* array of ngx_cidr_t */
26      ngx_flag_t    proxy_recursive;
27  #if (NGX_HAVE_GEOIP_V6)
28      unsigned      country_v6:1;
29      unsigned      org_v6:1;
30      unsigned      city_v6:1;
31  #endif
32  } ngx_http_geoip_conf_t;
33
34
35  typedef struct {
36      ngx_str_t    *name;
37      uintptr_t     data;
38  } ngx_http_geoip_var_t;
39
40
41  typedef const char *(*ngx_http_geoip_variable_handler_pt)(GeoIP *,
42      u_long addr);
43
44
45  ngx_http_geoip_variable_handler_pt ngx_http_geoip_country_functions[] = {
46      GeoIP_country_code_by_ipnum,
47      GeoIP_country_code3_by_ipnum,
48      GeoIP_country_name_by_ipnum,
49  };
50
51
52  #if (NGX_HAVE_GEOIP_V6)
53
54  typedef const char *(*ngx_http_geoip_variable_handler_v6_pt)(GeoIP *,
55      geoipv6_t addr);
56
57
58  ngx_http_geoip_variable_handler_v6_pt ngx_http_geoip_country_v6_functions[] = {
59      GeoIP_country_code_by_ipnum_v6,
60      GeoIP_country_code3_by_ipnum_v6,
```

```
    GeoIP_country_name_by_ipnum_v6,
};

#endif


static ngx_int_t ngx_http_geoip_country_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_geoip_org_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_geoip_city_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_geoip_region_name_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_geoip_city_float_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static ngx_int_t ngx_http_geoip_city_int_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static GeoIPRecord *ngx_http_geoip_get_city_record(ngx_http_request_t *r);

static ngx_int_t ngx_http_geoip_add_variables(ngx_conf_t *cf);
static void *ngx_http_geoip_create_conf(ngx_conf_t *cf);
static char *ngx_http_geoip_init_conf(ngx_conf_t *cf, void *conf);
static char *ngx_http_geoip_country(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_geoip_org(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_geoip_city(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_geoip_proxy(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static ngx_int_t ngx_http_geoip_cidr_value(ngx_conf_t *cf, ngx_str_t *net,
    ngx_cidr_t *cidr);
static void ngx_http_geoip_cleanup(void *data);


static ngx_command_t  ngx_http_geoip_commands[] = {

    { ngx_string("geoip_country"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE12,
      ngx_http_geoip_country,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("geoip_org"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE12,
      ngx_http_geoip_org,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("geoip_city"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE12,
      ngx_http_geoip_city,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("geoip_proxy"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_http_geoip_proxy,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("geoip_proxy_recursive"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_geoip_conf_t, proxy_recursive),
      NULL },

      ngx_null_command
};

```

```
137
138 static ngx_http_module_t  ngx_http_geoip_module_ctx = {
139     ngx_http_geoip_add_variables,          /* preconfiguration */
140     NULL,                                  /* postconfiguration */
141
142     ngx_http_geoip_create_conf,            /* create main configuration */
143     ngx_http_geoip_init_conf,              /* init main configuration */
144
145     NULL,                                  /* create server configuration */
146     NULL,                                  /* merge server configuration */
147
148     NULL,                                  /* create location configuration */
149     NULL                                   /* merge location configuration */
150 };
151
152
153 ngx_module_t  ngx_http_geoip_module = {
154     NGX_MODULE_V1,
155     &ngx_http_geoip_module_ctx,            /* module context */
156     ngx_http_geoip_commands,               /* module directives */
157     NGX_HTTP_MODULE,                       /* module type */
158     NULL,                                  /* init master */
159     NULL,                                  /* init module */
160     NULL,                                  /* init process */
161     NULL,                                  /* init thread */
162     NULL,                                  /* exit thread */
163     NULL,                                  /* exit process */
164     NULL,                                  /* exit master */
165     NGX_MODULE_V1_PADDING
166 };
167
168
169 static ngx_http_variable_t  ngx_http_geoip_vars[] = {
170
171     { ngx_string("geoip_country_code"), NULL,
172       ngx_http_geoip_country_variable,
173       NGX_GEOIP_COUNTRY_CODE, 0, 0 },
174
175     { ngx_string("geoip_country_code3"), NULL,
176       ngx_http_geoip_country_variable,
177       NGX_GEOIP_COUNTRY_CODE3, 0, 0 },
178
179     { ngx_string("geoip_country_name"), NULL,
180       ngx_http_geoip_country_variable,
181       NGX_GEOIP_COUNTRY_NAME, 0, 0 },
182
183     { ngx_string("geoip_org"), NULL,
184       ngx_http_geoip_org_variable,
185       0, 0, 0 },
186
187     { ngx_string("geoip_city_continent_code"), NULL,
188       ngx_http_geoip_city_variable,
189       offsetof(GeoIPRecord, continent_code), 0, 0 },
190
191     { ngx_string("geoip_city_country_code"), NULL,
192       ngx_http_geoip_city_variable,
193       offsetof(GeoIPRecord, country_code), 0, 0 },
194
195     { ngx_string("geoip_city_country_code3"), NULL,
196       ngx_http_geoip_city_variable,
197       offsetof(GeoIPRecord, country_code3), 0, 0 },
198
199     { ngx_string("geoip_city_country_name"), NULL,
200       ngx_http_geoip_city_variable,
201       offsetof(GeoIPRecord, country_name), 0, 0 },
202
203     { ngx_string("geoip_region"), NULL,
204       ngx_http_geoip_city_variable,
205       offsetof(GeoIPRecord, region), 0, 0 },
206
207     { ngx_string("geoip_region_name"), NULL,
208       ngx_http_geoip_region_name_variable,
209       0, 0, 0 },
210
211     { ngx_string("geoip_city"), NULL,
212       ngx_http_geoip_city_variable,
```

```
      offsetof(GeoIPRecord, city), 0, 0 },

    { ngx_string("geoip_postal_code"), NULL,
      ngx_http_geoip_city_variable,
      offsetof(GeoIPRecord, postal_code), 0, 0 },

    { ngx_string("geoip_latitude"), NULL,
      ngx_http_geoip_city_float_variable,
      offsetof(GeoIPRecord, latitude), 0, 0 },

    { ngx_string("geoip_longitude"), NULL,
      ngx_http_geoip_city_float_variable,
      offsetof(GeoIPRecord, longitude), 0, 0 },

    { ngx_string("geoip_dma_code"), NULL,
      ngx_http_geoip_city_int_variable,
      offsetof(GeoIPRecord, dma_code), 0, 0 },

    { ngx_string("geoip_area_code"), NULL,
      ngx_http_geoip_city_int_variable,
      offsetof(GeoIPRecord, area_code), 0, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static u_long
ngx_http_geoip_addr(ngx_http_request_t *r, ngx_http_geoip_conf_t *gcf)
{
    ngx_addr_t           addr;
    ngx_array_t         *xfwd;
    struct sockaddr_in  *sin;

    addr.sockaddr = r->connection->sockaddr;
    addr.socklen = r->connection->socklen;
    /* addr.name = r->connection->addr_text; */

    xfwd = &r->headers_in.x_forwarded_for;

    if (xfwd->nelts > 0 && gcf->proxies != NULL) {
        (void) ngx_http_get_forwarded_addr(r, &addr, xfwd, NULL,
                                           gcf->proxies, gcf->proxy_recursive);
    }

#if (NGX_HAVE_INET6)

    if (addr.sockaddr->sa_family == AF_INET6) {
        u_char           *p;
        in_addr_t         inaddr;
        struct in6_addr  *inaddr6;

        inaddr6 = &((struct sockaddr_in6 *) addr.sockaddr)->sin6_addr;

        if (IN6_IS_ADDR_V4MAPPED(inaddr6)) {
            p = inaddr6->s6_addr;

            inaddr = p[12] << 24;
            inaddr += p[13] << 16;
            inaddr += p[14] << 8;
            inaddr += p[15];

            return inaddr;
        }
    }

#endif

    if (addr.sockaddr->sa_family != AF_INET) {
        return INADDR_NONE;
    }

    sin = (struct sockaddr_in *) addr.sockaddr;
    return ntohl(sin->sin_addr.s_addr);
}
```

```
#if (NGX_HAVE_GEOIP_V6)

static geoipv6_t
ngx_http_geoip_addr_v6(ngx_http_request_t *r, ngx_http_geoip_conf_t *gcf)
{
    ngx_addr_t            addr;
    ngx_array_t          *xfwd;
    in_addr_t             addr4;
    struct in6_addr       addr6;
    struct sockaddr_in   *sin;
    struct sockaddr_in6  *sin6;

    addr.sockaddr = r->connection->sockaddr;
    addr.socklen = r->connection->socklen;
    /* addr.name = r->connection->addr_text; */

    xfwd = &r->headers_in.x_forwarded_for;

    if (xfwd->nelts > 0 && gcf->proxies != NULL) {
        (void) ngx_http_get_forwarded_addr(r, &addr, xfwd, NULL,
                                           gcf->proxies, gcf->proxy_recursive);
    }

    switch (addr.sockaddr->sa_family) {

    case AF_INET:
        /* Produce IPv4-mapped IPv6 address. */
        sin = (struct sockaddr_in *) addr.sockaddr;
        addr4 = ntohl(sin->sin_addr.s_addr);

        ngx_memzero(&addr6, sizeof(struct in6_addr));
        addr6.s6_addr[10] = 0xff;
        addr6.s6_addr[11] = 0xff;
        addr6.s6_addr[12] = addr4 >> 24;
        addr6.s6_addr[13] = addr4 >> 16;
        addr6.s6_addr[14] = addr4 >> 8;
        addr6.s6_addr[15] = addr4;
        return addr6;

    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) addr.sockaddr;
        return sin6->sin6_addr;

    default:
        return in6addr_any;
    }
}

#endif


static ngx_int_t
ngx_http_geoip_country_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_http_geoip_variable_handler_pt     handler =
        ngx_http_geoip_country_functions[data];
#if (NGX_HAVE_GEOIP_V6)
    ngx_http_geoip_variable_handler_v6_pt  handler_v6 =
        ngx_http_geoip_country_v6_functions[data];
#endif

    const char             *val;
    ngx_http_geoip_conf_t  *gcf;

    gcf = ngx_http_get_module_main_conf(r, ngx_http_geoip_module);

    if (gcf->country == NULL) {
        goto not_found;
    }

#if (NGX_HAVE_GEOIP_V6)
    val = gcf->country_v6
              ? handler_v6(gcf->country, ngx_http_geoip_addr_v6(r, gcf))
              : handler(gcf->country, ngx_http_geoip_addr(r, gcf));
#else
```

```
    val = handler(gcf->country, ngx_http_geoip_addr(r, gcf));
#endif

    if (val == NULL) {
        goto not_found;
    }

    v->len = ngx_strlen(val);
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = (u_char *) val;

    return NGX_OK;

not_found:

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_geoip_org_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    size_t                  len;
    char                   *val;
    ngx_http_geoip_conf_t  *gcf;

    gcf = ngx_http_get_module_main_conf(r, ngx_http_geoip_module);

    if (gcf->org == NULL) {
        goto not_found;
    }

#if (NGX_HAVE_GEOIP_V6)
    val = gcf->org_v6
              ? GeoIP_name_by_ipnum_v6(gcf->org,
                                       ngx_http_geoip_addr_v6(r, gcf))
              : GeoIP_name_by_ipnum(gcf->org,
                                    ngx_http_geoip_addr(r, gcf));
#else
    val = GeoIP_name_by_ipnum(gcf->org, ngx_http_geoip_addr(r, gcf));
#endif

    if (val == NULL) {
        goto not_found;
    }

    len = ngx_strlen(val);
    v->data = ngx_pnalloc(r->pool, len);
    if (v->data == NULL) {
        ngx_free(val);
        return NGX_ERROR;
    }

    ngx_memcpy(v->data, val, len);

    v->len = len;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    ngx_free(val);

    return NGX_OK;

not_found:

    v->not_found = 1;

    return NGX_OK;
}

```

```
441
442 static ngx_int_t
443 ngx_http_geoip_city_variable(ngx_http_request_t *r,
444     ngx_http_variable_value_t *v, uintptr_t data)
445 {
446     char         *val;
447     size_t        len;
448     GeoIPRecord  *gr;
449
450     gr = ngx_http_geoip_get_city_record(r);
451     if (gr == NULL) {
452         goto not_found;
453     }
454
455     val = *(char **) ((char *) gr + data);
456     if (val == NULL) {
457         goto no_value;
458     }
459
460     len = ngx_strlen(val);
461     v->data = ngx_pnalloc(r->pool, len);
462     if (v->data == NULL) {
463         GeoIPRecord_delete(gr);
464         return NGX_ERROR;
465     }
466
467     ngx_memcpy(v->data, val, len);
468
469     v->len = len;
470     v->valid = 1;
471     v->no_cacheable = 0;
472     v->not_found = 0;
473
474     GeoIPRecord_delete(gr);
475
476     return NGX_OK;
477
478 no_value:
479
480     GeoIPRecord_delete(gr);
481
482 not_found:
483
484     v->not_found = 1;
485
486     return NGX_OK;
487 }
488
489
490 static ngx_int_t
491 ngx_http_geoip_region_name_variable(ngx_http_request_t *r,
492     ngx_http_variable_value_t *v, uintptr_t data)
493 {
494     size_t        len;
495     const char   *val;
496     GeoIPRecord  *gr;
497
498     gr = ngx_http_geoip_get_city_record(r);
499     if (gr == NULL) {
500         goto not_found;
501     }
502
503     val = GeoIP_region_name_by_code(gr->country_code, gr->region);
504
505     GeoIPRecord_delete(gr);
506
507     if (val == NULL) {
508         goto not_found;
509     }
510
511     len = ngx_strlen(val);
512     v->data = ngx_pnalloc(r->pool, len);
513     if (v->data == NULL) {
514         return NGX_ERROR;
515     }
516
```

```
    ngx_memcpy(v->data, val, len);

    v->len = len;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    return NGX_OK;

not_found:

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_geoip_city_float_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    float         val;
    GeoIPRecord  *gr;

    gr = ngx_http_geoip_get_city_record(r);
    if (gr == NULL) {
        v->not_found = 1;
        return NGX_OK;
    }

    v->data = ngx_pnalloc(r->pool, NGX_INT64_LEN + 5);
    if (v->data == NULL) {
        GeoIPRecord_delete(gr);
        return NGX_ERROR;
    }

    val = *(float *) ((char *) gr + data);

    v->len = ngx_sprintf(v->data, "%.4f", val) - v->data;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    GeoIPRecord_delete(gr);

    return NGX_OK;
}


static ngx_int_t
ngx_http_geoip_city_int_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    int           val;
    GeoIPRecord  *gr;

    gr = ngx_http_geoip_get_city_record(r);
    if (gr == NULL) {
        v->not_found = 1;
        return NGX_OK;
    }

    v->data = ngx_pnalloc(r->pool, NGX_INT64_LEN);
    if (v->data == NULL) {
        GeoIPRecord_delete(gr);
        return NGX_ERROR;
    }

    val = *(int *) ((char *) gr + data);

    v->len = ngx_sprintf(v->data, "%d", val) - v->data;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    GeoIPRecord_delete(gr);
```

```
593
594     return NGX_OK;
595 }
596
597
598 static GeoIPRecord *
599 ngx_http_geoip_get_city_record(ngx_http_request_t *r)
600 {
601     ngx_http_geoip_conf_t  *gcf;
602
603     gcf = ngx_http_get_module_main_conf(r, ngx_http_geoip_module);
604
605     if (gcf->city) {
606 #if (NGX_HAVE_GEOIP_V6)
607         return gcf->city_v6
608                    ? GeoIP_record_by_ipnum_v6(gcf->city,
609                                               ngx_http_geoip_addr_v6(r, gcf))
610                    : GeoIP_record_by_ipnum(gcf->city,
611                                            ngx_http_geoip_addr(r, gcf));
612 #else
613         return GeoIP_record_by_ipnum(gcf->city, ngx_http_geoip_addr(r, gcf));
614 #endif
615     }
616
617     return NULL;
618 }
619
620
621 static ngx_int_t
622 ngx_http_geoip_add_variables(ngx_conf_t *cf)
623 {
624     ngx_http_variable_t  *var, *v;
625
626     for (v = ngx_http_geoip_vars; v->name.len; v++) {
627         var = ngx_http_add_variable(cf, &v->name, v->flags);
628         if (var == NULL) {
629             return NGX_ERROR;
630         }
631
632         var->get_handler = v->get_handler;
633         var->data = v->data;
634     }
635
636     return NGX_OK;
637 }
638
639
640 static void *
641 ngx_http_geoip_create_conf(ngx_conf_t *cf)
642 {
643     ngx_pool_cleanup_t     *cln;
644     ngx_http_geoip_conf_t  *conf;
645
646     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_geoip_conf_t));
647     if (conf == NULL) {
648         return NULL;
649     }
650
651     conf->proxy_recursive = NGX_CONF_UNSET;
652
653     cln = ngx_pool_cleanup_add(cf->pool, 0);
654     if (cln == NULL) {
655         return NULL;
656     }
657
658     cln->handler = ngx_http_geoip_cleanup;
659     cln->data = conf;
660
661     return conf;
662 }
663
664
665 static char *
666 ngx_http_geoip_init_conf(ngx_conf_t *cf, void *conf)
667 {
668     ngx_http_geoip_conf_t  *gcf = conf;
```

```
669
670     ngx_conf_init_value(gcf->proxy_recursive, 0);
671
672     return NGX_CONF_OK;
673 }
674
675
676 static char *
677 ngx_http_geoip_country(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
678 {
679     ngx_http_geoip_conf_t  *gcf = conf;
680
681     ngx_str_t  *value;
682
683     if (gcf->country) {
684         return "is duplicate";
685     }
686
687     value = cf->args->elts;
688
689     gcf->country = GeoIP_open((char *) value[1].data, GEOIP_MEMORY_CACHE);
690
691     if (gcf->country == NULL) {
692         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
693                            "GeoIP_open(\"%V\") failed", &value[1]);
694
695         return NGX_CONF_ERROR;
696     }
697
698     if (cf->args->nelts == 3) {
699         if (ngx_strcmp(value[2].data, "utf8") == 0) {
700             GeoIP_set_charset(gcf->country, GEOIP_CHARSET_UTF8);
701
702         } else {
703             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
704                                "invalid parameter \"%V\"", &value[2]);
705             return NGX_CONF_ERROR;
706         }
707     }
708
709     switch (gcf->country->databaseType) {
710
711     case GEOIP_COUNTRY_EDITION:
712
713         return NGX_CONF_OK;
714
715 #if (NGX_HAVE_GEOIP_V6)
716     case GEOIP_COUNTRY_EDITION_V6:
717
718         gcf->country_v6 = 1;
719         return NGX_CONF_OK;
720 #endif
721
722     default:
723         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
724                            "invalid GeoIP database \"%V\" type:%d",
725                            &value[1], gcf->country->databaseType);
726         return NGX_CONF_ERROR;
727     }
728 }
729
730
731 static char *
732 ngx_http_geoip_org(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
733 {
734     ngx_http_geoip_conf_t  *gcf = conf;
735
736     ngx_str_t  *value;
737
738     if (gcf->org) {
739         return "is duplicate";
740     }
741
742     value = cf->args->elts;
743
744     gcf->org = GeoIP_open((char *) value[1].data, GEOIP_MEMORY_CACHE);
```

```
745
746     if (gcf->org == NULL) {
747         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
748                            "GeoIP_open(\"%V\") failed", &value[1]);
749
750         return NGX_CONF_ERROR;
751     }
752
753     if (cf->args->nelts == 3) {
754         if (ngx_strcmp(value[2].data, "utf8") == 0) {
755             GeoIP_set_charset(gcf->org, GEOIP_CHARSET_UTF8);
756
757         } else {
758             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
759                                "invalid parameter \"%V\"", &value[2]);
760             return NGX_CONF_ERROR;
761         }
762     }
763
764     switch (gcf->org->databaseType) {
765
766     case GEOIP_ISP_EDITION:
767     case GEOIP_ORG_EDITION:
768     case GEOIP_DOMAIN_EDITION:
769     case GEOIP_ASNUM_EDITION:
770
771         return NGX_CONF_OK;
772
773 #if (NGX_HAVE_GEOIP_V6)
774     case GEOIP_ISP_EDITION_V6:
775     case GEOIP_ORG_EDITION_V6:
776     case GEOIP_DOMAIN_EDITION_V6:
777     case GEOIP_ASNUM_EDITION_V6:
778
779         gcf->org_v6 = 1;
780         return NGX_CONF_OK;
781 #endif
782
783     default:
784         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
785                            "invalid GeoIP database \"%V\" type:%d",
786                            &value[1], gcf->org->databaseType);
787         return NGX_CONF_ERROR;
788     }
789 }
790
791
792 static char *
793 ngx_http_geoip_city(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
794 {
795     ngx_http_geoip_conf_t  *gcf = conf;
796
797     ngx_str_t  *value;
798
799     if (gcf->city) {
800         return "is duplicate";
801     }
802
803     value = cf->args->elts;
804
805     gcf->city = GeoIP_open((char *) value[1].data, GEOIP_MEMORY_CACHE);
806
807     if (gcf->city == NULL) {
808         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
809                            "GeoIP_open(\"%V\") failed", &value[1]);
810
811         return NGX_CONF_ERROR;
812     }
813
814     if (cf->args->nelts == 3) {
815         if (ngx_strcmp(value[2].data, "utf8") == 0) {
816             GeoIP_set_charset(gcf->city, GEOIP_CHARSET_UTF8);
817
818         } else {
819             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
820                                "invalid parameter \"%V\"", &value[2]);
```

```
            return NGX_CONF_ERROR;
        }
    }

    switch (gcf->city->databaseType) {

    case GEOIP_CITY_EDITION_REV0:
    case GEOIP_CITY_EDITION_REV1:

        return NGX_CONF_OK;

#if (NGX_HAVE_GEOIP_V6)
    case GEOIP_CITY_EDITION_REV0_V6:
    case GEOIP_CITY_EDITION_REV1_V6:

        gcf->city_v6 = 1;
        return NGX_CONF_OK;
#endif

    default:
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid GeoIP City database \"%V\" type:%d",
                           &value[1], gcf->city->databaseType);
        return NGX_CONF_ERROR;
    }
}


static char *
ngx_http_geoip_proxy(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_geoip_conf_t  *gcf = conf;

    ngx_str_t   *value;
    ngx_cidr_t  cidr, *c;

    value = cf->args->elts;

    if (ngx_http_geoip_cidr_value(cf, &value[1], &cidr) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (gcf->proxies == NULL) {
        gcf->proxies = ngx_array_create(cf->pool, 4, sizeof(ngx_cidr_t));
        if (gcf->proxies == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    c = ngx_array_push(gcf->proxies);
    if (c == NULL) {
        return NGX_CONF_ERROR;
    }

    *c = cidr;

    return NGX_CONF_OK;
}

static ngx_int_t
ngx_http_geoip_cidr_value(ngx_conf_t *cf, ngx_str_t *net, ngx_cidr_t *cidr)
{
    ngx_int_t  rc;

    if (ngx_strcmp(net->data, "255.255.255.255") == 0) {
        cidr->family = AF_INET;
        cidr->u.in.addr = 0xffffffff;
        cidr->u.in.mask = 0xffffffff;

        return NGX_OK;
    }

    rc = ngx_ptocidr(net, cidr);

    if (rc == NGX_ERROR) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid network \"%V\"", net);
```

```
897         return NGX_ERROR;
898     }
899
900     if (rc == NGX_DONE) {
901         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
902                            "low address bits of %V are meaningless", net);
903     }
904
905     return NGX_OK;
906 }
907
908
909 static void
910 ngx_http_geoip_cleanup(void *data)
911 {
912     ngx_http_geoip_conf_t  *gcf = data;
913
914     if (gcf->country) {
915         GeoIP_delete(gcf->country);
916     }
917
918     if (gcf->org) {
919         GeoIP_delete(gcf->org);
920     }
921
922     if (gcf->city) {
923         GeoIP_delete(gcf->city);
924     }
925 }
```

One Level Up Top Level

src/http/modules/ngx_http_gunzip_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_gunzip_filter_commands
- ngx_http_gunzip_filter_module
- ngx_http_gunzip_filter_module_ctx
- ngx_http_next_body_filter
- ngx_http_next_header_filter

Data types defined

- ngx_http_gunzip_conf_t
- ngx_http_gunzip_ctx_t

Functions defined

- ngx_http_gunzip_body_filter
- ngx_http_gunzip_create_conf
- ngx_http_gunzip_filter_add_data
- ngx_http_gunzip_filter_alloc
- ngx_http_gunzip_filter_free
- ngx_http_gunzip_filter_get_buf
- ngx_http_gunzip_filter_inflate
- ngx_http_gunzip_filter_inflate_end
- ngx_http_gunzip_filter_inflate_start
- ngx_http_gunzip_filter_init
- ngx_http_gunzip_header_filter
- ngx_http_gunzip_merge_conf

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Maxim Dounin
5   * Copyright (C) Nginx, Inc.
6   */
7
8
9  #include <ngx_config.h>
10  #include <ngx_core.h>
11  #include <ngx_http.h>
12
13  #include <zlib.h>
14
```

```
15
16  typedef struct {
17      ngx_flag_t           enable;
18      ngx_bufs_t           bufs;
19  } ngx_http_gunzip_conf_t;
20
21
22  typedef struct {
23      ngx_chain_t         *in;
24      ngx_chain_t         *free;
25      ngx_chain_t         *busy;
26      ngx_chain_t         *out;
27      ngx_chain_t        **last_out;
28
29      ngx_buf_t           *in_buf;
30      ngx_buf_t           *out_buf;
31      ngx_int_t            bufs;
32
33      unsigned             started:1;
34      unsigned             flush:4;
35      unsigned             redo:1;
36      unsigned             done:1;
37      unsigned             nomem:1;
38
39      z_stream             zstream;
40      ngx_http_request_t  *request;
41  } ngx_http_gunzip_ctx_t;
42
43
44  static ngx_int_t ngx_http_gunzip_filter_inflate_start(ngx_http_request_t *r,
45      ngx_http_gunzip_ctx_t *ctx);
46  static ngx_int_t ngx_http_gunzip_filter_add_data(ngx_http_request_t *r,
47      ngx_http_gunzip_ctx_t *ctx);
48  static ngx_int_t ngx_http_gunzip_filter_get_buf(ngx_http_request_t *r,
49      ngx_http_gunzip_ctx_t *ctx);
50  static ngx_int_t ngx_http_gunzip_filter_inflate(ngx_http_request_t *r,
51      ngx_http_gunzip_ctx_t *ctx);
52  static ngx_int_t ngx_http_gunzip_filter_inflate_end(ngx_http_request_t *r,
53      ngx_http_gunzip_ctx_t *ctx);
54
55  static void *ngx_http_gunzip_filter_alloc(void *opaque, u_int items,
56      u_int size);
57  static void ngx_http_gunzip_filter_free(void *opaque, void *address);
58
59  static ngx_int_t ngx_http_gunzip_filter_init(ngx_conf_t *cf);
60  static void *ngx_http_gunzip_create_conf(ngx_conf_t *cf);
61  static char *ngx_http_gunzip_merge_conf(ngx_conf_t *cf,
62      void *parent, void *child);
63
64
65  static ngx_command_t  ngx_http_gunzip_filter_commands[] = {
66
67      { ngx_string("gunzip"),
68        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
69        ngx_conf_set_flag_slot,
70        NGX_HTTP_LOC_CONF_OFFSET,
71        offsetof(ngx_http_gunzip_conf_t, enable),
72        NULL },
73
74      { ngx_string("gunzip_buffers"),
75        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
76        ngx_conf_set_bufs_slot,
77        NGX_HTTP_LOC_CONF_OFFSET,
78        offsetof(ngx_http_gunzip_conf_t, bufs),
79        NULL },
80
81        ngx_null_command
82  };
83
84
85  static ngx_http_module_t  ngx_http_gunzip_filter_module_ctx = {
86      NULL,                                  /* preconfiguration */
87      ngx_http_gunzip_filter_init,           /* postconfiguration */
88
89      NULL,                                  /* create main configuration */
90      NULL,                                  /* init main configuration */
```

```
91
92      NULL,                                  /* create server configuration */
93      NULL,                                  /* merge server configuration */
94
95      ngx_http_gunzip_create_conf,           /* create location configuration */
96      ngx_http_gunzip_merge_conf             /* merge location configuration */
97  };
98
99
100 ngx_module_t  ngx_http_gunzip_filter_module = {
101     NGX_MODULE_V1,
102     &ngx_http_gunzip_filter_module_ctx,    /* module context */
103     ngx_http_gunzip_filter_commands,       /* module directives */
104     NGX_HTTP_MODULE,                       /* module type */
105     NULL,                                  /* init master */
106     NULL,                                  /* init module */
107     NULL,                                  /* init process */
108     NULL,                                  /* init thread */
109     NULL,                                  /* exit thread */
110     NULL,                                  /* exit process */
111     NULL,                                  /* exit master */
112     NGX_MODULE_V1_PADDING
113 };
114
115
116 static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
117 static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;
118
119
120 static ngx_int_t
121 ngx_http_gunzip_header_filter(ngx_http_request_t *r)
122 {
123     ngx_http_gunzip_ctx_t   *ctx;
124     ngx_http_gunzip_conf_t  *conf;
125
126     conf = ngx_http_get_module_loc_conf(r, ngx_http_gunzip_filter_module);
127
128     /* TODO support multiple content-codings */
129     /* TODO always gunzip - due to configuration or module request */
130     /* TODO ignore content encoding? */
131
132     if (!conf->enable
133         || r->headers_out.content_encoding == NULL
134         || r->headers_out.content_encoding->value.len != 4
135         || ngx_strncasecmp(r->headers_out.content_encoding->value.data,
136                            (u_char *) "gzip", 4) != 0)
137     {
138         return ngx_http_next_header_filter(r);
139     }
140
141     r->gzip_vary = 1;
142
143     if (!r->gzip_tested) {
144         if (ngx_http_gzip_ok(r) == NGX_OK) {
145             return ngx_http_next_header_filter(r);
146         }
147
148     } else if (r->gzip_ok) {
149         return ngx_http_next_header_filter(r);
150     }
151
152     ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_gunzip_ctx_t));
153     if (ctx == NULL) {
154         return NGX_ERROR;
155     }
156
157     ngx_http_set_ctx(r, ctx, ngx_http_gunzip_filter_module);
158
159     ctx->request = r;
160
161     r->filter_need_in_memory = 1;
162
163     r->headers_out.content_encoding->hash = 0;
164     r->headers_out.content_encoding = NULL;
165
166     ngx_http_clear_content_length(r);
```

```
    ngx_http_clear_accept_ranges(r);
    ngx_http_weak_etag(r);

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_gunzip_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    int                     rc;
    ngx_uint_t              flush;
    ngx_chain_t            *cl;
    ngx_http_gunzip_ctx_t  *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_gunzip_filter_module);

    if (ctx == NULL || ctx->done) {
        return ngx_http_next_body_filter(r, in);
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http gunzip filter");

    if (!ctx->started) {
        if (ngx_http_gunzip_filter_inflate_start(r, ctx) != NGX_OK) {
            goto failed;
        }
    }

    if (in) {
        if (ngx_chain_add_copy(r->pool, &ctx->in, in) != NGX_OK) {
            goto failed;
        }
    }

    if (ctx->nomem) {

        /* flush busy buffers */

        if (ngx_http_next_body_filter(r, NULL) == NGX_ERROR) {
            goto failed;
        }

        cl = NULL;

        ngx_chain_update_chains(r->pool, &ctx->free, &ctx->busy, &cl,
                                (ngx_buf_tag_t) &ngx_http_gunzip_filter_module);
        ctx->nomem = 0;
        flush = 0;

    } else {
        flush = ctx->busy ? 1 : 0;
    }

    for ( ;; ) {

        /* cycle while we can write to a client */

        for ( ;; ) {

            /* cycle while there is data to feed zlib and ... */

            rc = ngx_http_gunzip_filter_add_data(r, ctx);

            if (rc == NGX_DECLINED) {
                break;
            }

            if (rc == NGX_AGAIN) {
                continue;
            }


            /* ... there are buffers to write zlib output */

```

```
            rc = ngx_http_gunzip_filter_get_buf(r, ctx);

            if (rc == NGX_DECLINED) {
                break;
            }

            if (rc == NGX_ERROR) {
                goto failed;
            }

            rc = ngx_http_gunzip_filter_inflate(r, ctx);

            if (rc == NGX_OK) {
                break;
            }

            if (rc == NGX_ERROR) {
                goto failed;
            }

            /* rc == NGX_AGAIN */
        }

        if (ctx->out == NULL && !flush) {
            return ctx->busy ? NGX_AGAIN : NGX_OK;
        }

        rc = ngx_http_next_body_filter(r, ctx->out);

        if (rc == NGX_ERROR) {
            goto failed;
        }

        ngx_chain_update_chains(r->pool, &ctx->free, &ctx->busy, &ctx->out,
                                (ngx_buf_tag_t) &ngx_http_gunzip_filter_module);
        ctx->last_out = &ctx->out;

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "gunzip out: %p", ctx->out);

        ctx->nomem = 0;
        flush = 0;

        if (ctx->done) {
            return rc;
        }
    }

    /* unreachable */

failed:

    ctx->done = 1;

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_gunzip_filter_inflate_start(ngx_http_request_t *r,
    ngx_http_gunzip_ctx_t *ctx)
{
    int  rc;

    ctx->zstream.next_in = Z_NULL;
    ctx->zstream.avail_in = 0;

    ctx->zstream.zalloc = ngx_http_gunzip_filter_alloc;
    ctx->zstream.zfree = ngx_http_gunzip_filter_free;
    ctx->zstream.opaque = ctx;

    /* windowBits +16 to decode gzip, zlib 1.2.0.4+ */
    rc = inflateInit2(&ctx->zstream, MAX_WBITS + 16);

    if (rc != Z_OK) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
```

```
                      "inflateInit2() failed: %d", rc);
        return NGX_ERROR;
    }

    ctx->started = 1;

    ctx->last_out = &ctx->out;
    ctx->flush = Z_NO_FLUSH;

    return NGX_OK;
}


static ngx_int_t
ngx_http_gunzip_filter_add_data(ngx_http_request_t *r,
    ngx_http_gunzip_ctx_t *ctx)
{
    if (ctx->zstream.avail_in || ctx->flush != Z_NO_FLUSH || ctx->redo) {
        return NGX_OK;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gunzip in: %p", ctx->in);

    if (ctx->in == NULL) {
        return NGX_DECLINED;
    }

    ctx->in_buf = ctx->in->buf;
    ctx->in = ctx->in->next;

    ctx->zstream.next_in = ctx->in_buf->pos;
    ctx->zstream.avail_in = ctx->in_buf->last - ctx->in_buf->pos;

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gunzip in_buf:%p ni:%p ai:%ud",
                   ctx->in_buf,
                   ctx->zstream.next_in, ctx->zstream.avail_in);

    if (ctx->in_buf->last_buf || ctx->in_buf->last_in_chain) {
        ctx->flush = Z_FINISH;

    } else if (ctx->in_buf->flush) {
        ctx->flush = Z_SYNC_FLUSH;

    } else if (ctx->zstream.avail_in == 0) {
        /* ctx->flush == Z_NO_FLUSH */
        return NGX_AGAIN;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_gunzip_filter_get_buf(ngx_http_request_t *r,
    ngx_http_gunzip_ctx_t *ctx)
{
    ngx_http_gunzip_conf_t  *conf;

    if (ctx->zstream.avail_out) {
        return NGX_OK;
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gunzip_filter_module);

    if (ctx->free) {
        ctx->out_buf = ctx->free->buf;
        ctx->free = ctx->free->next;

        ctx->out_buf->flush = 0;

    } else if (ctx->bufs < conf->bufs.num) {

        ctx->out_buf = ngx_create_temp_buf(r->pool, conf->bufs.size);
        if (ctx->out_buf == NULL) {
```

```
            return NGX_ERROR;
        }

        ctx->out_buf->tag = (ngx_buf_tag_t) &ngx_http_gunzip_filter_module;
        ctx->out_buf->recycled = 1;
        ctx->bufs++;

    } else {
        ctx->nomem = 1;
        return NGX_DECLINED;
    }

    ctx->zstream.next_out = ctx->out_buf->pos;
    ctx->zstream.avail_out = conf->bufs.size;

    return NGX_OK;
}


static ngx_int_t
ngx_http_gunzip_filter_inflate(ngx_http_request_t *r,
    ngx_http_gunzip_ctx_t *ctx)
{
    int           rc;
    ngx_buf_t    *b;
    ngx_chain_t  *cl;

    ngx_log_debug6(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "inflate in: ni:%p no:%p ai:%ud ao:%ud fl:%d redo:%d",
                   ctx->zstream.next_in, ctx->zstream.next_out,
                   ctx->zstream.avail_in, ctx->zstream.avail_out,
                   ctx->flush, ctx->redo);

    rc = inflate(&ctx->zstream, ctx->flush);

    if (rc != Z_OK && rc != Z_STREAM_END && rc != Z_BUF_ERROR) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "inflate() failed: %d, %d", ctx->flush, rc);
        return NGX_ERROR;
    }

    ngx_log_debug5(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "inflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
                   ctx->zstream.next_in, ctx->zstream.next_out,
                   ctx->zstream.avail_in, ctx->zstream.avail_out,
                   rc);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gunzip in_buf:%p pos:%p",
                   ctx->in_buf, ctx->in_buf->pos);

    if (ctx->zstream.next_in) {
        ctx->in_buf->pos = ctx->zstream.next_in;

        if (ctx->zstream.avail_in == 0) {
            ctx->zstream.next_in = NULL;
        }
    }

    ctx->out_buf->last = ctx->zstream.next_out;

    if (ctx->zstream.avail_out == 0) {

        /* zlib wants to output some more data */

        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        cl->buf = ctx->out_buf;
        cl->next = NULL;
        *ctx->last_out = cl;
        ctx->last_out = &cl->next;

        ctx->redo = 1;
```

```
471
472         return NGX_AGAIN;
473     }
474
475     ctx->redo = 0;
476
477     if (ctx->flush == Z_SYNC_FLUSH) {
478
479         ctx->flush = Z_NO_FLUSH;
480
481         cl = ngx_alloc_chain_link(r->pool);
482         if (cl == NULL) {
483             return NGX_ERROR;
484         }
485
486         b = ctx->out_buf;
487
488         if (ngx_buf_size(b) == 0) {
489
490             b = ngx_calloc_buf(ctx->request->pool);
491             if (b == NULL) {
492                 return NGX_ERROR;
493             }
494
495         } else {
496             ctx->zstream.avail_out = 0;
497         }
498
499         b->flush = 1;
500
501         cl->buf = b;
502         cl->next = NULL;
503         *ctx->last_out = cl;
504         ctx->last_out = &cl->next;
505
506         return NGX_OK;
507     }
508
509     if (ctx->flush == Z_FINISH && ctx->zstream.avail_in == 0) {
510
511         if (rc != Z_STREAM_END) {
512             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
513                           "inflate() returned %d on response end", rc);
514             return NGX_ERROR;
515         }
516
517         if (ngx_http_gunzip_filter_inflate_end(r, ctx) != NGX_OK) {
518             return NGX_ERROR;
519         }
520
521         return NGX_OK;
522     }
523
524     if (rc == Z_STREAM_END && ctx->zstream.avail_in > 0) {
525
526         rc = inflateReset(&ctx->zstream);
527
528         if (rc != Z_OK) {
529             ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
530                           "inflateReset() failed: %d", rc);
531             return NGX_ERROR;
532         }
533
534         ctx->redo = 1;
535
536         return NGX_AGAIN;
537     }
538
539     if (ctx->in == NULL) {
540
541         b = ctx->out_buf;
542
543         if (ngx_buf_size(b) == 0) {
544             return NGX_OK;
545         }
546
```

```
        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        ctx->zstream.avail_out = 0;

        cl->buf = b;
        cl->next = NULL;
        *ctx->last_out = cl;
        ctx->last_out = &cl->next;

        return NGX_OK;
    }

    return NGX_AGAIN;
}


static ngx_int_t
ngx_http_gunzip_filter_inflate_end(ngx_http_request_t *r,
    ngx_http_gunzip_ctx_t *ctx)
{
    int           rc;
    ngx_buf_t    *b;
    ngx_chain_t  *cl;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gunzip inflate end");

    rc = inflateEnd(&ctx->zstream);

    if (rc != Z_OK) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      "inflateEnd() failed: %d", rc);
        return NGX_ERROR;
    }

    b = ctx->out_buf;

    if (ngx_buf_size(b) == 0) {

        b = ngx_calloc_buf(ctx->request->pool);
        if (b == NULL) {
            return NGX_ERROR;
        }
    }

    cl = ngx_alloc_chain_link(r->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    cl->buf = b;
    cl->next = NULL;
    *ctx->last_out = cl;
    ctx->last_out = &cl->next;

    b->last_buf = (r == r->main) ? 1 : 0;
    b->last_in_chain = 1;
    b->sync = 1;

    ctx->done = 1;

    return NGX_OK;
}


static void *
ngx_http_gunzip_filter_alloc(void *opaque, u_int items, u_int size)
{
    ngx_http_gunzip_ctx_t *ctx = opaque;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, ctx->request->connection->log, 0,
                   "gunzip alloc: n:%ud s:%ud",
                   items, size);
```

```
623
624     return ngx_palloc(ctx->request->pool, items * size);
625 }
626
627
628 static void
629 ngx_http_gunzip_filter_free(void *opaque, void *address)
630 {
631 #if 0
632     ngx_http_gunzip_ctx_t *ctx = opaque;
633
634     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->request->connection->log, 0,
635                    "gunzip free: %p", address);
636 #endif
637 }
638
639
640 static void *
641 ngx_http_gunzip_create_conf(ngx_conf_t *cf)
642 {
643     ngx_http_gunzip_conf_t  *conf;
644
645     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_gunzip_conf_t));
646     if (conf == NULL) {
647         return NULL;
648     }
649
650     /*
651      * set by ngx_pcalloc():
652      *
653      *     conf->bufs.num = 0;
654      */
655
656     conf->enable = NGX_CONF_UNSET;
657
658     return conf;
659 }
660
661
662 static char *
663 ngx_http_gunzip_merge_conf(ngx_conf_t *cf, void *parent, void *child)
664 {
665     ngx_http_gunzip_conf_t *prev = parent;
666     ngx_http_gunzip_conf_t *conf = child;
667
668     ngx_conf_merge_value(conf->enable, prev->enable, 0);
669
670     ngx_conf_merge_bufs_value(conf->bufs, prev->bufs,
671                               (128 * 1024) / ngx_pagesize, ngx_pagesize);
672
673     return NGX_CONF_OK;
674 }
675
676
677 static ngx_int_t
678 ngx_http_gunzip_filter_init(ngx_conf_t *cf)
679 {
680     ngx_http_next_header_filter = ngx_http_top_header_filter;
681     ngx_http_top_header_filter = ngx_http_gunzip_header_filter;
682
683     ngx_http_next_body_filter = ngx_http_top_body_filter;
684     ngx_http_top_body_filter = ngx_http_gunzip_body_filter;
685
686     return NGX_OK;
687 }
```

One Level Up Top Level

src/http/modules/ngx_http_gzip_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_gzip_comp_level_bounds
- ngx_http_gzip_filter_commands
- ngx_http_gzip_filter_module
- ngx_http_gzip_filter_module_ctx
- ngx_http_gzip_hash_p
- ngx_http_gzip_ratio
- ngx_http_gzip_window_p
- ngx_http_next_body_filter
- ngx_http_next_header_filter

Data types defined

- gztrailer
- gztrailer
- ngx_http_gzip_conf_t
- ngx_http_gzip_ctx_t

Functions defined

- ngx_http_gzip_add_variables
- ngx_http_gzip_body_filter
- ngx_http_gzip_create_conf
- ngx_http_gzip_filter_add_data
- ngx_http_gzip_filter_alloc
- ngx_http_gzip_filter_buffer
- ngx_http_gzip_filter_deflate
- ngx_http_gzip_filter_deflate_end
- ngx_http_gzip_filter_deflate_start
- ngx_http_gzip_filter_free
- ngx_http_gzip_filter_free_copy_buf
- ngx_http_gzip_filter_get_buf
- ngx_http_gzip_filter_gzheader
- ngx_http_gzip_filter_init

- ngx_http_gzip_filter_memory
- ngx_http_gzip_hash
- ngx_http_gzip_header_filter
- ngx_http_gzip_merge_conf
- ngx_http_gzip_ratio_variable
- ngx_http_gzip_window

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12  #include <zlib.h>
13
14
15  typedef struct {
16      ngx_flag_t           enable;
17      ngx_flag_t           no_buffer;
18
19      ngx_hash_t           types;
20
21      ngx_bufs_t           bufs;
22
23      size_t               postpone_gzipping;
24      ngx_int_t            level;
25      size_t               wbits;
26      size_t               memlevel;
27      ssize_t              min_length;
28
29      ngx_array_t         *types_keys;
30  } ngx_http_gzip_conf_t;
31
32
33  typedef struct {
34      ngx_chain_t         *in;
35      ngx_chain_t         *free;
36      ngx_chain_t         *busy;
37      ngx_chain_t         *out;
38      ngx_chain_t        **last_out;
39
40      ngx_chain_t         *copied;
41      ngx_chain_t         *copy_buf;
42
43      ngx_buf_t           *in_buf;
44      ngx_buf_t           *out_buf;
45      ngx_int_t            bufs;
46
47      void                *preallocated;
48      char                *free_mem;
49      ngx_uint_t           allocated;
50
51      int                  wbits;
52      int                  memlevel;
53
54      unsigned             flush:4;
55      unsigned             redo:1;
56      unsigned             done:1;
57      unsigned             nomem:1;
58      unsigned             gzheader:1;
59      unsigned             buffering:1;
```

```
60
61      size_t              zin;
62      size_t              zout;
63
64      uint32_t            crc32;
65      z_stream            zstream;
66      ngx_http_request_t  *request;
67  } ngx_http_gzip_ctx_t;
68
69
70  #if (NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED)
71
72  struct gztrailer {
73      uint32_t  crc32;
74      uint32_t  zlen;
75  };
76
77  #else /* NGX_HAVE_BIG_ENDIAN || !NGX_HAVE_NONALIGNED */
78
79  struct gztrailer {
80      u_char  crc32[4];
81      u_char  zlen[4];
82  };
83
84  #endif
85
86
87  static void ngx_http_gzip_filter_memory(ngx_http_request_t *r,
88      ngx_http_gzip_ctx_t *ctx);
89  static ngx_int_t ngx_http_gzip_filter_buffer(ngx_http_gzip_ctx_t *ctx,
90      ngx_chain_t *in);
91  static ngx_int_t ngx_http_gzip_filter_deflate_start(ngx_http_request_t *r,
92      ngx_http_gzip_ctx_t *ctx);
93  static ngx_int_t ngx_http_gzip_filter_gzheader(ngx_http_request_t *r,
94      ngx_http_gzip_ctx_t *ctx);
95  static ngx_int_t ngx_http_gzip_filter_add_data(ngx_http_request_t *r,
96      ngx_http_gzip_ctx_t *ctx);
97  static ngx_int_t ngx_http_gzip_filter_get_buf(ngx_http_request_t *r,
98      ngx_http_gzip_ctx_t *ctx);
99  static ngx_int_t ngx_http_gzip_filter_deflate(ngx_http_request_t *r,
100      ngx_http_gzip_ctx_t *ctx);
101  static ngx_int_t ngx_http_gzip_filter_deflate_end(ngx_http_request_t *r,
102      ngx_http_gzip_ctx_t *ctx);
103
104  static void *ngx_http_gzip_filter_alloc(void *opaque, u_int items,
105      u_int size);
106  static void ngx_http_gzip_filter_free(void *opaque, void *address);
107  static void ngx_http_gzip_filter_free_copy_buf(ngx_http_request_t *r,
108      ngx_http_gzip_ctx_t *ctx);
109
110  static ngx_int_t ngx_http_gzip_add_variables(ngx_conf_t *cf);
111  static ngx_int_t ngx_http_gzip_ratio_variable(ngx_http_request_t *r,
112      ngx_http_variable_value_t *v, uintptr_t data);
113
114  static ngx_int_t ngx_http_gzip_filter_init(ngx_conf_t *cf);
115  static void *ngx_http_gzip_create_conf(ngx_conf_t *cf);
116  static char *ngx_http_gzip_merge_conf(ngx_conf_t *cf,
117      void *parent, void *child);
118  static char *ngx_http_gzip_window(ngx_conf_t *cf, void *post, void *data);
119  static char *ngx_http_gzip_hash(ngx_conf_t *cf, void *post, void *data);
120
121
122  static ngx_conf_num_bounds_t  ngx_http_gzip_comp_level_bounds = {
123      ngx_conf_check_num_bounds, 1, 9
124  };
125
126  static ngx_conf_post_handler_pt  ngx_http_gzip_window_p = ngx_http_gzip_window;
127  static ngx_conf_post_handler_pt  ngx_http_gzip_hash_p = ngx_http_gzip_hash;
128
129
130  static ngx_command_t  ngx_http_gzip_filter_commands[] = {
131
132      { ngx_string("gzip"),
133        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
134                          |NGX_CONF_FLAG,
135        ngx_conf_set_flag_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, enable),
      NULL },

    { ngx_string("gzip_buffers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_conf_set_bufs_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, bufs),
      NULL },

    { ngx_string("gzip_types"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_types_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, types_keys),
      &ngx_http_html_default_types[0] },

    { ngx_string("gzip_comp_level"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, level),
      &ngx_http_gzip_comp_level_bounds },

    { ngx_string("gzip_window"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, wbits),
      &ngx_http_gzip_window_p },

    { ngx_string("gzip_hash"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, memlevel),
      &ngx_http_gzip_hash_p },

    { ngx_string("postpone_gzipping"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, postpone_gzipping),
      NULL },

    { ngx_string("gzip_no_buffer"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, no_buffer),
      NULL },

    { ngx_string("gzip_min_length"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_gzip_conf_t, min_length),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_gzip_filter_module_ctx = {
    ngx_http_gzip_add_variables,           /* preconfiguration */
    ngx_http_gzip_filter_init,             /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_gzip_create_conf,             /* create location configuration */
    ngx_http_gzip_merge_conf               /* merge location configuration */
```

```
};


ngx_module_t  ngx_http_gzip_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_gzip_filter_module_ctx,      /* module context */
    ngx_http_gzip_filter_commands,         /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t  ngx_http_gzip_ratio = ngx_string("gzip_ratio");

static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static ngx_int_t
ngx_http_gzip_header_filter(ngx_http_request_t *r)
{
    ngx_table_elt_t       *h;
    ngx_http_gzip_ctx_t   *ctx;
    ngx_http_gzip_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_filter_module);

    if (!conf->enable
        || (r->headers_out.status != NGX_HTTP_OK
            && r->headers_out.status != NGX_HTTP_FORBIDDEN
            && r->headers_out.status != NGX_HTTP_NOT_FOUND)
        || (r->headers_out.content_encoding
            && r->headers_out.content_encoding->value.len)
        || (r->headers_out.content_length_n != -1
            && r->headers_out.content_length_n < conf->min_length)
        || ngx_http_test_content_type(r, &conf->types) == NULL
        || r->header_only)
    {
        return ngx_http_next_header_filter(r);
    }

    r->gzip_vary = 1;

#if (NGX_HTTP_DEGRADATION)
    {
    ngx_http_core_loc_conf_t  *clcf;

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (clcf->gzip_disable_degradation && ngx_http_degraded(r)) {
        return ngx_http_next_header_filter(r);
    }
    }
#endif

    if (!r->gzip_tested) {
        if (ngx_http_gzip_ok(r) != NGX_OK) {
            return ngx_http_next_header_filter(r);
        }

    } else if (!r->gzip_ok) {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_gzip_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

```

```
    ngx_http_set_ctx(r, ctx, ngx_http_gzip_filter_module);

    ctx->request = r;
    ctx->buffering = (conf->postpone_gzipping != 0);

    ngx_http_gzip_filter_memory(r, ctx);

    h = ngx_list_push(&r->headers_out.headers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    h->hash = 1;
    ngx_str_set(&h->key, "Content-Encoding");
    ngx_str_set(&h->value, "gzip");
    r->headers_out.content_encoding = h;

    r->main_filter_need_in_memory = 1;

    ngx_http_clear_content_length(r);
    ngx_http_clear_accept_ranges(r);
    ngx_http_weak_etag(r);

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_gzip_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    int                    rc;
    ngx_uint_t             flush;
    ngx_chain_t           *cl;
    ngx_http_gzip_ctx_t   *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_gzip_filter_module);

    if (ctx == NULL || ctx->done || r->header_only) {
        return ngx_http_next_body_filter(r, in);
    }

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http gzip filter");

    if (ctx->buffering) {

        /*
         * With default memory settings zlib starts to output gzipped data
         * only after it has got about 90K, so it makes sense to allocate
         * zlib memory (200-400K) only after we have enough data to compress.
         * Although we copy buffers, nevertheless for not big responses
         * this allows to allocate zlib memory, to compress and to output
         * the response in one step using hot CPU cache.
         */

        if (in) {
            switch (ngx_http_gzip_filter_buffer(ctx, in)) {

            case NGX_OK:
                return NGX_OK;

            case NGX_DONE:
                in = NULL;
                break;

            default:  /* NGX_ERROR */
                goto failed;
            }

        } else {
            ctx->buffering = 0;
        }
    }

    if (ctx->preallocated == NULL) {
        if (ngx_http_gzip_filter_deflate_start(r, ctx) != NGX_OK) {
```

```
            goto failed;
        }
    }

    if (in) {
        if (ngx_chain_add_copy(r->pool, &ctx->in, in) != NGX_OK) {
            goto failed;
        }

        r->connection->buffered |= NGX_HTTP_GZIP_BUFFERED;
    }

    if (ctx->nomem) {

        /* flush busy buffers */

        if (ngx_http_next_body_filter(r, NULL) == NGX_ERROR) {
            goto failed;
        }

        cl = NULL;

        ngx_chain_update_chains(r->pool, &ctx->free, &ctx->busy, &cl,
                                (ngx_buf_tag_t) &ngx_http_gzip_filter_module);
        ctx->nomem = 0;
        flush = 0;

    } else {
        flush = ctx->busy ? 1 : 0;
    }

    for ( ;; ) {

        /* cycle while we can write to a client */

        for ( ;; ) {

            /* cycle while there is data to feed zlib and ... */

            rc = ngx_http_gzip_filter_add_data(r, ctx);

            if (rc == NGX_DECLINED) {
                break;
            }

            if (rc == NGX_AGAIN) {
                continue;
            }


            /* ... there are buffers to write zlib output */

            rc = ngx_http_gzip_filter_get_buf(r, ctx);

            if (rc == NGX_DECLINED) {
                break;
            }

            if (rc == NGX_ERROR) {
                goto failed;
            }


            rc = ngx_http_gzip_filter_deflate(r, ctx);

            if (rc == NGX_OK) {
                break;
            }

            if (rc == NGX_ERROR) {
                goto failed;
            }

            /* rc == NGX_AGAIN */
        }

```

```
        if (ctx->out == NULL && !flush) {
            ngx_http_gzip_filter_free_copy_buf(r, ctx);

            return ctx->busy ? NGX_AGAIN : NGX_OK;
        }

        if (!ctx->gzheader) {
            if (ngx_http_gzip_filter_gzheader(r, ctx) != NGX_OK) {
                goto failed;
            }
        }

        rc = ngx_http_next_body_filter(r, ctx->out);

        if (rc == NGX_ERROR) {
            goto failed;
        }

        ngx_http_gzip_filter_free_copy_buf(r, ctx);

        ngx_chain_update_chains(r->pool, &ctx->free, &ctx->busy, &ctx->out,
                                (ngx_buf_tag_t) &ngx_http_gzip_filter_module);
        ctx->last_out = &ctx->out;

        ctx->nomem = 0;
        flush = 0;

        if (ctx->done) {
            return rc;
        }
    }

    /* unreachable */

failed:

    ctx->done = 1;

    if (ctx->preallocated) {
        deflateEnd(&ctx->zstream);

        ngx_pfree(r->pool, ctx->preallocated);
    }

    ngx_http_gzip_filter_free_copy_buf(r, ctx);

    return NGX_ERROR;
}


static void
ngx_http_gzip_filter_memory(ngx_http_request_t *r, ngx_http_gzip_ctx_t *ctx)
{
    int                    wbits, memlevel;
    ngx_http_gzip_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_filter_module);

    wbits = conf->wbits;
    memlevel = conf->memlevel;

    if (r->headers_out.content_length_n > 0) {

        /* the actual zlib window size is smaller by 262 bytes */

        while (r->headers_out.content_length_n < ((1 << (wbits - 1)) - 262)) {
            wbits--;
            memlevel--;
        }

        if (memlevel < 1) {
            memlevel = 1;
        }
    }

    ctx->wbits = wbits;
```

```
    ctx->memlevel = memlevel;

    /*
     * We preallocate a memory for zlib in one buffer (200K-400K), this
     * decreases a number of malloc() and free() calls and also probably
     * decreases a number of syscalls (sbrk()/mmap() and so on).
     * Besides we free the memory as soon as a gzipping will complete
     * and do not wait while a whole response will be sent to a client.
     *
     * 8K is for zlib deflate_state, it takes
     *  *) 5816 bytes on i386 and sparc64 (32-bit mode)
     *  *) 5920 bytes on amd64 and sparc64
     */

    ctx->allocated = 8192 + (1 << (wbits + 2)) + (1 << (memlevel + 9));
}


static ngx_int_t
ngx_http_gzip_filter_buffer(ngx_http_gzip_ctx_t *ctx, ngx_chain_t *in)
{
    size_t                 size, buffered;
    ngx_buf_t             *b, *buf;
    ngx_chain_t           *cl, **ll;
    ngx_http_request_t    *r;
    ngx_http_gzip_conf_t  *conf;

    r = ctx->request;

    r->connection->buffered |= NGX_HTTP_GZIP_BUFFERED;

    buffered = 0;
    ll = &ctx->in;

    for (cl = ctx->in; cl; cl = cl->next) {
        buffered += cl->buf->last - cl->buf->pos;
        ll = &cl->next;
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_filter_module);

    while (in) {
        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        b = in->buf;

        size = b->last - b->pos;
        buffered += size;

        if (b->flush || b->last_buf || buffered > conf->postpone_gzipping) {
            ctx->buffering = 0;
        }

        if (ctx->buffering && size) {

            buf = ngx_create_temp_buf(r->pool, size);
            if (buf == NULL) {
                return NGX_ERROR;
            }

            buf->last = ngx_cpymem(buf->pos, b->pos, size);
            b->pos = b->last;

            buf->last_buf = b->last_buf;
            buf->tag = (ngx_buf_tag_t) &ngx_http_gzip_filter_module;

            cl->buf = buf;

        } else {
            cl->buf = b;
        }

        *ll = cl;
```

```
        ll = &cl->next;
        in = in->next;
    }

    *ll = NULL;

    return ctx->buffering ? NGX_OK : NGX_DONE;
}


static ngx_int_t
ngx_http_gzip_filter_deflate_start(ngx_http_request_t *r,
    ngx_http_gzip_ctx_t *ctx)
{
    int                  rc;
    ngx_http_gzip_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_filter_module);

    ctx->preallocated = ngx_palloc(r->pool, ctx->allocated);
    if (ctx->preallocated == NULL) {
        return NGX_ERROR;
    }

    ctx->free_mem = ctx->preallocated;

    ctx->zstream.zalloc = ngx_http_gzip_filter_alloc;
    ctx->zstream.zfree = ngx_http_gzip_filter_free;
    ctx->zstream.opaque = ctx;

    rc = deflateInit2(&ctx->zstream, (int) conf->level, Z_DEFLATED,
                      - ctx->wbits, ctx->memlevel, Z_DEFAULT_STRATEGY);

    if (rc != Z_OK) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      "deflateInit2() failed: %d", rc);
        return NGX_ERROR;
    }

    ctx->last_out = &ctx->out;
    ctx->crc32 = crc32(0L, Z_NULL, 0);
    ctx->flush = Z_NO_FLUSH;

    return NGX_OK;
}


static ngx_int_t
ngx_http_gzip_filter_gzheader(ngx_http_request_t *r, ngx_http_gzip_ctx_t *ctx)
{
    ngx_buf_t      *b;
    ngx_chain_t    *cl;
    static u_char  gzheader[10] =
                               { 0x1f, 0x8b, Z_DEFLATED, 0, 0, 0, 0, 0, 0, 3 };

    b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
    if (b == NULL) {
        return NGX_ERROR;
    }

    b->memory = 1;
    b->pos = gzheader;
    b->last = b->pos + 10;

    cl = ngx_alloc_chain_link(r->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    cl->buf = b;
    cl->next = ctx->out;
    ctx->out = cl;

    ctx->gzheader = 1;

    return NGX_OK;
```

```
}


static ngx_int_t
ngx_http_gzip_filter_add_data(ngx_http_request_t *r, ngx_http_gzip_ctx_t *ctx)
{
    if (ctx->zstream.avail_in || ctx->flush != Z_NO_FLUSH || ctx->redo) {
        return NGX_OK;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gzip in: %p", ctx->in);

    if (ctx->in == NULL) {
        return NGX_DECLINED;
    }

    if (ctx->copy_buf) {

        /*
         * to avoid CPU cache trashing we do not free() just quit buf,
         * but postpone free()ing after zlib compressing and data output
         */

        ctx->copy_buf->next = ctx->copied;
        ctx->copied = ctx->copy_buf;
        ctx->copy_buf = NULL;
    }

    ctx->in_buf = ctx->in->buf;

    if (ctx->in_buf->tag == (ngx_buf_tag_t) &ngx_http_gzip_filter_module) {
        ctx->copy_buf = ctx->in;
    }

    ctx->in = ctx->in->next;

    ctx->zstream.next_in = ctx->in_buf->pos;
    ctx->zstream.avail_in = ctx->in_buf->last - ctx->in_buf->pos;

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gzip in_buf:%p ni:%p ai:%ud",
                   ctx->in_buf,
                   ctx->zstream.next_in, ctx->zstream.avail_in);

    if (ctx->in_buf->last_buf) {
        ctx->flush = Z_FINISH;

    } else if (ctx->in_buf->flush) {
        ctx->flush = Z_SYNC_FLUSH;
    }

    if (ctx->zstream.avail_in) {

        ctx->crc32 = crc32(ctx->crc32, ctx->zstream.next_in,
                           ctx->zstream.avail_in);

    } else if (ctx->flush == Z_NO_FLUSH) {
        return NGX_AGAIN;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_gzip_filter_get_buf(ngx_http_request_t *r, ngx_http_gzip_ctx_t *ctx)
{
    ngx_http_gzip_conf_t  *conf;

    if (ctx->zstream.avail_out) {
        return NGX_OK;
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_filter_module);

```

```
    if (ctx->free) {
        ctx->out_buf = ctx->free->buf;
        ctx->free = ctx->free->next;

    } else if (ctx->bufs < conf->bufs.num) {

        ctx->out_buf = ngx_create_temp_buf(r->pool, conf->bufs.size);
        if (ctx->out_buf == NULL) {
            return NGX_ERROR;
        }

        ctx->out_buf->tag = (ngx_buf_tag_t) &ngx_http_gzip_filter_module;
        ctx->out_buf->recycled = 1;
        ctx->bufs++;

    } else {
        ctx->nomem = 1;
        return NGX_DECLINED;
    }

    ctx->zstream.next_out = ctx->out_buf->pos;
    ctx->zstream.avail_out = conf->bufs.size;

    return NGX_OK;
}


static ngx_int_t
ngx_http_gzip_filter_deflate(ngx_http_request_t *r, ngx_http_gzip_ctx_t *ctx)
{
    int                    rc;
    ngx_buf_t             *b;
    ngx_chain_t           *cl;
    ngx_http_gzip_conf_t  *conf;

    ngx_log_debug6(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "deflate in: ni:%p no:%p ai:%ud ao:%ud fl:%d redo:%d",
                   ctx->zstream.next_in, ctx->zstream.next_out,
                   ctx->zstream.avail_in, ctx->zstream.avail_out,
                   ctx->flush, ctx->redo);

    rc = deflate(&ctx->zstream, ctx->flush);

    if (rc != Z_OK && rc != Z_STREAM_END && rc != Z_BUF_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                      "deflate() failed: %d, %d", ctx->flush, rc);
        return NGX_ERROR;
    }

    ngx_log_debug5(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "deflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
                   ctx->zstream.next_in, ctx->zstream.next_out,
                   ctx->zstream.avail_in, ctx->zstream.avail_out,
                   rc);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "gzip in_buf:%p pos:%p",
                   ctx->in_buf, ctx->in_buf->pos);

    if (ctx->zstream.next_in) {
        ctx->in_buf->pos = ctx->zstream.next_in;

        if (ctx->zstream.avail_in == 0) {
            ctx->zstream.next_in = NULL;
        }
    }

    ctx->out_buf->last = ctx->zstream.next_out;

    if (ctx->zstream.avail_out == 0) {

        /* zlib wants to output some more gzipped data */

        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
```

```
        }

        cl->buf = ctx->out_buf;
        cl->next = NULL;
        *ctx->last_out = cl;
        ctx->last_out = &cl->next;

        ctx->redo = 1;

        return NGX_AGAIN;
    }

    ctx->redo = 0;

    if (ctx->flush == Z_SYNC_FLUSH) {

        ctx->flush = Z_NO_FLUSH;

        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        b = ctx->out_buf;

        if (ngx_buf_size(b) == 0) {

            b = ngx_calloc_buf(ctx->request->pool);
            if (b == NULL) {
                return NGX_ERROR;
            }

        } else {
            ctx->zstream.avail_out = 0;
        }

        b->flush = 1;

        cl->buf = b;
        cl->next = NULL;
        *ctx->last_out = cl;
        ctx->last_out = &cl->next;

        r->connection->buffered &= ~NGX_HTTP_GZIP_BUFFERED;

        return NGX_OK;
    }

    if (rc == Z_STREAM_END) {

        if (ngx_http_gzip_filter_deflate_end(r, ctx) != NGX_OK) {
            return NGX_ERROR;
        }

        return NGX_OK;
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_filter_module);

    if (conf->no_buffer && ctx->in == NULL) {

        cl = ngx_alloc_chain_link(r->pool);
        if (cl == NULL) {
            return NGX_ERROR;
        }

        cl->buf = ctx->out_buf;
        cl->next = NULL;
        *ctx->last_out = cl;
        ctx->last_out = &cl->next;

        return NGX_OK;
    }

    return NGX_AGAIN;
}
```

```
896
897
898 static ngx_int_t
899 ngx_http_gzip_filter_deflate_end(ngx_http_request_t *r,
900     ngx_http_gzip_ctx_t *ctx)
901 {
902     int               rc;
903     ngx_buf_t         *b;
904     ngx_chain_t       *cl;
905     struct gztrailer  *trailer;
906
907     ctx->zin = ctx->zstream.total_in;
908     ctx->zout = 10 + ctx->zstream.total_out + 8;
909
910     rc = deflateEnd(&ctx->zstream);
911
912     if (rc != Z_OK) {
913         ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
914                       "deflateEnd() failed: %d", rc);
915         return NGX_ERROR;
916     }
917
918     ngx_pfree(r->pool, ctx->preallocated);
919
920     cl = ngx_alloc_chain_link(r->pool);
921     if (cl == NULL) {
922         return NGX_ERROR;
923     }
924
925     cl->buf = ctx->out_buf;
926     cl->next = NULL;
927     *ctx->last_out = cl;
928     ctx->last_out = &cl->next;
929
930     if (ctx->zstream.avail_out >= 8) {
931         trailer = (struct gztrailer *) ctx->out_buf->last;
932         ctx->out_buf->last += 8;
933         ctx->out_buf->last_buf = 1;
934
935     } else {
936         b = ngx_create_temp_buf(r->pool, 8);
937         if (b == NULL) {
938             return NGX_ERROR;
939         }
940
941         b->last_buf = 1;
942
943         cl = ngx_alloc_chain_link(r->pool);
944         if (cl == NULL) {
945             return NGX_ERROR;
946         }
947
948         cl->buf = b;
949         cl->next = NULL;
950         *ctx->last_out = cl;
951         ctx->last_out = &cl->next;
952         trailer = (struct gztrailer *) b->pos;
953         b->last += 8;
954     }
955
956 #if (NGX_HAVE_LITTLE_ENDIAN && NGX_HAVE_NONALIGNED)
957
958     trailer->crc32 = ctx->crc32;
959     trailer->zlen = ctx->zin;
960
961 #else
962
963     trailer->crc32[0] = (u_char) (ctx->crc32 & 0xff);
964     trailer->crc32[1] = (u_char) ((ctx->crc32 >> 8) & 0xff);
965     trailer->crc32[2] = (u_char) ((ctx->crc32 >> 16) & 0xff);
966     trailer->crc32[3] = (u_char) ((ctx->crc32 >> 24) & 0xff);
967
968     trailer->zlen[0] = (u_char) (ctx->zin & 0xff);
969     trailer->zlen[1] = (u_char) ((ctx->zin >> 8) & 0xff);
970     trailer->zlen[2] = (u_char) ((ctx->zin >> 16) & 0xff);
971     trailer->zlen[3] = (u_char) ((ctx->zin >> 24) & 0xff);
```

```
#endif

    ctx->zstream.avail_in = 0;
    ctx->zstream.avail_out = 0;

    ctx->done = 1;

    r->connection->buffered &= ~NGX_HTTP_GZIP_BUFFERED;

    return NGX_OK;
}


static void *
ngx_http_gzip_filter_alloc(void *opaque, u_int items, u_int size)
{
    ngx_http_gzip_ctx_t *ctx = opaque;

    void        *p;
    ngx_uint_t   alloc;

    alloc = items * size;

    if (alloc % 512 != 0 && alloc < 8192) {

        /*
         * The zlib deflate_state allocation, it takes about 6K,
         * we allocate 8K.  Other allocations are divisible by 512.
         */

        alloc = 8192;
    }

    if (alloc <= ctx->allocated) {
        p = ctx->free_mem;
        ctx->free_mem += alloc;
        ctx->allocated -= alloc;

        ngx_log_debug4(NGX_LOG_DEBUG_HTTP, ctx->request->connection->log, 0,
                       "gzip alloc: n:%ud s:%ud a:%ud p:%p",
                       items, size, alloc, p);

        return p;
    }

    ngx_log_error(NGX_LOG_ALERT, ctx->request->connection->log, 0,
                  "gzip filter failed to use preallocated memory: %ud of %ud",
                  items * size, ctx->allocated);

    p = ngx_palloc(ctx->request->pool, items * size);

    return p;
}


static void
ngx_http_gzip_filter_free(void *opaque, void *address)
{
#if 0
    ngx_http_gzip_ctx_t *ctx = opaque;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, ctx->request->connection->log, 0,
                   "gzip free: %p", address);
#endif
}


static void
ngx_http_gzip_filter_free_copy_buf(ngx_http_request_t *r,
    ngx_http_gzip_ctx_t *ctx)
{
    ngx_chain_t  *cl;

    for (cl = ctx->copied; cl; cl = cl->next) {
        ngx_pfree(r->pool, cl->buf->start);
```

```
    }

    ctx->copied = NULL;
}


static ngx_int_t
ngx_http_gzip_add_variables(ngx_conf_t *cf)
{
    ngx_http_variable_t  *var;

    var = ngx_http_add_variable(cf, &ngx_http_gzip_ratio, NGX_HTTP_VAR_NOHASH);
    if (var == NULL) {
        return NGX_ERROR;
    }

    var->get_handler = ngx_http_gzip_ratio_variable;

    return NGX_OK;
}


static ngx_int_t
ngx_http_gzip_ratio_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_uint_t            zint, zfrac;
    ngx_http_gzip_ctx_t  *ctx;

    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    ctx = ngx_http_get_module_ctx(r, ngx_http_gzip_filter_module);

    if (ctx == NULL || ctx->zout == 0) {
        v->not_found = 1;
        return NGX_OK;
    }

    v->data = ngx_pnalloc(r->pool, NGX_INT32_LEN + 3);
    if (v->data == NULL) {
        return NGX_ERROR;
    }

    zint = (ngx_uint_t) (ctx->zin / ctx->zout);
    zfrac = (ngx_uint_t) ((ctx->zin * 100 / ctx->zout) % 100);

    if ((ctx->zin * 1000 / ctx->zout) % 10 > 4) {

        /* the rounding, e.g., 2.125 to 2.13 */

        zfrac++;

        if (zfrac > 99) {
            zint++;
            zfrac = 0;
        }
    }

    v->len = ngx_sprintf(v->data, "%ui.%02ui", zint, zfrac) - v->data;

    return NGX_OK;
}


static void *
ngx_http_gzip_create_conf(ngx_conf_t *cf)
{
    ngx_http_gzip_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_gzip_conf_t));
    if (conf == NULL) {
        return NULL;
    }

```

```
    /*
     * set by ngx_pcalloc():
     *
     *     conf->bufs.num = 0;
     *     conf->types = { NULL };
     *     conf->types_keys = NULL;
     */

    conf->enable = NGX_CONF_UNSET;
    conf->no_buffer = NGX_CONF_UNSET;

    conf->postpone_gzipping = NGX_CONF_UNSET_SIZE;
    conf->level = NGX_CONF_UNSET;
    conf->wbits = NGX_CONF_UNSET_SIZE;
    conf->memlevel = NGX_CONF_UNSET_SIZE;
    conf->min_length = NGX_CONF_UNSET;

    return conf;
}


static char *
ngx_http_gzip_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_gzip_conf_t *prev = parent;
    ngx_http_gzip_conf_t *conf = child;

    ngx_conf_merge_value(conf->enable, prev->enable, 0);
    ngx_conf_merge_value(conf->no_buffer, prev->no_buffer, 0);

    ngx_conf_merge_bufs_value(conf->bufs, prev->bufs,
                              (128 * 1024) / ngx_pagesize, ngx_pagesize);

    ngx_conf_merge_size_value(conf->postpone_gzipping, prev->postpone_gzipping,
                              0);
    ngx_conf_merge_value(conf->level, prev->level, 1);
    ngx_conf_merge_size_value(conf->wbits, prev->wbits, MAX_WBITS);
    ngx_conf_merge_size_value(conf->memlevel, prev->memlevel,
                              MAX_MEM_LEVEL - 1);
    ngx_conf_merge_value(conf->min_length, prev->min_length, 20);

    if (ngx_http_merge_types(cf, &conf->types_keys, &conf->types,
                             &prev->types_keys, &prev->types,
                             ngx_http_html_default_types)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_gzip_filter_init(ngx_conf_t *cf)
{
    ngx_http_next_header_filter = ngx_http_top_header_filter;
    ngx_http_top_header_filter = ngx_http_gzip_header_filter;

    ngx_http_next_body_filter = ngx_http_top_body_filter;
    ngx_http_top_body_filter = ngx_http_gzip_body_filter;

    return NGX_OK;
}


static char *
ngx_http_gzip_window(ngx_conf_t *cf, void *post, void *data)
{
    size_t *np = data;

    size_t  wbits, wsize;

    wbits = 15;

    for (wsize = 32 * 1024; wsize > 256; wsize >>= 1) {
```

```
1200
1201            if (wsize == *np) {
1202                *np = wbits;
1203
1204                return NGX_CONF_OK;
1205            }
1206
1207            wbits--;
1208        }
1209
1210        return "must be 512, 1k, 2k, 4k, 8k, 16k, or 32k";
1211    }
1212
1213
1214    static char *
1215    ngx_http_gzip_hash(ngx_conf_t *cf, void *post, void *data)
1216    {
1217        size_t *np = data;
1218
1219        size_t  memlevel, hsize;
1220
1221        memlevel = 9;
1222
1223        for (hsize = 128 * 1024; hsize > 256; hsize >>= 1) {
1224
1225            if (hsize == *np) {
1226                *np = memlevel;
1227
1228                return NGX_CONF_OK;
1229            }
1230
1231            memlevel--;
1232        }
1233
1234        return "must be 512, 1k, 2k, 4k, 8k, 16k, 32k, 64k, or 128k";
1235    }
```

One Level Up Top Level

src/http/modules/ngx_http_gzip_static_module.c - nginx-1.7.10

Global variables defined

- ngx_http_gzip_static
- ngx_http_gzip_static_commands
- ngx_http_gzip_static_module
- ngx_http_gzip_static_module_ctx

Data types defined

- ngx_http_gzip_static_conf_t

Functions defined

- ngx_http_gzip_static_create_conf
- ngx_http_gzip_static_handler
- ngx_http_gzip_static_init
- ngx_http_gzip_static_merge_conf

Macros defined

- NGX_HTTP_GZIP_STATIC_ALWAYS
- NGX_HTTP_GZIP_STATIC_OFF
- NGX_HTTP_GZIP_STATIC_ON

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define NGX_HTTP_GZIP_STATIC_OFF     0
14  #define NGX_HTTP_GZIP_STATIC_ON      1
15  #define NGX_HTTP_GZIP_STATIC_ALWAYS  2
16
17
18  typedef struct {
19      ngx_uint_t  enable;
20  } ngx_http_gzip_static_conf_t;
21
22
23  static ngx_int_t ngx_http_gzip_static_handler(ngx_http_request_t *r);
24  static void *ngx_http_gzip_static_create_conf(ngx_conf_t *cf);
25  static char *ngx_http_gzip_static_merge_conf(ngx_conf_t *cf, void *parent,
26      void *child);
27  static ngx_int_t ngx_http_gzip_static_init(ngx_conf_t *cf);
```

```
28
29
30  static ngx_conf_enum_t  ngx_http_gzip_static[] = {
31      { ngx_string("off"), NGX_HTTP_GZIP_STATIC_OFF },
32      { ngx_string("on"), NGX_HTTP_GZIP_STATIC_ON },
33      { ngx_string("always"), NGX_HTTP_GZIP_STATIC_ALWAYS },
34      { ngx_null_string, 0 }
35  };
36
37
38  static ngx_command_t  ngx_http_gzip_static_commands[] = {
39
40      { ngx_string("gzip_static"),
41        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
42        ngx_conf_set_enum_slot,
43        NGX_HTTP_LOC_CONF_OFFSET,
44        offsetof(ngx_http_gzip_static_conf_t, enable),
45        &ngx_http_gzip_static },
46
47        ngx_null_command
48  };
49
50
51  ngx_http_module_t  ngx_http_gzip_static_module_ctx = {
52      NULL,                                  /* preconfiguration */
53      ngx_http_gzip_static_init,             /* postconfiguration */
54
55      NULL,                                  /* create main configuration */
56      NULL,                                  /* init main configuration */
57
58      NULL,                                  /* create server configuration */
59      NULL,                                  /* merge server configuration */
60
61      ngx_http_gzip_static_create_conf,      /* create location configuration */
62      ngx_http_gzip_static_merge_conf        /* merge location configuration */
63  };
64
65
66  ngx_module_t  ngx_http_gzip_static_module = {
67      NGX_MODULE_V1,
68      &ngx_http_gzip_static_module_ctx,      /* module context */
69      ngx_http_gzip_static_commands,         /* module directives */
70      NGX_HTTP_MODULE,                       /* module type */
71      NULL,                                  /* init master */
72      NULL,                                  /* init module */
73      NULL,                                  /* init process */
74      NULL,                                  /* init thread */
75      NULL,                                  /* exit thread */
76      NULL,                                  /* exit process */
77      NULL,                                  /* exit master */
78      NGX_MODULE_V1_PADDING
79  };
80
81
82  static ngx_int_t
83  ngx_http_gzip_static_handler(ngx_http_request_t *r)
84  {
85      u_char                     *p;
86      size_t                      root;
87      ngx_str_t                   path;
88      ngx_int_t                   rc;
89      ngx_uint_t                  level;
90      ngx_log_t                  *log;
91      ngx_buf_t                  *b;
92      ngx_chain_t                 out;
93      ngx_table_elt_t            *h;
94      ngx_open_file_info_t        of;
95      ngx_http_core_loc_conf_t   *clcf;
96      ngx_http_gzip_static_conf_t  *gzcf;
97
98      if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD))) {
99          return NGX_DECLINED;
100     }
101
102     if (r->uri.data[r->uri.len - 1] == '/') {
103         return NGX_DECLINED;
```

```
    }

    gzcf = ngx_http_get_module_loc_conf(r, ngx_http_gzip_static_module);

    if (gzcf->enable == NGX_HTTP_GZIP_STATIC_OFF) {
        return NGX_DECLINED;
    }

    if (gzcf->enable == NGX_HTTP_GZIP_STATIC_ON) {
        rc = ngx_http_gzip_ok(r);

    } else {
        /* always */
        rc = NGX_OK;
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (!clcf->gzip_vary && rc != NGX_OK) {
        return NGX_DECLINED;
    }

    log = r->connection->log;

    p = ngx_http_map_uri_to_path(r, &path, &root, sizeof(".gz") - 1);
    if (p == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    *p++ = '.';
    *p++ = 'g';
    *p++ = 'z';
    *p = '\0';

    path.len = p - path.data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, log, 0,
                   "http filename: \"%s\"", path.data);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.read_ahead = clcf->read_ahead;
    of.directio = clcf->directio;
    of.valid = clcf->open_file_cache_valid;
    of.min_uses = clcf->open_file_cache_min_uses;
    of.errors = clcf->open_file_cache_errors;
    of.events = clcf->open_file_cache_events;

    if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
        != NGX_OK)
    {
        switch (of.err) {

        case 0:
            return NGX_HTTP_INTERNAL_SERVER_ERROR;

        case NGX_ENOENT:
        case NGX_ENOTDIR:
        case NGX_ENAMETOOLONG:

            return NGX_DECLINED;

        case NGX_EACCES:
#if (NGX_HAVE_OPENAT)
        case NGX_EMLINK:
        case NGX_ELOOP:
#endif

            level = NGX_LOG_ERR;
            break;

        default:
```

```
            level = NGX_LOG_CRIT;
            break;
        }

        ngx_log_error(level, log, of.err,
                      "%s \"%s\" failed", of.failed, path.data);

        return NGX_DECLINED;
    }

    if (gzcf->enable == NGX_HTTP_GZIP_STATIC_ON) {
        r->gzip_vary = 1;

        if (rc != NGX_OK) {
            return NGX_DECLINED;
        }
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, log, 0, "http static fd: %d", of.fd);

    if (of.is_dir) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, log, 0, "http dir");
        return NGX_DECLINED;
    }

#if !(NGX_WIN32) /* the not regular files are probably Unix specific */

    if (!of.is_file) {
        ngx_log_error(NGX_LOG_CRIT, log, 0,
                      "\"%s\" is not a regular file", path.data);

        return NGX_HTTP_NOT_FOUND;
    }

#endif

    r->root_tested = !r->error_page;

    rc = ngx_http_discard_request_body(r);

    if (rc != NGX_OK) {
        return rc;
    }

    log->action = "sending response to client";

    r->headers_out.status = NGX_HTTP_OK;
    r->headers_out.content_length_n = of.size;
    r->headers_out.last_modified_time = of.mtime;

    if (ngx_http_set_etag(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (ngx_http_set_content_type(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    h = ngx_list_push(&r->headers_out.headers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    h->hash = 1;
    ngx_str_set(&h->key, "Content-Encoding");
    ngx_str_set(&h->value, "gzip");
    r->headers_out.content_encoding = h;

    /* we need to allocate all before the header would be sent */

    b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
    if (b == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

```

```
    b->file = ngx_pcalloc(r->pool, sizeof(ngx_file_t));
    if (b->file == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
        return rc;
    }

    b->file_pos = 0;
    b->file_last = of.size;

    b->in_file = b->file_last ? 1 : 0;
    b->last_buf = (r == r->main) ? 1 : 0;
    b->last_in_chain = 1;

    b->file->fd = of.fd;
    b->file->name = path;
    b->file->log = log;
    b->file->directio = of.is_directio;

    out.buf = b;
    out.next = NULL;

    return ngx_http_output_filter(r, &out);
}


static void *
ngx_http_gzip_static_create_conf(ngx_conf_t *cf)
{
    ngx_http_gzip_static_conf_t  *conf;

    conf = ngx_palloc(cf->pool, sizeof(ngx_http_gzip_static_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->enable = NGX_CONF_UNSET_UINT;

    return conf;
}


static char *
ngx_http_gzip_static_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_gzip_static_conf_t *prev = parent;
    ngx_http_gzip_static_conf_t *conf = child;

    ngx_conf_merge_uint_value(conf->enable, prev->enable,
                              NGX_HTTP_GZIP_STATIC_OFF);

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_gzip_static_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt        *h;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    *h = ngx_http_gzip_static_handler;

    return NGX_OK;
}
```

src/http/modules/ngx_http_headers_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_headers_filter_commands
- ngx_http_headers_filter_module
- ngx_http_headers_filter_module_ctx
- ngx_http_next_header_filter
- ngx_http_set_headers

Data types defined

- ngx_http_expires_t
- ngx_http_header_val_s
- ngx_http_header_val_t
- ngx_http_headers_conf_t
- ngx_http_set_header_pt
- ngx_http_set_header_t

Functions defined

- ngx_http_add_cache_control
- ngx_http_add_header
- ngx_http_headers_add
- ngx_http_headers_create_conf
- ngx_http_headers_expires
- ngx_http_headers_filter
- ngx_http_headers_filter_init
- ngx_http_headers_merge_conf
- ngx_http_parse_expires
- ngx_http_set_expires
- ngx_http_set_last_modified
- ngx_http_set_response_header

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
```

```
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct ngx_http_header_val_s  ngx_http_header_val_t;
14
15  typedef ngx_int_t (*ngx_http_set_header_pt)(ngx_http_request_t *r,
16      ngx_http_header_val_t *hv, ngx_str_t *value);
17
18
19  typedef struct {
20      ngx_str_t                  name;
21      ngx_uint_t                 offset;
22      ngx_http_set_header_pt     handler;
23  } ngx_http_set_header_t;
24
25
26  struct ngx_http_header_val_s {
27      ngx_http_complex_value_t   value;
28      ngx_str_t                  key;
29      ngx_http_set_header_pt     handler;
30      ngx_uint_t                 offset;
31      ngx_uint_t                 always;  /* unsigned  always:1 */
32  };
33
34
35  typedef enum {
36      NGX_HTTP_EXPIRES_OFF,
37      NGX_HTTP_EXPIRES_EPOCH,
38      NGX_HTTP_EXPIRES_MAX,
39      NGX_HTTP_EXPIRES_ACCESS,
40      NGX_HTTP_EXPIRES_MODIFIED,
41      NGX_HTTP_EXPIRES_DAILY,
42      NGX_HTTP_EXPIRES_UNSET
43  } ngx_http_expires_t;
44
45
46  typedef struct {
47      ngx_http_expires_t         expires;
48      time_t                     expires_time;
49      ngx_http_complex_value_t  *expires_value;
50      ngx_array_t               *headers;
51  } ngx_http_headers_conf_t;
52
53
54  static ngx_int_t ngx_http_set_expires(ngx_http_request_t *r,
55      ngx_http_headers_conf_t *conf);
56  static ngx_int_t ngx_http_parse_expires(ngx_str_t *value,
57      ngx_http_expires_t *expires, time_t *expires_time, char **err);
58  static ngx_int_t ngx_http_add_cache_control(ngx_http_request_t *r,
59      ngx_http_header_val_t *hv, ngx_str_t *value);
60  static ngx_int_t ngx_http_add_header(ngx_http_request_t *r,
61      ngx_http_header_val_t *hv, ngx_str_t *value);
62  static ngx_int_t ngx_http_set_last_modified(ngx_http_request_t *r,
63      ngx_http_header_val_t *hv, ngx_str_t *value);
64  static ngx_int_t ngx_http_set_response_header(ngx_http_request_t *r,
65      ngx_http_header_val_t *hv, ngx_str_t *value);
66
67  static void *ngx_http_headers_create_conf(ngx_conf_t *cf);
68  static char *ngx_http_headers_merge_conf(ngx_conf_t *cf,
69      void *parent, void *child);
70  static ngx_int_t ngx_http_headers_filter_init(ngx_conf_t *cf);
71  static char *ngx_http_headers_expires(ngx_conf_t *cf, ngx_command_t *cmd,
72      void *conf);
73  static char *ngx_http_headers_add(ngx_conf_t *cf, ngx_command_t *cmd,
74      void *conf);
75
76
77  static ngx_http_set_header_t  ngx_http_set_headers[] = {
78
79      { ngx_string("Cache-Control"), 0, ngx_http_add_cache_control },
80
81      { ngx_string("Last-Modified"),
```

```
                offsetof(ngx_http_headers_out_t, last_modified),
                ngx_http_set_last_modified },

    { ngx_string("ETag"),
                offsetof(ngx_http_headers_out_t, etag),
                ngx_http_set_response_header },

    { ngx_null_string, 0, NULL }
};


static ngx_command_t  ngx_http_headers_filter_commands[] = {

    { ngx_string("expires"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                        |NGX_CONF_TAKE12,
      ngx_http_headers_expires,
      NGX_HTTP_LOC_CONF_OFFSET,
       0,
       NULL},

     { ngx_string("add_header"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                         |NGX_CONF_TAKE23,
       ngx_http_headers_add,
       NGX_HTTP_LOC_CONF_OFFSET,
       0,
       NULL},

       ngx_null_command
};


static ngx_http_module_t  ngx_http_headers_filter_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_headers_filter_init,          /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_headers_create_conf,          /* create location configuration */
    ngx_http_headers_merge_conf            /* merge location configuration */
};


ngx_module_t  ngx_http_headers_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_headers_filter_module_ctx,   /* module context */
    ngx_http_headers_filter_commands,      /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;


static ngx_int_t
ngx_http_headers_filter(ngx_http_request_t *r)
{
    ngx_str_t                 value;
    ngx_uint_t                i, safe_status;
    ngx_http_header_val_t    *h;
    ngx_http_headers_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r, ngx_http_headers_filter_module);
```

```
158
159     if ((conf->expires == NGX_HTTP_EXPIRES_OFF && conf->headers == NULL)
160         || r != r->main)
161     {
162         return ngx_http_next_header_filter(r);
163     }
164
165     switch (r->headers_out.status) {
166
167     case NGX_HTTP_OK:
168     case NGX_HTTP_CREATED:
169     case NGX_HTTP_NO_CONTENT:
170     case NGX_HTTP_PARTIAL_CONTENT:
171     case NGX_HTTP_MOVED_PERMANENTLY:
172     case NGX_HTTP_MOVED_TEMPORARILY:
173     case NGX_HTTP_SEE_OTHER:
174     case NGX_HTTP_NOT_MODIFIED:
175     case NGX_HTTP_TEMPORARY_REDIRECT:
176         safe_status = 1;
177         break;
178
179     default:
180         safe_status = 0;
181         break;
182     }
183
184     if (conf->expires != NGX_HTTP_EXPIRES_OFF && safe_status) {
185         if (ngx_http_set_expires(r, conf) != NGX_OK) {
186             return NGX_ERROR;
187         }
188     }
189
190     if (conf->headers) {
191         h = conf->headers->elts;
192         for (i = 0; i < conf->headers->nelts; i++) {
193
194             if (!safe_status && !h[i].always) {
195                 continue;
196             }
197
198             if (ngx_http_complex_value(r, &h[i].value, &value) != NGX_OK) {
199                 return NGX_ERROR;
200             }
201
202             if (h[i].handler(r, &h[i], &value) != NGX_OK) {
203                 return NGX_ERROR;
204             }
205         }
206     }
207
208     return ngx_http_next_header_filter(r);
209 }
210
211
212 static ngx_int_t
213 ngx_http_set_expires(ngx_http_request_t *r, ngx_http_headers_conf_t *conf)
214 {
215     char               *err;
216     size_t              len;
217     time_t              now, expires_time, max_age;
218     ngx_str_t           value;
219     ngx_int_t           rc;
220     ngx_uint_t          i;
221     ngx_table_elt_t    *e, *cc, **ccp;
222     ngx_http_expires_t  expires;
223
224     expires = conf->expires;
225     expires_time = conf->expires_time;
226
227     if (conf->expires_value != NULL) {
228
229         if (ngx_http_complex_value(r, conf->expires_value, &value) != NGX_OK) {
230             return NGX_ERROR;
231         }
232
233         rc = ngx_http_parse_expires(&value, &expires, &expires_time, &err);
```

```
234
235         if (rc != NGX_OK) {
236             return NGX_OK;
237         }
238
239         if (expires == NGX_HTTP_EXPIRES_OFF) {
240             return NGX_OK;
241         }
242     }
243
244     e = r->headers_out.expires;
245
246     if (e == NULL) {
247
248         e = ngx_list_push(&r->headers_out.headers);
249         if (e == NULL) {
250             return NGX_ERROR;
251         }
252
253         r->headers_out.expires = e;
254
255         e->hash = 1;
256         ngx_str_set(&e->key, "Expires");
257     }
258
259     len = sizeof("Mon, 28 Sep 1970 06:00:00 GMT");
260     e->value.len = len - 1;
261
262     ccp = r->headers_out.cache_control.elts;
263
264     if (ccp == NULL) {
265
266         if (ngx_array_init(&r->headers_out.cache_control, r->pool,
267                            1, sizeof(ngx_table_elt_t *))
268             != NGX_OK)
269         {
270             return NGX_ERROR;
271         }
272
273         ccp = ngx_array_push(&r->headers_out.cache_control);
274         if (ccp == NULL) {
275             return NGX_ERROR;
276         }
277
278         cc = ngx_list_push(&r->headers_out.headers);
279         if (cc == NULL) {
280             return NGX_ERROR;
281         }
282
283         cc->hash = 1;
284         ngx_str_set(&cc->key, "Cache-Control");
285         *ccp = cc;
286
287     } else {
288         for (i = 1; i < r->headers_out.cache_control.nelts; i++) {
289             ccp[i]->hash = 0;
290         }
291
292         cc = ccp[0];
293     }
294
295     if (expires == NGX_HTTP_EXPIRES_EPOCH) {
296         e->value.data = (u_char *) "Thu, 01 Jan 1970 00:00:01 GMT";
297         ngx_str_set(&cc->value, "no-cache");
298         return NGX_OK;
299     }
300
301     if (expires == NGX_HTTP_EXPIRES_MAX) {
302         e->value.data = (u_char *) "Thu, 31 Dec 2037 23:55:55 GMT";
303         /* 10 years */
304         ngx_str_set(&cc->value, "max-age=315360000");
305         return NGX_OK;
306     }
307
308     e->value.data = ngx_pnalloc(r->pool, len);
309     if (e->value.data == NULL) {
```

```
        return NGX_ERROR;
    }

    if (expires_time == 0 && expires != NGX_HTTP_EXPIRES_DAILY) {
        ngx_memcpy(e->value.data, ngx_cached_http_time.data,
                   ngx_cached_http_time.len + 1);
        ngx_str_set(&cc->value, "max-age=0");
        return NGX_OK;
    }

    now = ngx_time();

    if (expires == NGX_HTTP_EXPIRES_DAILY) {
        expires_time = ngx_next_time(expires_time);
        max_age = expires_time - now;

    } else if (expires == NGX_HTTP_EXPIRES_ACCESS
               || r->headers_out.last_modified_time == -1)
    {
        max_age = expires_time;
        expires_time += now;

    } else {
        expires_time += r->headers_out.last_modified_time;
        max_age = expires_time - now;
    }

    ngx_http_time(e->value.data, expires_time);

    if (conf->expires_time < 0 || max_age < 0) {
        ngx_str_set(&cc->value, "no-cache");
        return NGX_OK;
    }

    cc->value.data = ngx_pnalloc(r->pool,
                                 sizeof("max-age=") + NGX_TIME_T_LEN + 1);
    if (cc->value.data == NULL) {
        return NGX_ERROR;
    }

    cc->value.len = ngx_sprintf(cc->value.data, "max-age=%T", max_age)
                    - cc->value.data;

    return NGX_OK;
}


static ngx_int_t
ngx_http_parse_expires(ngx_str_t *value, ngx_http_expires_t *expires,
    time_t *expires_time, char **err)
{
    ngx_uint_t  minus;

    if (*expires != NGX_HTTP_EXPIRES_MODIFIED) {

        if (value->len == 5 && ngx_strncmp(value->data, "epoch", 5) == 0) {
            *expires = NGX_HTTP_EXPIRES_EPOCH;
            return NGX_OK;
        }

        if (value->len == 3 && ngx_strncmp(value->data, "max", 3) == 0) {
            *expires = NGX_HTTP_EXPIRES_MAX;
            return NGX_OK;
        }

        if (value->len == 3 && ngx_strncmp(value->data, "off", 3) == 0) {
            *expires = NGX_HTTP_EXPIRES_OFF;
            return NGX_OK;
        }
    }

    if (value->data[0] == '@') {
        value->data++;
        value->len--;
        minus = 0;

```

```
        if (*expires == NGX_HTTP_EXPIRES_MODIFIED) {
            *err = "daily time cannot be used with \"modified\" parameter";
            return NGX_ERROR;
        }

        *expires = NGX_HTTP_EXPIRES_DAILY;

    } else if (value->data[0] == '+') {
        value->data++;
        value->len--;
        minus = 0;

    } else if (value->data[0] == '-') {
        value->data++;
        value->len--;
        minus = 1;

    } else {
        minus = 0;
    }

    *expires_time = ngx_parse_time(value, 1);

    if (*expires_time == (time_t) NGX_ERROR) {
        *err = "invalid value";
        return NGX_ERROR;
    }

    if (*expires == NGX_HTTP_EXPIRES_DAILY
        && *expires_time > 24 * 60 * 60)
    {
        *err = "daily time value must be less than 24 hours";
        return NGX_ERROR;
    }

    if (minus) {
        *expires_time = - *expires_time;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_add_header(ngx_http_request_t *r, ngx_http_header_val_t *hv,
    ngx_str_t *value)
{
    ngx_table_elt_t  *h;

    if (value->len) {
        h = ngx_list_push(&r->headers_out.headers);
        if (h == NULL) {
            return NGX_ERROR;
        }

        h->hash = 1;
        h->key = hv->key;
        h->value = *value;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_add_cache_control(ngx_http_request_t *r, ngx_http_header_val_t *hv,
    ngx_str_t *value)
{
    ngx_table_elt_t  *cc, **ccp;

    if (value->len == 0) {
        return NGX_OK;
    }

    ccp = r->headers_out.cache_control.elts;

```

```
    if (ccp == NULL) {

        if (ngx_array_init(&r->headers_out.cache_control, r->pool,
                           1, sizeof(ngx_table_elt_t *))
            != NGX_OK)
        {
            return NGX_ERROR;
        }
    }

    ccp = ngx_array_push(&r->headers_out.cache_control);
    if (ccp == NULL) {
        return NGX_ERROR;
    }

    cc = ngx_list_push(&r->headers_out.headers);
    if (cc == NULL) {
        return NGX_ERROR;
    }

    cc->hash = 1;
    ngx_str_set(&cc->key, "Cache-Control");
    cc->value = *value;

    *ccp = cc;

    return NGX_OK;
}


static ngx_int_t
ngx_http_set_last_modified(ngx_http_request_t *r, ngx_http_header_val_t *hv,
    ngx_str_t *value)
{
    if (ngx_http_set_response_header(r, hv, value) != NGX_OK) {
        return NGX_ERROR;
    }

    r->headers_out.last_modified_time =
        (value->len) ? ngx_http_parse_time(value->data, value->len) : -1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_set_response_header(ngx_http_request_t *r, ngx_http_header_val_t *hv,
    ngx_str_t *value)
{
    ngx_table_elt_t  *h, **old;

    old = (ngx_table_elt_t **) ((char *) &r->headers_out + hv->offset);

    if (value->len == 0) {
        if (*old) {
            (*old)->hash = 0;
            *old = NULL;
        }

        return NGX_OK;
    }

    if (*old) {
        h = *old;

    } else {
        h = ngx_list_push(&r->headers_out.headers);
        if (h == NULL) {
            return NGX_ERROR;
        }

        *old = h;
    }

    h->hash = 1;
    h->key = hv->key;
```

```
538     h->value = *value;
539
540     return NGX_OK;
541 }
542
543
544 static void *
545 ngx_http_headers_create_conf(ngx_conf_t *cf)
546 {
547     ngx_http_headers_conf_t  *conf;
548
549     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_headers_conf_t));
550     if (conf == NULL) {
551         return NULL;
552     }
553
554     /*
555      * set by ngx_pcalloc():
556      *
557      *     conf->headers = NULL;
558      *     conf->expires_time = 0;
559      *     conf->expires_value = NULL;
560      */
561
562     conf->expires = NGX_HTTP_EXPIRES_UNSET;
563
564     return conf;
565 }
566
567
568 static char *
569 ngx_http_headers_merge_conf(ngx_conf_t *cf, void *parent, void *child)
570 {
571     ngx_http_headers_conf_t *prev = parent;
572     ngx_http_headers_conf_t *conf = child;
573
574     if (conf->expires == NGX_HTTP_EXPIRES_UNSET) {
575         conf->expires = prev->expires;
576         conf->expires_time = prev->expires_time;
577         conf->expires_value = prev->expires_value;
578
579         if (conf->expires == NGX_HTTP_EXPIRES_UNSET) {
580             conf->expires = NGX_HTTP_EXPIRES_OFF;
581         }
582     }
583
584     if (conf->headers == NULL) {
585         conf->headers = prev->headers;
586     }
587
588     return NGX_CONF_OK;
589 }
590
591
592 static ngx_int_t
593 ngx_http_headers_filter_init(ngx_conf_t *cf)
594 {
595     ngx_http_next_header_filter = ngx_http_top_header_filter;
596     ngx_http_top_header_filter = ngx_http_headers_filter;
597
598     return NGX_OK;
599 }
600
601
602 static char *
603 ngx_http_headers_expires(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
604 {
605     ngx_http_headers_conf_t *hcf = conf;
606
607     char                               *err;
608     ngx_str_t                          *value;
609     ngx_int_t                           rc;
610     ngx_uint_t                          n;
611     ngx_http_complex_value_t            cv;
612     ngx_http_compile_complex_value_t    ccv;
613
```

```
    if (hcf->expires != NGX_HTTP_EXPIRES_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    if (cf->args->nelts == 2) {

        hcf->expires = NGX_HTTP_EXPIRES_ACCESS;

        n = 1;

    } else { /* cf->args->nelts == 3 */

        if (ngx_strcmp(value[1].data, "modified") != 0) {
            return "invalid value";
        }

        hcf->expires = NGX_HTTP_EXPIRES_MODIFIED;

        n = 2;
    }

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[n];
    ccv.complex_value = &cv;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (cv.lengths != NULL) {

        hcf->expires_value = ngx_palloc(cf->pool,
                                        sizeof(ngx_http_complex_value_t));
        if (hcf->expires_value == NULL) {
            return NGX_CONF_ERROR;
        }

        *hcf->expires_value = cv;

        return NGX_CONF_OK;
    }

    rc = ngx_http_parse_expires(&value[n], &hcf->expires, &hcf->expires_time,
                                &err);
    if (rc != NGX_OK) {
        return err;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_headers_add(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_headers_conf_t *hcf = conf;

    ngx_str_t                          *value;
    ngx_uint_t                          i;
    ngx_http_header_val_t              *hv;
    ngx_http_set_header_t              *set;
    ngx_http_compile_complex_value_t    ccv;

    value = cf->args->elts;

    if (hcf->headers == NULL) {
        hcf->headers = ngx_array_create(cf->pool, 1,
                                        sizeof(ngx_http_header_val_t));
        if (hcf->headers == NULL) {
            return NGX_CONF_ERROR;
        }
    }
```

```
690
691     hv = ngx_array_push(hcf->headers);
692     if (hv == NULL) {
693         return NGX_CONF_ERROR;
694     }
695
696     hv->key = value[1];
697     hv->handler = ngx_http_add_header;
698     hv->offset = 0;
699     hv->always = 0;
700
701     set = ngx_http_set_headers;
702     for (i = 0; set[i].name.len; i++) {
703         if (ngx_strcasecmp(value[1].data, set[i].name.data) != 0) {
704             continue;
705         }
706
707         hv->offset = set[i].offset;
708         hv->handler = set[i].handler;
709
710         break;
711     }
712
713     if (value[2].len == 0) {
714         ngx_memzero(&hv->value, sizeof(ngx_http_complex_value_t));
715         return NGX_CONF_OK;
716     }
717
718     ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
719
720     ccv.cf = cf;
721     ccv.value = &value[2];
722     ccv.complex_value = &hv->value;
723
724     if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
725         return NGX_CONF_ERROR;
726     }
727
728     if (cf->args->nelts == 3) {
729         return NGX_CONF_OK;
730     }
731
732     if (ngx_strcmp(value[3].data, "always") != 0) {
733         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
734                            "invalid parameter \"%V\"", &value[3]);
735         return NGX_CONF_ERROR;
736     }
737
738     hv->always = 1;
739
740     return NGX_CONF_OK;
741 }
```

One Level Up Top Level

src/http/modules/ngx_http_image_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_image_filter_commands
- ngx_http_image_filter_module
- ngx_http_image_filter_module_ctx
- ngx_http_image_types
- ngx_http_next_body_filter
- ngx_http_next_header_filter

Data types defined

- ngx_http_image_filter_conf_t
- ngx_http_image_filter_ctx_t

Functions defined

- ngx_http_image_asis
- ngx_http_image_body_filter
- ngx_http_image_cleanup
- ngx_http_image_filter
- ngx_http_image_filter_create_conf
- ngx_http_image_filter_get_value
- ngx_http_image_filter_init
- ngx_http_image_filter_jpeg_quality
- ngx_http_image_filter_merge_conf
- ngx_http_image_filter_sharpen
- ngx_http_image_filter_value
- ngx_http_image_header_filter
- ngx_http_image_json
- ngx_http_image_length
- ngx_http_image_new
- ngx_http_image_out
- ngx_http_image_process
- ngx_http_image_read
- ngx_http_image_resize

- ngx_http_image_send
- ngx_http_image_size
- ngx_http_image_source
- ngx_http_image_test

Macros defined

- NGX_HTTP_IMAGE_BUFFERED
- NGX_HTTP_IMAGE_CROP
- NGX_HTTP_IMAGE_DONE
- NGX_HTTP_IMAGE_GIF
- NGX_HTTP_IMAGE_JPEG
- NGX_HTTP_IMAGE_NONE
- NGX_HTTP_IMAGE_OFF
- NGX_HTTP_IMAGE_PASS
- NGX_HTTP_IMAGE_PNG
- NGX_HTTP_IMAGE_PROCESS
- NGX_HTTP_IMAGE_READ
- NGX_HTTP_IMAGE_RESIZE
- NGX_HTTP_IMAGE_ROTATE
- NGX_HTTP_IMAGE_SIZE
- NGX_HTTP_IMAGE_START
- NGX_HTTP_IMAGE_TEST

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12  #include <gd.h>
13
14
15  #define NGX_HTTP_IMAGE_OFF       0
16  #define NGX_HTTP_IMAGE_TEST      1
17  #define NGX_HTTP_IMAGE_SIZE      2
18  #define NGX_HTTP_IMAGE_RESIZE    3
19  #define NGX_HTTP_IMAGE_CROP      4
20  #define NGX_HTTP_IMAGE_ROTATE    5
21
22
23  #define NGX_HTTP_IMAGE_START     0
```

```
#define NGX_HTTP_IMAGE_READ      1
#define NGX_HTTP_IMAGE_PROCESS   2
#define NGX_HTTP_IMAGE_PASS      3
#define NGX_HTTP_IMAGE_DONE      4


#define NGX_HTTP_IMAGE_NONE      0
#define NGX_HTTP_IMAGE_JPEG      1
#define NGX_HTTP_IMAGE_GIF       2
#define NGX_HTTP_IMAGE_PNG       3


#define NGX_HTTP_IMAGE_BUFFERED  0x08


typedef struct {
    ngx_uint_t                   filter;
    ngx_uint_t                   width;
    ngx_uint_t                   height;
    ngx_uint_t                   angle;
    ngx_uint_t                   jpeg_quality;
    ngx_uint_t                   sharpen;

    ngx_flag_t                   transparency;
    ngx_flag_t                   interlace;

    ngx_http_complex_value_t    *wcv;
    ngx_http_complex_value_t    *hcv;
    ngx_http_complex_value_t    *acv;
    ngx_http_complex_value_t    *jqcv;
    ngx_http_complex_value_t    *shcv;

    size_t                       buffer_size;
} ngx_http_image_filter_conf_t;


typedef struct {
    u_char                      *image;
    u_char                      *last;

    size_t                       length;

    ngx_uint_t                   width;
    ngx_uint_t                   height;
    ngx_uint_t                   max_width;
    ngx_uint_t                   max_height;
    ngx_uint_t                   angle;

    ngx_uint_t                   phase;
    ngx_uint_t                   type;
    ngx_uint_t                   force;
} ngx_http_image_filter_ctx_t;


static ngx_int_t ngx_http_image_send(ngx_http_request_t *r,
    ngx_http_image_filter_ctx_t *ctx, ngx_chain_t *in);
static ngx_uint_t ngx_http_image_test(ngx_http_request_t *r, ngx_chain_t *in);
static ngx_int_t ngx_http_image_read(ngx_http_request_t *r, ngx_chain_t *in);
static ngx_buf_t *ngx_http_image_process(ngx_http_request_t *r);
static ngx_buf_t *ngx_http_image_json(ngx_http_request_t *r,
    ngx_http_image_filter_ctx_t *ctx);
static ngx_buf_t *ngx_http_image_asis(ngx_http_request_t *r,
    ngx_http_image_filter_ctx_t *ctx);
static void ngx_http_image_length(ngx_http_request_t *r, ngx_buf_t *b);
static ngx_int_t ngx_http_image_size(ngx_http_request_t *r,
    ngx_http_image_filter_ctx_t *ctx);

static ngx_buf_t *ngx_http_image_resize(ngx_http_request_t *r,
    ngx_http_image_filter_ctx_t *ctx);
static gdImagePtr ngx_http_image_source(ngx_http_request_t *r,
    ngx_http_image_filter_ctx_t *ctx);
static gdImagePtr ngx_http_image_new(ngx_http_request_t *r, int w, int h,
    int colors);
static u_char *ngx_http_image_out(ngx_http_request_t *r, ngx_uint_t type,
    gdImagePtr img, int *size);
static void ngx_http_image_cleanup(void *data);
```

```
static ngx_uint_t ngx_http_image_filter_get_value(ngx_http_request_t *r,
    ngx_http_complex_value_t *cv, ngx_uint_t v);
static ngx_uint_t ngx_http_image_filter_value(ngx_str_t *value);


static void *ngx_http_image_filter_create_conf(ngx_conf_t *cf);
static char *ngx_http_image_filter_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);
static char *ngx_http_image_filter(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_image_filter_jpeg_quality(ngx_conf_t *cf,
    ngx_command_t *cmd, void *conf);
static char *ngx_http_image_filter_sharpen(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static ngx_int_t ngx_http_image_filter_init(ngx_conf_t *cf);


static ngx_command_t  ngx_http_image_filter_commands[] = {

    { ngx_string("image_filter"),
      NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
      ngx_http_image_filter,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("image_filter_jpeg_quality"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_image_filter_jpeg_quality,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("image_filter_sharpen"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_image_filter_sharpen,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("image_filter_transparency"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_image_filter_conf_t, transparency),
      NULL },

   { ngx_string("image_filter_interlace"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_image_filter_conf_t, interlace),
      NULL },

    { ngx_string("image_filter_buffer"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_image_filter_conf_t, buffer_size),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_image_filter_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_image_filter_init,            /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_image_filter_create_conf,     /* create location configuration */
```

```
    ngx_http_image_filter_merge_conf       /* merge location configuration */
};


ngx_module_t  ngx_http_image_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_image_filter_module_ctx,     /* module context */
    ngx_http_image_filter_commands,        /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static ngx_str_t  ngx_http_image_types[] = {
    ngx_string("image/jpeg"),
    ngx_string("image/gif"),
    ngx_string("image/png")
};


static ngx_int_t
ngx_http_image_header_filter(ngx_http_request_t *r)
{
    off_t                          len;
    ngx_http_image_filter_ctx_t   *ctx;
    ngx_http_image_filter_conf_t  *conf;

    if (r->headers_out.status == NGX_HTTP_NOT_MODIFIED) {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_http_get_module_ctx(r, ngx_http_image_filter_module);

    if (ctx) {
        ngx_http_set_ctx(r, NULL, ngx_http_image_filter_module);
        return ngx_http_next_header_filter(r);
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_image_filter_module);

    if (conf->filter == NGX_HTTP_IMAGE_OFF) {
        return ngx_http_next_header_filter(r);
    }

    if (r->headers_out.content_type.len
            >= sizeof("multipart/x-mixed-replace") - 1
        && ngx_strncasecmp(r->headers_out.content_type.data,
                           (u_char *) "multipart/x-mixed-replace",
                           sizeof("multipart/x-mixed-replace") - 1)
           == 0)
    {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "image filter: multipart/x-mixed-replace response");

        return NGX_ERROR;
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_image_filter_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_image_filter_module);

    len = r->headers_out.content_length_n;
```

```
252
253     if (len != -1 && len > (off_t) conf->buffer_size) {
254         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
255                       "image filter: too big response: %O", len);
256
257         return NGX_HTTP_UNSUPPORTED_MEDIA_TYPE;
258     }
259
260     if (len == -1) {
261         ctx->length = conf->buffer_size;
262
263     } else {
264         ctx->length = (size_t) len;
265     }
266
267     if (r->headers_out.refresh) {
268         r->headers_out.refresh->hash = 0;
269     }
270
271     r->main_filter_need_in_memory = 1;
272     r->allow_ranges = 0;
273
274     return NGX_OK;
275 }
276
277
278 static ngx_int_t
279 ngx_http_image_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
280 {
281     ngx_int_t                    rc;
282     ngx_str_t                   *ct;
283     ngx_chain_t                  out;
284     ngx_http_image_filter_ctx_t  *ctx;
285     ngx_http_image_filter_conf_t  *conf;
286
287     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0, "image filter");
288
289     if (in == NULL) {
290         return ngx_http_next_body_filter(r, in);
291     }
292
293     ctx = ngx_http_get_module_ctx(r, ngx_http_image_filter_module);
294
295     if (ctx == NULL) {
296         return ngx_http_next_body_filter(r, in);
297     }
298
299     switch (ctx->phase) {
300
301     case NGX_HTTP_IMAGE_START:
302
303         ctx->type = ngx_http_image_test(r, in);
304
305         conf = ngx_http_get_module_loc_conf(r, ngx_http_image_filter_module);
306
307         if (ctx->type == NGX_HTTP_IMAGE_NONE) {
308
309             if (conf->filter == NGX_HTTP_IMAGE_SIZE) {
310                 out.buf = ngx_http_image_json(r, NULL);
311
312                 if (out.buf) {
313                     out.next = NULL;
314                     ctx->phase = NGX_HTTP_IMAGE_DONE;
315
316                     return ngx_http_image_send(r, ctx, &out);
317                 }
318             }
319
320             return ngx_http_filter_finalize_request(r,
321                                                 &ngx_http_image_filter_module,
322                                                 NGX_HTTP_UNSUPPORTED_MEDIA_TYPE);
323         }
324
325         /* override content type */
326
327         ct = &ngx_http_image_types[ctx->type - 1];
```

```
328         r->headers_out.content_type_len = ct->len;
329         r->headers_out.content_type = *ct;
330         r->headers_out.content_type_lowcase = NULL;
331 
332         if (conf->filter == NGX_HTTP_IMAGE_TEST) {
333             ctx->phase = NGX_HTTP_IMAGE_PASS;
334 
335             return ngx_http_image_send(r, ctx, in);
336         }
337 
338         ctx->phase = NGX_HTTP_IMAGE_READ;
339 
340         /* fall through */
341 
342     case NGX_HTTP_IMAGE_READ:
343 
344         rc = ngx_http_image_read(r, in);
345 
346         if (rc == NGX_AGAIN) {
347             return NGX_OK;
348         }
349 
350         if (rc == NGX_ERROR) {
351             return ngx_http_filter_finalize_request(r,
352                                                     &ngx_http_image_filter_module,
353                                                     NGX_HTTP_UNSUPPORTED_MEDIA_TYPE);
354         }
355 
356         /* fall through */
357 
358     case NGX_HTTP_IMAGE_PROCESS:
359 
360         out.buf = ngx_http_image_process(r);
361 
362         if (out.buf == NULL) {
363             return ngx_http_filter_finalize_request(r,
364                                                     &ngx_http_image_filter_module,
365                                                     NGX_HTTP_UNSUPPORTED_MEDIA_TYPE);
366         }
367 
368         out.next = NULL;
369         ctx->phase = NGX_HTTP_IMAGE_PASS;
370 
371         return ngx_http_image_send(r, ctx, &out);
372 
373     case NGX_HTTP_IMAGE_PASS:
374 
375         return ngx_http_next_body_filter(r, in);
376 
377     default: /* NGX_HTTP_IMAGE_DONE */
378 
379         rc = ngx_http_next_body_filter(r, NULL);
380 
381         /* NGX_ERROR resets any pending data */
382         return (rc == NGX_OK) ? NGX_ERROR : rc;
383     }
384 }
385 
386 
387 static ngx_int_t
388 ngx_http_image_send(ngx_http_request_t *r, ngx_http_image_filter_ctx_t *ctx,
389     ngx_chain_t *in)
390 {
391     ngx_int_t  rc;
392 
393     rc = ngx_http_next_header_filter(r);
394 
395     if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
396         return NGX_ERROR;
397     }
398 
399     rc = ngx_http_next_body_filter(r, in);
400 
401     if (ctx->phase == NGX_HTTP_IMAGE_DONE) {
402         /* NGX_ERROR resets any pending data */
403         return (rc == NGX_OK) ? NGX_ERROR : rc;
```

```
    }

    return rc;
}


static ngx_uint_t
ngx_http_image_test(ngx_http_request_t *r, ngx_chain_t *in)
{
    u_char  *p;

    p = in->buf->pos;

    if (in->buf->last - p < 16) {
        return NGX_HTTP_IMAGE_NONE;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "image filter: \"%c%c\"", p[0], p[1]);

    if (p[0] == 0xff && p[1] == 0xd8) {

        /* JPEG */

        return NGX_HTTP_IMAGE_JPEG;

    } else if (p[0] == 'G' && p[1] == 'I' && p[2] == 'F' && p[3] == '8'
               && p[5] == 'a')
    {
        if (p[4] == '9' || p[4] == '7') {
            /* GIF */
            return NGX_HTTP_IMAGE_GIF;
        }

    } else if (p[0] == 0x89 && p[1] == 'P' && p[2] == 'N' && p[3] == 'G'
               && p[4] == 0x0d && p[5] == 0x0a && p[6] == 0x1a && p[7] == 0x0a)
    {
        /* PNG */

        return NGX_HTTP_IMAGE_PNG;
    }

    return NGX_HTTP_IMAGE_NONE;
}


static ngx_int_t
ngx_http_image_read(ngx_http_request_t *r, ngx_chain_t *in)
{
    u_char                       *p;
    size_t                        size, rest;
    ngx_buf_t                    *b;
    ngx_chain_t                  *cl;
    ngx_http_image_filter_ctx_t  *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_image_filter_module);

    if (ctx->image == NULL) {
        ctx->image = ngx_palloc(r->pool, ctx->length);
        if (ctx->image == NULL) {
            return NGX_ERROR;
        }

        ctx->last = ctx->image;
    }

    p = ctx->last;

    for (cl = in; cl; cl = cl->next) {

        b = cl->buf;
        size = b->last - b->pos;

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "image buf: %uz", size);

```

```
        rest = ctx->image + ctx->length - p;

        if (size > rest) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "image filter: too big response");
            return NGX_ERROR;
        }

        p = ngx_cpymem(p, b->pos, size);
        b->pos += size;

        if (b->last_buf) {
            ctx->last = p;
            return NGX_OK;
        }
    }

    ctx->last = p;
    r->connection->buffered |= NGX_HTTP_IMAGE_BUFFERED;

    return NGX_AGAIN;
}


static ngx_buf_t *
ngx_http_image_process(ngx_http_request_t *r)
{
    ngx_int_t                      rc;
    ngx_http_image_filter_ctx_t   *ctx;
    ngx_http_image_filter_conf_t  *conf;

    r->connection->buffered &= ~NGX_HTTP_IMAGE_BUFFERED;

    ctx = ngx_http_get_module_ctx(r, ngx_http_image_filter_module);

    rc = ngx_http_image_size(r, ctx);

    conf = ngx_http_get_module_loc_conf(r, ngx_http_image_filter_module);

    if (conf->filter == NGX_HTTP_IMAGE_SIZE) {
        return ngx_http_image_json(r, rc == NGX_OK ? ctx : NULL);
    }

    ctx->angle = ngx_http_image_filter_get_value(r, conf->acv, conf->angle);

    if (conf->filter == NGX_HTTP_IMAGE_ROTATE) {

        if (ctx->angle != 90 && ctx->angle != 180 && ctx->angle != 270) {
            return NULL;
        }

        return ngx_http_image_resize(r, ctx);
    }

    ctx->max_width = ngx_http_image_filter_get_value(r, conf->wcv, conf->width);
    if (ctx->max_width == 0) {
        return NULL;
    }

    ctx->max_height = ngx_http_image_filter_get_value(r, conf->hcv,
                                                      conf->height);
    if (ctx->max_height == 0) {
        return NULL;
    }

    if (rc == NGX_OK
        && ctx->width <= ctx->max_width
        && ctx->height <= ctx->max_height
        && ctx->angle == 0
        && !ctx->force)
    {
        return ngx_http_image_asis(r, ctx);
    }

    return ngx_http_image_resize(r, ctx);
}
```

```
556
557
558 static ngx_buf_t *
559 ngx_http_image_json(ngx_http_request_t *r, ngx_http_image_filter_ctx_t *ctx)
560 {
561     size_t      len;
562     ngx_buf_t  *b;
563
564     b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
565     if (b == NULL) {
566         return NULL;
567     }
568
569     b->memory = 1;
570     b->last_buf = 1;
571
572     ngx_http_clean_header(r);
573
574     r->headers_out.status = NGX_HTTP_OK;
575     r->headers_out.content_type_len = sizeof("application/json") - 1;
576     ngx_str_set(&r->headers_out.content_type, "application/json");
577     r->headers_out.content_type_lowcase = NULL;
578
579     if (ctx == NULL) {
580         b->pos = (u_char *) "{}" CRLF;
581         b->last = b->pos + sizeof("{}" CRLF) - 1;
582
583         ngx_http_image_length(r, b);
584
585         return b;
586     }
587
588     len = sizeof("{ \"img\" : "
589                  "{ \"width\": , \"height\": , \"type\": \"jpeg\" } }" CRLF) - 1
590           + 2 * NGX_SIZE_T_LEN;
591
592     b->pos = ngx_pnalloc(r->pool, len);
593     if (b->pos == NULL) {
594         return NULL;
595     }
596
597     b->last = ngx_sprintf(b->pos,
598                           "{ \"img\" : "
599                                       "{ \"width\": %uz,"
600                                        " \"height\": %uz,"
601                                        " \"type\": \"%s\" } }" CRLF,
602                           ctx->width, ctx->height,
603                           ngx_http_image_types[ctx->type - 1].data + 6);
604
605     ngx_http_image_length(r, b);
606
607     return b;
608 }
609
610
611 static ngx_buf_t *
612 ngx_http_image_asis(ngx_http_request_t *r, ngx_http_image_filter_ctx_t *ctx)
613 {
614     ngx_buf_t  *b;
615
616     b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
617     if (b == NULL) {
618         return NULL;
619     }
620
621     b->pos = ctx->image;
622     b->last = ctx->last;
623     b->memory = 1;
624     b->last_buf = 1;
625
626     ngx_http_image_length(r, b);
627
628     return b;
629 }
630
631
```

```
632 static void
633 ngx_http_image_length(ngx_http_request_t *r, ngx_buf_t *b)
634 {
635     r->headers_out.content_length_n = b->last - b->pos;
636
637     if (r->headers_out.content_length) {
638         r->headers_out.content_length->hash = 0;
639     }
640
641     r->headers_out.content_length = NULL;
642 }
643
644
645 static ngx_int_t
646 ngx_http_image_size(ngx_http_request_t *r, ngx_http_image_filter_ctx_t *ctx)
647 {
648     u_char      *p, *last;
649     size_t       len, app;
650     ngx_uint_t   width, height;
651
652     p = ctx->image;
653
654     switch (ctx->type) {
655
656     case NGX_HTTP_IMAGE_JPEG:
657
658         p += 2;
659         last = ctx->image + ctx->length - 10;
660         width = 0;
661         height = 0;
662         app = 0;
663
664         while (p < last) {
665
666             if (p[0] == 0xff && p[1] != 0xff) {
667
668                 ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
669                                "JPEG: %02xd %02xd", p[0], p[1]);
670
671                 p++;
672
673                 if ((*p == 0xc0 || *p == 0xc1 || *p == 0xc2 || *p == 0xc3
674                      || *p == 0xc9 || *p == 0xca || *p == 0xcb)
675                     && (width == 0 || height == 0))
676                 {
677                     width = p[6] * 256 + p[7];
678                     height = p[4] * 256 + p[5];
679                 }
680
681                 ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
682                                "JPEG: %02xd %02xd", p[1], p[2]);
683
684                 len = p[1] * 256 + p[2];
685
686                 if (*p >= 0xe1 && *p <= 0xef) {
687                     /* application data, e.g., EXIF, Adobe XMP, etc. */
688                     app += len;
689                 }
690
691                 p += len;
692
693                 continue;
694             }
695
696             p++;
697         }
698
699         if (width == 0 || height == 0) {
700             return NGX_DECLINED;
701         }
702
703         if (ctx->length / 20 < app) {
704             /* force conversion if application data consume more than 5% */
705             ctx->force = 1;
706             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
707                            "app data size: %uz", app);
```

```
        }

        break;

    case NGX_HTTP_IMAGE_GIF:

        if (ctx->length < 10) {
            return NGX_DECLINED;
        }

        width = p[7] * 256 + p[6];
        height = p[9] * 256 + p[8];

        break;

    case NGX_HTTP_IMAGE_PNG:

        if (ctx->length < 24) {
            return NGX_DECLINED;
        }

        width = p[18] * 256 + p[19];
        height = p[22] * 256 + p[23];

        break;

    default:

        return NGX_DECLINED;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "image size: %d x %d", width, height);

    ctx->width = width;
    ctx->height = height;

    return NGX_OK;
}


static ngx_buf_t *
ngx_http_image_resize(ngx_http_request_t *r, ngx_http_image_filter_ctx_t *ctx)
{
    int                            sx, sy, dx, dy, ox, oy, ax, ay, size,
                                   colors, palette, transparent, sharpen,
                                   red, green, blue, t;
    u_char                        *out;
    ngx_buf_t                     *b;
    ngx_uint_t                     resize;
    gdImagePtr                     src, dst;
    ngx_pool_cleanup_t            *cln;
    ngx_http_image_filter_conf_t  *conf;

    src = ngx_http_image_source(r, ctx);

    if (src == NULL) {
        return NULL;
    }

    sx = gdImageSX(src);
    sy = gdImageSY(src);

    conf = ngx_http_get_module_loc_conf(r, ngx_http_image_filter_module);

    if (!ctx->force
        && ctx->angle == 0
        && (ngx_uint_t) sx <= ctx->max_width
        && (ngx_uint_t) sy <= ctx->max_height)
    {
        gdImageDestroy(src);
        return ngx_http_image_asis(r, ctx);
    }

    colors = gdImageColorsTotal(src);

```

```
    if (colors && conf->transparency) {
        transparent = gdImageGetTransparent(src);

        if (transparent != -1) {
            palette = colors;
            red = gdImageRed(src, transparent);
            green = gdImageGreen(src, transparent);
            blue = gdImageBlue(src, transparent);

            goto transparent;
        }
    }

    palette = 0;
    transparent = -1;
    red = 0;
    green = 0;
    blue = 0;

transparent:

    gdImageColorTransparent(src, -1);

    dx = sx;
    dy = sy;

    if (conf->filter == NGX_HTTP_IMAGE_RESIZE) {

        if ((ngx_uint_t) dx > ctx->max_width) {
            dy = dy * ctx->max_width / dx;
            dy = dy ? dy : 1;
            dx = ctx->max_width;
        }

        if ((ngx_uint_t) dy > ctx->max_height) {
            dx = dx * ctx->max_height / dy;
            dx = dx ? dx : 1;
            dy = ctx->max_height;
        }

        resize = 1;

    } else if (conf->filter == NGX_HTTP_IMAGE_ROTATE) {

        resize = 0;

    } else { /* NGX_HTTP_IMAGE_CROP */

        resize = 0;

        if ((double) dx / dy < (double) ctx->max_width / ctx->max_height) {
            if ((ngx_uint_t) dx > ctx->max_width) {
                dy = dy * ctx->max_width / dx;
                dy = dy ? dy : 1;
                dx = ctx->max_width;
                resize = 1;
            }

        } else {
            if ((ngx_uint_t) dy > ctx->max_height) {
                dx = dx * ctx->max_height / dy;
                dx = dx ? dx : 1;
                dy = ctx->max_height;
                resize = 1;
            }
        }
    }

    if (resize) {
        dst = ngx_http_image_new(r, dx, dy, palette);
        if (dst == NULL) {
            gdImageDestroy(src);
            return NULL;
        }

        if (colors == 0) {
```

```
            gdImageSaveAlpha(dst, 1);
            gdImageAlphaBlending(dst, 0);
        }

        gdImageCopyResampled(dst, src, 0, 0, 0, 0, dx, dy, sx, sy);

        if (colors) {
            gdImageTrueColorToPalette(dst, 1, 256);
        }

        gdImageDestroy(src);

    } else {
        dst = src;
    }

    if (ctx->angle) {
        src = dst;

        ax = (dx % 2 == 0) ? 1 : 0;
        ay = (dy % 2 == 0) ? 1 : 0;

        switch (ctx->angle) {

        case 90:
        case 270:
            dst = ngx_http_image_new(r, dy, dx, palette);
            if (dst == NULL) {
                gdImageDestroy(src);
                return NULL;
            }
            if (ctx->angle == 90) {
                ox = dy / 2 + ay;
                oy = dx / 2 - ax;

            } else {
                ox = dy / 2 - ay;
                oy = dx / 2 + ax;
            }

            gdImageCopyRotated(dst, src, ox, oy, 0, 0,
                               dx + ax, dy + ay, ctx->angle);
            gdImageDestroy(src);

            t = dx;
            dx = dy;
            dy = t;
            break;

        case 180:
            dst = ngx_http_image_new(r, dx, dy, palette);
            if (dst == NULL) {
                gdImageDestroy(src);
                return NULL;
            }
            gdImageCopyRotated(dst, src, dx / 2 - ax, dy / 2 - ay, 0, 0,
                               dx + ax, dy + ay, ctx->angle);
            gdImageDestroy(src);
            break;
        }
    }

    if (conf->filter == NGX_HTTP_IMAGE_CROP) {

        src = dst;

        if ((ngx_uint_t) dx > ctx->max_width) {
            ox = dx - ctx->max_width;

        } else {
            ox = 0;
        }

        if ((ngx_uint_t) dy > ctx->max_height) {
            oy = dy - ctx->max_height;

```

```
        } else {
            oy = 0;
        }

        if (ox || oy) {

            dst = ngx_http_image_new(r, dx - ox, dy - oy, colors);

            if (dst == NULL) {
                gdImageDestroy(src);
                return NULL;
            }

            ox /= 2;
            oy /= 2;

            ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "image crop: %d x %d @ %d x %d",
                           dx, dy, ox, oy);

            if (colors == 0) {
                gdImageSaveAlpha(dst, 1);
                gdImageAlphaBlending(dst, 0);
            }

            gdImageCopy(dst, src, 0, 0, ox, oy, dx - ox, dy - oy);

            if (colors) {
                gdImageTrueColorToPalette(dst, 1, 256);
            }

            gdImageDestroy(src);
        }
    }

    if (transparent != -1 && colors) {
        gdImageColorTransparent(dst, gdImageColorExact(dst, red, green, blue));
    }

    sharpen = ngx_http_image_filter_get_value(r, conf->shcv, conf->sharpen);
    if (sharpen > 0) {
        gdImageSharpen(dst, sharpen);
    }

    gdImageInterlace(dst, (int) conf->interlace);

    out = ngx_http_image_out(r, ctx->type, dst, &size);

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "image: %d x %d %d", sx, sy, colors);

    gdImageDestroy(dst);
    ngx_pfree(r->pool, ctx->image);

    if (out == NULL) {
        return NULL;
    }

    cln = ngx_pool_cleanup_add(r->pool, 0);
    if (cln == NULL) {
        gdFree(out);
        return NULL;
    }

     b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
     if (b == NULL) {
         gdFree(out);
         return NULL;
     }

     cln->handler = ngx_http_image_cleanup;
     cln->data = out;

     b->pos = out;
     b->last = out + size;
     b->memory = 1;
```

```
1012      b->last_buf = 1;
1013
1014      ngx_http_image_length(r, b);
1015      ngx_http_weak_etag(r);
1016
1017      return b;
1018  }
1019
1020
1021  static gdImagePtr
1022  ngx_http_image_source(ngx_http_request_t *r, ngx_http_image_filter_ctx_t *ctx)
1023  {
1024      char        *failed;
1025      gdImagePtr   img;
1026
1027      img = NULL;
1028
1029      switch (ctx->type) {
1030
1031      case NGX_HTTP_IMAGE_JPEG:
1032          img = gdImageCreateFromJpegPtr(ctx->length, ctx->image);
1033          failed = "gdImageCreateFromJpegPtr() failed";
1034          break;
1035
1036      case NGX_HTTP_IMAGE_GIF:
1037          img = gdImageCreateFromGifPtr(ctx->length, ctx->image);
1038          failed = "gdImageCreateFromGifPtr() failed";
1039          break;
1040
1041      case NGX_HTTP_IMAGE_PNG:
1042          img = gdImageCreateFromPngPtr(ctx->length, ctx->image);
1043          failed = "gdImageCreateFromPngPtr() failed";
1044          break;
1045
1046      default:
1047          failed = "unknown image type";
1048          break;
1049      }
1050
1051      if (img == NULL) {
1052          ngx_log_error(NGX_LOG_ERR, r->connection->log, 0, failed);
1053      }
1054
1055      return img;
1056  }
1057
1058
1059  static gdImagePtr
1060  ngx_http_image_new(ngx_http_request_t *r, int w, int h, int colors)
1061  {
1062      gdImagePtr  img;
1063
1064      if (colors == 0) {
1065          img = gdImageCreateTrueColor(w, h);
1066
1067          if (img == NULL) {
1068              ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1069                            "gdImageCreateTrueColor() failed");
1070              return NULL;
1071          }
1072
1073      } else {
1074          img = gdImageCreate(w, h);
1075
1076          if (img == NULL) {
1077              ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1078                            "gdImageCreate() failed");
1079              return NULL;
1080          }
1081      }
1082
1083      return img;
1084  }
1085
1086
1087  static u_char *
```

```
1088 ngx_http_image_out(ngx_http_request_t *r, ngx_uint_t type, gdImagePtr img,
1089     int *size)
1090 {
1091     char                          *failed;
1092     u_char                        *out;
1093     ngx_int_t                      jq;
1094     ngx_http_image_filter_conf_t  *conf;
1095
1096     out = NULL;
1097
1098     switch (type) {
1099
1100     case NGX_HTTP_IMAGE_JPEG:
1101         conf = ngx_http_get_module_loc_conf(r, ngx_http_image_filter_module);
1102
1103         jq = ngx_http_image_filter_get_value(r, conf->jqcv, conf->jpeg_quality);
1104         if (jq <= 0) {
1105             return NULL;
1106         }
1107
1108         out = gdImageJpegPtr(img, size, jq);
1109         failed = "gdImageJpegPtr() failed";
1110         break;
1111
1112     case NGX_HTTP_IMAGE_GIF:
1113         out = gdImageGifPtr(img, size);
1114         failed = "gdImageGifPtr() failed";
1115         break;
1116
1117     case NGX_HTTP_IMAGE_PNG:
1118         out = gdImagePngPtr(img, size);
1119         failed = "gdImagePngPtr() failed";
1120         break;
1121
1122     default:
1123         failed = "unknown image type";
1124         break;
1125     }
1126
1127     if (out == NULL) {
1128         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0, failed);
1129     }
1130
1131     return out;
1132 }
1133
1134
1135 static void
1136 ngx_http_image_cleanup(void *data)
1137 {
1138     gdFree(data);
1139 }
1140
1141
1142 static ngx_uint_t
1143 ngx_http_image_filter_get_value(ngx_http_request_t *r,
1144     ngx_http_complex_value_t *cv, ngx_uint_t v)
1145 {
1146     ngx_str_t  val;
1147
1148     if (cv == NULL) {
1149         return v;
1150     }
1151
1152     if (ngx_http_complex_value(r, cv, &val) != NGX_OK) {
1153         return 0;
1154     }
1155
1156     return ngx_http_image_filter_value(&val);
1157 }
1158
1159
1160 static ngx_uint_t
1161 ngx_http_image_filter_value(ngx_str_t *value)
1162 {
1163     ngx_int_t  n;
```

```
1164
1165    if (value->len == 1 && value->data[0] == '-') {
1166        return (ngx_uint_t) -1;
1167    }
1168
1169    n = ngx_atoi(value->data, value->len);
1170
1171    if (n > 0) {
1172        return (ngx_uint_t) n;
1173    }
1174
1175    return 0;
1176 }
1177
1178
1179 static void *
1180 ngx_http_image_filter_create_conf(ngx_conf_t *cf)
1181 {
1182     ngx_http_image_filter_conf_t  *conf;
1183
1184     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_image_filter_conf_t));
1185     if (conf == NULL) {
1186         return NULL;
1187     }
1188
1189     /*
1190      * set by ngx_pcalloc():
1191      *
1192      *     conf->width = 0;
1193      *     conf->height = 0;
1194      *     conf->angle = 0;
1195      *     conf->wcv = NULL;
1196      *     conf->hcv = NULL;
1197      *     conf->acv = NULL;
1198      *     conf->jqcv = NULL;
1199      *     conf->shcv = NULL;
1200      */
1201
1202     conf->filter = NGX_CONF_UNSET_UINT;
1203     conf->jpeg_quality = NGX_CONF_UNSET_UINT;
1204     conf->sharpen = NGX_CONF_UNSET_UINT;
1205     conf->transparency = NGX_CONF_UNSET;
1206     conf->interlace = NGX_CONF_UNSET;
1207     conf->buffer_size = NGX_CONF_UNSET_SIZE;
1208
1209     return conf;
1210 }
1211
1212
1213 static char *
1214 ngx_http_image_filter_merge_conf(ngx_conf_t *cf, void *parent, void *child)
1215 {
1216     ngx_http_image_filter_conf_t *prev = parent;
1217     ngx_http_image_filter_conf_t *conf = child;
1218
1219     if (conf->filter == NGX_CONF_UNSET_UINT) {
1220
1221         if (prev->filter == NGX_CONF_UNSET_UINT) {
1222             conf->filter = NGX_HTTP_IMAGE_OFF;
1223
1224         } else {
1225             conf->filter = prev->filter;
1226             conf->width = prev->width;
1227             conf->height = prev->height;
1228             conf->angle = prev->angle;
1229             conf->wcv = prev->wcv;
1230             conf->hcv = prev->hcv;
1231             conf->acv = prev->acv;
1232         }
1233     }
1234
1235     if (conf->jpeg_quality == NGX_CONF_UNSET_UINT) {
1236
1237         /* 75 is libjpeg default quality */
1238         ngx_conf_merge_uint_value(conf->jpeg_quality, prev->jpeg_quality, 75);
1239
```

```
        if (conf->jqcv == NULL) {
            conf->jqcv = prev->jqcv;
        }
    }

    if (conf->sharpen == NGX_CONF_UNSET_UINT) {
        ngx_conf_merge_uint_value(conf->sharpen, prev->sharpen, 0);

        if (conf->shcv == NULL) {
            conf->shcv = prev->shcv;
        }
    }

    ngx_conf_merge_value(conf->transparency, prev->transparency, 1);

    ngx_conf_merge_value(conf->interlace, prev->interlace, 0);

    ngx_conf_merge_size_value(conf->buffer_size, prev->buffer_size,
                              1 * 1024 * 1024);

    return NGX_CONF_OK;
}


static char *
ngx_http_image_filter(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_image_filter_conf_t *imcf = conf;

    ngx_str_t                          *value;
    ngx_int_t                           n;
    ngx_uint_t                          i;
    ngx_http_complex_value_t            cv;
    ngx_http_compile_complex_value_t    ccv;

    value = cf->args->elts;

    i = 1;

    if (cf->args->nelts == 2) {
        if (ngx_strcmp(value[i].data, "off") == 0) {
            imcf->filter = NGX_HTTP_IMAGE_OFF;

        } else if (ngx_strcmp(value[i].data, "test") == 0) {
            imcf->filter = NGX_HTTP_IMAGE_TEST;

        } else if (ngx_strcmp(value[i].data, "size") == 0) {
            imcf->filter = NGX_HTTP_IMAGE_SIZE;

        } else {
            goto failed;
        }

        return NGX_CONF_OK;

    } else if (cf->args->nelts == 3) {

        if (ngx_strcmp(value[i].data, "rotate") == 0) {
            if (imcf->filter != NGX_HTTP_IMAGE_RESIZE
                && imcf->filter != NGX_HTTP_IMAGE_CROP)
            {
                imcf->filter = NGX_HTTP_IMAGE_ROTATE;
            }

            ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

            ccv.cf = cf;
            ccv.value = &value[++i];
            ccv.complex_value = &cv;

            if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

            if (cv.lengths == NULL) {
                n = ngx_http_image_filter_value(&value[i]);
```

```
            if (n != 90 && n != 180 && n != 270) {
                goto failed;
            }

            imcf->angle = (ngx_uint_t) n;

        } else {
            imcf->acv = ngx_palloc(cf->pool,
                                   sizeof(ngx_http_complex_value_t));
            if (imcf->acv == NULL) {
                return NGX_CONF_ERROR;
            }

            *imcf->acv = cv;
        }

        return NGX_CONF_OK;

    } else {
        goto failed;
    }
}

if (ngx_strcmp(value[i].data, "resize") == 0) {
    imcf->filter = NGX_HTTP_IMAGE_RESIZE;

} else if (ngx_strcmp(value[i].data, "crop") == 0) {
    imcf->filter = NGX_HTTP_IMAGE_CROP;

} else {
    goto failed;
}

ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

ccv.cf = cf;
ccv.value = &value[++i];
ccv.complex_value = &cv;

if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
    return NGX_CONF_ERROR;
}

if (cv.lengths == NULL) {
    n = ngx_http_image_filter_value(&value[i]);

    if (n == 0) {
        goto failed;
    }

    imcf->width = (ngx_uint_t) n;

} else {
    imcf->wcv = ngx_palloc(cf->pool, sizeof(ngx_http_complex_value_t));
    if (imcf->wcv == NULL) {
        return NGX_CONF_ERROR;
    }

    *imcf->wcv = cv;
}

ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

ccv.cf = cf;
ccv.value = &value[++i];
ccv.complex_value = &cv;

if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
    return NGX_CONF_ERROR;
}

if (cv.lengths == NULL) {
    n = ngx_http_image_filter_value(&value[i]);

    if (n == 0) {
```

```
            goto failed;
        }

        imcf->height = (ngx_uint_t) n;

    } else {
        imcf->hcv = ngx_palloc(cf->pool, sizeof(ngx_http_complex_value_t));
        if (imcf->hcv == NULL) {
            return NGX_CONF_ERROR;
        }

        *imcf->hcv = cv;
    }

    return NGX_CONF_OK;

failed:

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid parameter \"%V\"",
                       &value[i]);

    return NGX_CONF_ERROR;
}


static char *
ngx_http_image_filter_jpeg_quality(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf)
{
    ngx_http_image_filter_conf_t *imcf = conf;

    ngx_str_t                         *value;
    ngx_int_t                          n;
    ngx_http_complex_value_t           cv;
    ngx_http_compile_complex_value_t   ccv;

    value = cf->args->elts;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &cv;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (cv.lengths == NULL) {
        n = ngx_http_image_filter_value(&value[1]);

        if (n <= 0) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid value \"%V\"", &value[1]);
            return NGX_CONF_ERROR;
        }

        imcf->jpeg_quality = (ngx_uint_t) n;

    } else {
        imcf->jqcv = ngx_palloc(cf->pool, sizeof(ngx_http_complex_value_t));
        if (imcf->jqcv == NULL) {
            return NGX_CONF_ERROR;
        }

        *imcf->jqcv = cv;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_image_filter_sharpen(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf)
{
```

```
1468     ngx_http_image_filter_conf_t *imcf = conf;
1469
1470     ngx_str_t                         *value;
1471     ngx_int_t                          n;
1472     ngx_http_complex_value_t           cv;
1473     ngx_http_compile_complex_value_t   ccv;
1474
1475     value = cf->args->elts;
1476
1477     ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
1478
1479     ccv.cf = cf;
1480     ccv.value = &value[1];
1481     ccv.complex_value = &cv;
1482
1483     if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
1484         return NGX_CONF_ERROR;
1485     }
1486
1487     if (cv.lengths == NULL) {
1488         n = ngx_http_image_filter_value(&value[1]);
1489
1490         if (n < 0) {
1491             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1492                                "invalid value \"%V\"", &value[1]);
1493             return NGX_CONF_ERROR;
1494         }
1495
1496         imcf->sharpen = (ngx_uint_t) n;
1497
1498     } else {
1499         imcf->shcv = ngx_palloc(cf->pool, sizeof(ngx_http_complex_value_t));
1500         if (imcf->shcv == NULL) {
1501             return NGX_CONF_ERROR;
1502         }
1503
1504         *imcf->shcv = cv;
1505     }
1506
1507     return NGX_CONF_OK;
1508 }
1509
1510
1511 static ngx_int_t
1512 ngx_http_image_filter_init(ngx_conf_t *cf)
1513 {
1514     ngx_http_next_header_filter = ngx_http_top_header_filter;
1515     ngx_http_top_header_filter = ngx_http_image_header_filter;
1516
1517     ngx_http_next_body_filter = ngx_http_top_body_filter;
1518     ngx_http_top_body_filter = ngx_http_image_body_filter;
1519
1520     return NGX_OK;
1521 }
```

One Level Up Top Level

src/http/modules/ngx_http_index_module.c - nginx-1.7.10

Global variables defined

- ngx_http_index_commands
- ngx_http_index_module
- ngx_http_index_module_ctx

Data types defined

- ngx_http_index_loc_conf_t
- ngx_http_index_t

Functions defined

- ngx_http_index_create_loc_conf
- ngx_http_index_error
- ngx_http_index_handler
- ngx_http_index_init
- ngx_http_index_merge_loc_conf
- ngx_http_index_set_index
- ngx_http_index_test_dir

Macros defined

- NGX_HTTP_DEFAULT_INDEX

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_str_t                 name;
15      ngx_array_t              *lengths;
16      ngx_array_t              *values;
17  } ngx_http_index_t;
18
19
20  typedef struct {
21      ngx_array_t              *indices;    /* array of ngx_http_index_t */
22      size_t                    max_index_len;
23  } ngx_http_index_loc_conf_t;
24
```

```
25
26  #define NGX_HTTP_DEFAULT_INDEX   "index.html"
27
28
29  static ngx_int_t ngx_http_index_test_dir(ngx_http_request_t *r,
30      ngx_http_core_loc_conf_t *clcf, u_char *path, u_char *last);
31  static ngx_int_t ngx_http_index_error(ngx_http_request_t *r,
32      ngx_http_core_loc_conf_t *clcf, u_char *file, ngx_err_t err);
33
34  static ngx_int_t ngx_http_index_init(ngx_conf_t *cf);
35  static void *ngx_http_index_create_loc_conf(ngx_conf_t *cf);
36  static char *ngx_http_index_merge_loc_conf(ngx_conf_t *cf,
37      void *parent, void *child);
38  static char *ngx_http_index_set_index(ngx_conf_t *cf, ngx_command_t *cmd,
39      void *conf);
40
41
42  static ngx_command_t  ngx_http_index_commands[] = {
43
44      { ngx_string("index"),
45        NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
46        ngx_http_index_set_index,
47        NGX_HTTP_LOC_CONF_OFFSET,
48        0,
49        NULL },
50
51        ngx_null_command
52  };
53
54
55  static ngx_http_module_t  ngx_http_index_module_ctx = {
56      NULL,                                  /* preconfiguration */
57      ngx_http_index_init,                   /* postconfiguration */
58
59      NULL,                                  /* create main configuration */
60      NULL,                                  /* init main configuration */
61
62      NULL,                                  /* create server configuration */
63      NULL,                                  /* merge server configuration */
64
65      ngx_http_index_create_loc_conf,        /* create location configuration */
66      ngx_http_index_merge_loc_conf          /* merge location configuration */
67  };
68
69
70  ngx_module_t  ngx_http_index_module = {
71      NGX_MODULE_V1,
72      &ngx_http_index_module_ctx,            /* module context */
73      ngx_http_index_commands,               /* module directives */
74      NGX_HTTP_MODULE,                       /* module type */
75      NULL,                                  /* init master */
76      NULL,                                  /* init module */
77      NULL,                                  /* init process */
78      NULL,                                  /* init thread */
79      NULL,                                  /* exit thread */
80      NULL,                                  /* exit process */
81      NULL,                                  /* exit master */
82      NGX_MODULE_V1_PADDING
83  };
84
85
86  /*
87  * Try to open/test the first index file before the test of directory
88  * existence because valid requests should prevail over invalid ones.
89  * If open()/stat() of a file will fail then stat() of a directory
90  * should be faster because kernel may have already cached some data.
91  * Besides, Win32 may return ERROR_PATH_NOT_FOUND (NGX_ENOTDIR) at once.
92  * Unix has ENOTDIR error; however, it's less helpful than Win32's one:
93  * it only indicates that path points to a regular file, not a directory.
94  */
95
96  static ngx_int_t
97  ngx_http_index_handler(ngx_http_request_t *r)
98  {
99      u_char                     *p, *name;
100     size_t                      len, root, reserve, allocated;
```

```
    ngx_int_t                    rc;
    ngx_str_t                    path, uri;
    ngx_uint_t                   i, dir_tested;
    ngx_http_index_t            *index;
    ngx_open_file_info_t         of;
    ngx_http_script_code_pt      code;
    ngx_http_script_engine_t     e;
    ngx_http_core_loc_conf_t    *clcf;
    ngx_http_index_loc_conf_t   *ilcf;
    ngx_http_script_len_code_pt  lcode;

    if (r->uri.data[r->uri.len - 1] != '/') {
        return NGX_DECLINED;
    }

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD|NGX_HTTP_POST))) {
        return NGX_DECLINED;
    }

    ilcf = ngx_http_get_module_loc_conf(r, ngx_http_index_module);
    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    allocated = 0;
    root = 0;
    dir_tested = 0;
    name = NULL;
    /* suppress MSVC warning */
    path.data = NULL;

    index = ilcf->indices->elts;
    for (i = 0; i < ilcf->indices->nelts; i++) {

        if (index[i].lengths == NULL) {

            if (index[i].name.data[0] == '/') {
                return ngx_http_internal_redirect(r, &index[i].name, &r->args);
            }

            reserve = ilcf->max_index_len;
            len = index[i].name.len;

        } else {
            ngx_memzero(&e, sizeof(ngx_http_script_engine_t));

            e.ip = index[i].lengths->elts;
            e.request = r;
            e.flushed = 1;

            /* 1 is for terminating '\0' as in static names */
            len = 1;

            while (*(uintptr_t *) e.ip) {
                lcode = *(ngx_http_script_len_code_pt *) e.ip;
                len += lcode(&e);
            }

            /* 16 bytes are preallocation */

            reserve = len + 16;
        }

        if (reserve > allocated) {

            name = ngx_http_map_uri_to_path(r, &path, &root, reserve);
            if (name == NULL) {
                return NGX_ERROR;
            }

            allocated = path.data + path.len - name;
        }

        if (index[i].values == NULL) {

            /* index[i].name.len includes the terminating '\0' */

            ngx_memcpy(name, index[i].name.data, index[i].name.len);
```

```
177
178             path.len = (name + index[i].name.len - 1) - path.data;
179
180         } else {
181             e.ip = index[i].values->elts;
182             e.pos = name;
183
184             while (*(uintptr_t *) e.ip) {
185                 code = *(ngx_http_script_code_pt *) e.ip;
186                 code((ngx_http_script_engine_t *) &e);
187             }
188
189             if (*name == '/') {
190                 uri.len = len - 1;
191                 uri.data = name;
192                 return ngx_http_internal_redirect(r, &uri, &r->args);
193             }
194
195             path.len = e.pos - path.data;
196
197             *e.pos = '\0';
198         }
199
200         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
201                        "open index \"%V\"", &path);
202
203         ngx_memzero(&of, sizeof(ngx_open_file_info_t));
204
205         of.read_ahead = clcf->read_ahead;
206         of.directio = clcf->directio;
207         of.valid = clcf->open_file_cache_valid;
208         of.min_uses = clcf->open_file_cache_min_uses;
209         of.test_only = 1;
210         of.errors = clcf->open_file_cache_errors;
211         of.events = clcf->open_file_cache_events;
212
213         if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
214             return NGX_HTTP_INTERNAL_SERVER_ERROR;
215         }
216
217         if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
218             != NGX_OK)
219         {
220             ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, of.err,
221                            "%s \"%s\" failed", of.failed, path.data);
222
223             if (of.err == 0) {
224                 return NGX_HTTP_INTERNAL_SERVER_ERROR;
225             }
226
227 #if (NGX_HAVE_OPENAT)
228             if (of.err == NGX_EMLINK
229                 || of.err == NGX_ELOOP)
230             {
231                 return NGX_HTTP_FORBIDDEN;
232             }
233 #endif
234
235             if (of.err == NGX_ENOTDIR
236                 || of.err == NGX_ENAMETOOLONG
237                 || of.err == NGX_EACCES)
238             {
239                 return ngx_http_index_error(r, clcf, path.data, of.err);
240             }
241
242             if (!dir_tested) {
243                 rc = ngx_http_index_test_dir(r, clcf, path.data, name - 1);
244
245                 if (rc != NGX_OK) {
246                     return rc;
247                 }
248
249                 dir_tested = 1;
250             }
251
252             if (of.err == NGX_ENOENT) {
```

```
                continue;
            }

            ngx_log_error(NGX_LOG_CRIT, r->connection->log, of.err,
                          "%s \"%s\" failed", of.failed, path.data);

            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        uri.len = r->uri.len + len - 1;

        if (!clcf->alias) {
            uri.data = path.data + root;

        } else {
            uri.data = ngx_pnalloc(r->pool, uri.len);
            if (uri.data == NULL) {
                return NGX_HTTP_INTERNAL_SERVER_ERROR;
            }

            p = ngx_copy(uri.data, r->uri.data, r->uri.len);
            ngx_memcpy(p, name, len - 1);
        }

        return ngx_http_internal_redirect(r, &uri, &r->args);
    }

    return NGX_DECLINED;
}


static ngx_int_t
ngx_http_index_test_dir(ngx_http_request_t *r, ngx_http_core_loc_conf_t *clcf,
    u_char *path, u_char *last)
{
    u_char               c;
    ngx_str_t            dir;
    ngx_open_file_info_t of;

    c = *last;
    if (c != '/' || path == last) {
        /* "alias" without trailing slash */
        c = *(++last);
    }
    *last = '\0';

    dir.len = last - path;
    dir.data = path;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http index check dir: \"%V\"", &dir);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.test_dir = 1;
    of.test_only = 1;
    of.valid = clcf->open_file_cache_valid;
    of.errors = clcf->open_file_cache_errors;

    if (ngx_http_set_disable_symlinks(r, clcf, &dir, &of) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (ngx_open_cached_file(clcf->open_file_cache, &dir, &of, r->pool)
        != NGX_OK)
    {
        if (of.err) {

#if (NGX_HAVE_OPENAT)
            if (of.err == NGX_EMLINK
                || of.err == NGX_ELOOP)
            {
                return NGX_HTTP_FORBIDDEN;
            }
#endif

```

```
            if (of.err == NGX_ENOENT) {
                *last = c;
                return ngx_http_index_error(r, clcf, dir.data, NGX_ENOENT);
            }

            if (of.err == NGX_EACCES) {

                *last = c;

                /*
                 * ngx_http_index_test_dir() is called after the first index
                 * file testing has returned an error distinct from NGX_EACCES.
                 * This means that directory searching is allowed.
                 */

                return NGX_OK;
            }

            ngx_log_error(NGX_LOG_CRIT, r->connection->log, of.err,
                          "%s \"%s\" failed", of.failed, dir.data);
        }

        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    *last = c;

    if (of.is_dir) {
        return NGX_OK;
    }

    ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                  "\"%s\" is not a directory", dir.data);

    return NGX_HTTP_INTERNAL_SERVER_ERROR;
}


static ngx_int_t
ngx_http_index_error(ngx_http_request_t *r, ngx_http_core_loc_conf_t  *clcf,
    u_char *file, ngx_err_t err)
{
    if (err == NGX_EACCES) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, err,
                      "\"%s\" is forbidden", file);

        return NGX_HTTP_FORBIDDEN;
    }

    if (clcf->log_not_found) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, err,
                      "\"%s\" is not found", file);
    }

    return NGX_HTTP_NOT_FOUND;
}


static void *
ngx_http_index_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_index_loc_conf_t  *conf;

    conf = ngx_palloc(cf->pool, sizeof(ngx_http_index_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->indices = NULL;
    conf->max_index_len = 0;

    return conf;
}


static char *
```

```
405 ngx_http_index_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
406 {
407     ngx_http_index_loc_conf_t  *prev = parent;
408     ngx_http_index_loc_conf_t  *conf = child;
409
410     ngx_http_index_t  *index;
411
412     if (conf->indices == NULL) {
413         conf->indices = prev->indices;
414         conf->max_index_len = prev->max_index_len;
415     }
416
417     if (conf->indices == NULL) {
418         conf->indices = ngx_array_create(cf->pool, 1, sizeof(ngx_http_index_t));
419         if (conf->indices == NULL) {
420             return NGX_CONF_ERROR;
421         }
422
423         index = ngx_array_push(conf->indices);
424         if (index == NULL) {
425             return NGX_CONF_ERROR;
426         }
427
428         index->name.len = sizeof(NGX_HTTP_DEFAULT_INDEX);
429         index->name.data = (u_char *) NGX_HTTP_DEFAULT_INDEX;
430         index->lengths = NULL;
431         index->values = NULL;
432
433         conf->max_index_len = sizeof(NGX_HTTP_DEFAULT_INDEX);
434
435         return NGX_CONF_OK;
436     }
437
438     return NGX_CONF_OK;
439 }
440
441
442 static ngx_int_t
443 ngx_http_index_init(ngx_conf_t *cf)
444 {
445     ngx_http_handler_pt        *h;
446     ngx_http_core_main_conf_t  *cmcf;
447
448     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
449
450     h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
451     if (h == NULL) {
452         return NGX_ERROR;
453     }
454
455     *h = ngx_http_index_handler;
456
457     return NGX_OK;
458 }
459
460
461 /* TODO: warn about duplicate indices */
462
463 static char *
464 ngx_http_index_set_index(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
465 {
466     ngx_http_index_loc_conf_t *ilcf = conf;
467
468     ngx_str_t                  *value;
469     ngx_uint_t                  i, n;
470     ngx_http_index_t           *index;
471     ngx_http_script_compile_t   sc;
472
473     if (ilcf->indices == NULL) {
474         ilcf->indices = ngx_array_create(cf->pool, 2, sizeof(ngx_http_index_t));
475         if (ilcf->indices == NULL) {
476             return NGX_CONF_ERROR;
477         }
478     }
479
480     value = cf->args->elts;
```

```
481
482     for (i = 1; i < cf->args->nelts; i++) {
483
484         if (value[i].data[0] == '/' && i != cf->args->nelts - 1) {
485             ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
486                                "only the last index in \"index\" directive "
487                                "should be absolute");
488         }
489
490         if (value[i].len == 0) {
491             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
492                                "index \"%V\" in \"index\" directive is invalid",
493                                &value[1]);
494             return NGX_CONF_ERROR;
495         }
496
497         index = ngx_array_push(ilcf->indices);
498         if (index == NULL) {
499             return NGX_CONF_ERROR;
500         }
501
502         index->name.len = value[i].len;
503         index->name.data = value[i].data;
504         index->lengths = NULL;
505         index->values = NULL;
506
507         n = ngx_http_script_variables_count(&value[i]);
508
509         if (n == 0) {
510             if (ilcf->max_index_len < index->name.len) {
511                 ilcf->max_index_len = index->name.len;
512             }
513
514             if (index->name.data[0] == '/') {
515                 continue;
516             }
517
518             /* include the terminating '\0' to the length to use ngx_memcpy() */
519             index->name.len++;
520
521             continue;
522         }
523
524         ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
525
526         sc.cf = cf;
527         sc.source = &value[i];
528         sc.lengths = &index->lengths;
529         sc.values = &index->values;
530         sc.variables = n;
531         sc.complete_lengths = 1;
532         sc.complete_values = 1;
533
534         if (ngx_http_script_compile(&sc) != NGX_OK) {
535             return NGX_CONF_ERROR;
536         }
537     }
538
539     return NGX_CONF_OK;
540 }
```

One Level Up Top Level

src/http/modules/ngx_http_log_module.c - nginx-1.7.10

Global variables defined

- ngx_http_access_log
- ngx_http_combined_fmt
- ngx_http_log_commands
- ngx_http_log_module
- ngx_http_log_module_ctx
- ngx_http_log_vars

Data types defined

- ngx_http_log_buf_t
- ngx_http_log_fmt_t
- ngx_http_log_loc_conf_t
- ngx_http_log_main_conf_t
- ngx_http_log_op_getlen_pt
- ngx_http_log_op_run_pt
- ngx_http_log_op_s
- ngx_http_log_op_t
- ngx_http_log_script_t
- ngx_http_log_t
- ngx_http_log_var_t

Functions defined

- ngx_http_log_body_bytes_sent
- ngx_http_log_bytes_sent
- ngx_http_log_compile_format
- ngx_http_log_copy_long
- ngx_http_log_copy_short
- ngx_http_log_create_loc_conf
- ngx_http_log_create_main_conf
- ngx_http_log_escape
- ngx_http_log_flush
- ngx_http_log_flush_handler

- ngx_http_log_gzip
- ngx_http_log_gzip_alloc
- ngx_http_log_gzip_free
- ngx_http_log_handler
- ngx_http_log_init
- ngx_http_log_iso8601
- ngx_http_log_merge_loc_conf
- ngx_http_log_msec
- ngx_http_log_open_file_cache
- ngx_http_log_pipe
- ngx_http_log_request_length
- ngx_http_log_request_time
- ngx_http_log_script_write
- ngx_http_log_set_format
- ngx_http_log_set_log
- ngx_http_log_status
- ngx_http_log_time
- ngx_http_log_variable
- ngx_http_log_variable_compile
- ngx_http_log_variable_getlen
- ngx_http_log_write

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12  #if (NGX_ZLIB)
13  #include <zlib.h>
14  #endif
15
16
17  typedef struct ngx_http_log_op_s  ngx_http_log_op_t;
18
19  typedef u_char *(*ngx_http_log_op_run_pt) (ngx_http_request_t *r, u_char *buf,
20      ngx_http_log_op_t *op);
21
22  typedef size_t (*ngx_http_log_op_getlen_pt) (ngx_http_request_t *r,
23      uintptr_t data);
24
25
```

```
struct ngx_http_log_op_s {
    size_t                      len;
    ngx_http_log_op_getlen_pt   getlen;
    ngx_http_log_op_run_pt      run;
    uintptr_t                   data;
};


typedef struct {
    ngx_str_t                   name;
    ngx_array_t                *flushes;
    ngx_array_t                *ops;        /* array of ngx_http_log_op_t */
} ngx_http_log_fmt_t;


typedef struct {
    ngx_array_t                 formats;    /* array of ngx_http_log_fmt_t */
    ngx_uint_t                  combined_used; /* unsigned  combined_used:1 */
} ngx_http_log_main_conf_t;


typedef struct {
    u_char                     *start;
    u_char                     *pos;
    u_char                     *last;

    ngx_event_t                *event;
    ngx_msec_t                  flush;
    ngx_int_t                   gzip;
} ngx_http_log_buf_t;


typedef struct {
    ngx_array_t                *lengths;
    ngx_array_t                *values;
} ngx_http_log_script_t;


typedef struct {
    ngx_open_file_t            *file;
    ngx_http_log_script_t      *script;
    time_t                      disk_full_time;
    time_t                      error_log_time;
    ngx_syslog_peer_t          *syslog_peer;
    ngx_http_log_fmt_t         *format;
    ngx_http_complex_value_t   *filter;
} ngx_http_log_t;


typedef struct {
    ngx_array_t                *logs;       /* array of ngx_http_log_t */

    ngx_open_file_cache_t      *open_file_cache;
    time_t                      open_file_cache_valid;
    ngx_uint_t                  open_file_cache_min_uses;

    ngx_uint_t                  off;        /* unsigned  off:1 */
} ngx_http_log_loc_conf_t;


typedef struct {
    ngx_str_t                   name;
    size_t                      len;
    ngx_http_log_op_run_pt      run;
} ngx_http_log_var_t;


static void ngx_http_log_write(ngx_http_request_t *r, ngx_http_log_t *log,
    u_char *buf, size_t len);
static ssize_t ngx_http_log_script_write(ngx_http_request_t *r,
    ngx_http_log_script_t *script, u_char **name, u_char *buf, size_t len);

#if (NGX_ZLIB)
static ssize_t ngx_http_log_gzip(ngx_fd_t fd, u_char *buf, size_t len,
     ngx_int_t level, ngx_log_t *log);

```

```
static void *ngx_http_log_gzip_alloc(void *opaque, u_int items, u_int size);
static void ngx_http_log_gzip_free(void *opaque, void *address);
#endif

static void ngx_http_log_flush(ngx_open_file_t *file, ngx_log_t *log);
static void ngx_http_log_flush_handler(ngx_event_t *ev);

static u_char *ngx_http_log_pipe(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_time(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_iso8601(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_msec(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_request_time(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_status(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_bytes_sent(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static u_char *ngx_http_log_body_bytes_sent(ngx_http_request_t *r,
    u_char *buf, ngx_http_log_op_t *op);
static u_char *ngx_http_log_request_length(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);

static ngx_int_t ngx_http_log_variable_compile(ngx_conf_t *cf,
    ngx_http_log_op_t *op, ngx_str_t *value);
static size_t ngx_http_log_variable_getlen(ngx_http_request_t *r,
    uintptr_t data);
static u_char *ngx_http_log_variable(ngx_http_request_t *r, u_char *buf,
    ngx_http_log_op_t *op);
static uintptr_t ngx_http_log_escape(u_char *dst, u_char *src, size_t size);


static void *ngx_http_log_create_main_conf(ngx_conf_t *cf);
static void *ngx_http_log_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_log_merge_loc_conf(ngx_conf_t *cf, void *parent,
    void *child);
static char *ngx_http_log_set_log(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_log_set_format(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_log_compile_format(ngx_conf_t *cf,
    ngx_array_t *flushes, ngx_array_t *ops, ngx_array_t *args, ngx_uint_t s);
static char *ngx_http_log_open_file_cache(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static ngx_int_t ngx_http_log_init(ngx_conf_t *cf);


static ngx_command_t  ngx_http_log_commands[] = {

    { ngx_string("log_format"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_2MORE,
      ngx_http_log_set_format,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("access_log"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                        |NGX_HTTP_LMT_CONF|NGX_CONF_1MORE,
      ngx_http_log_set_log,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("open_log_file_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1234,
      ngx_http_log_open_file_cache,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};
```

```
178
179
180 static ngx_http_module_t  ngx_http_log_module_ctx = {
181     NULL,                                  /* preconfiguration */
182     ngx_http_log_init,                     /* postconfiguration */
183
184     ngx_http_log_create_main_conf,         /* create main configuration */
185     NULL,                                  /* init main configuration */
186
187     NULL,                                  /* create server configuration */
188     NULL,                                  /* merge server configuration */
189
190     ngx_http_log_create_loc_conf,          /* create location configuration */
191     ngx_http_log_merge_loc_conf            /* merge location configuration */
192 };
193
194
195 ngx_module_t  ngx_http_log_module = {
196     NGX_MODULE_V1,
197     &ngx_http_log_module_ctx,              /* module context */
198     ngx_http_log_commands,                 /* module directives */
199     NGX_HTTP_MODULE,                       /* module type */
200     NULL,                                  /* init master */
201     NULL,                                  /* init module */
202     NULL,                                  /* init process */
203     NULL,                                  /* init thread */
204     NULL,                                  /* exit thread */
205     NULL,                                  /* exit process */
206     NULL,                                  /* exit master */
207     NGX_MODULE_V1_PADDING
208 };
209
210
211 static ngx_str_t  ngx_http_access_log = ngx_string(NGX_HTTP_LOG_PATH);
212
213
214 static ngx_str_t  ngx_http_combined_fmt =
215     ngx_string("$remote_addr - $remote_user [$time_local] "
216                "\"$request\" $status $body_bytes_sent "
217                "\"$http_referer\" \"$http_user_agent\"");
218
219
220 static ngx_http_log_var_t  ngx_http_log_vars[] = {
221     { ngx_string("pipe"), 1, ngx_http_log_pipe },
222     { ngx_string("time_local"), sizeof("28/Sep/1970:12:00:00 +0600") - 1,
223                           ngx_http_log_time },
224     { ngx_string("time_iso8601"), sizeof("1970-09-28T12:00:00+06:00") - 1,
225                           ngx_http_log_iso8601 },
226     { ngx_string("msec"), NGX_TIME_T_LEN + 4, ngx_http_log_msec },
227     { ngx_string("request_time"), NGX_TIME_T_LEN + 4,
228                           ngx_http_log_request_time },
229     { ngx_string("status"), NGX_INT_T_LEN, ngx_http_log_status },
230     { ngx_string("bytes_sent"), NGX_OFF_T_LEN, ngx_http_log_bytes_sent },
231     { ngx_string("body_bytes_sent"), NGX_OFF_T_LEN,
232                           ngx_http_log_body_bytes_sent },
233     { ngx_string("request_length"), NGX_SIZE_T_LEN,
234                           ngx_http_log_request_length },
235
236     { ngx_null_string, 0, NULL }
237 };
238
239
240 static ngx_int_t
241 ngx_http_log_handler(ngx_http_request_t *r)
242 {
243     u_char                  *line, *p;
244     size_t                   len, size;
245     ssize_t                  n;
246     ngx_str_t                val;
247     ngx_uint_t               i, l;
248     ngx_http_log_t          *log;
249     ngx_http_log_op_t       *op;
250     ngx_http_log_buf_t      *buffer;
251     ngx_http_log_loc_conf_t  *lcf;
252
253     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
```

```
254                   "http log handler");
255
256       lcf = ngx_http_get_module_loc_conf(r, ngx_http_log_module);
257
258       if (lcf->off) {
259           return NGX_OK;
260       }
261
262       log = lcf->logs->elts;
263       for (l = 0; l < lcf->logs->nelts; l++) {
264
265           if (log[l].filter) {
266               if (ngx_http_complex_value(r, log[l].filter, &val) != NGX_OK) {
267                   return NGX_ERROR;
268               }
269
270               if (val.len == 0 || (val.len == 1 && val.data[0] == '0')) {
271                   continue;
272               }
273           }
274
275           if (ngx_time() == log[l].disk_full_time) {
276
277               /*
278                * on FreeBSD writing to a full filesystem with enabled softupdates
279                * may block process for much longer time than writing to non-full
280                * filesystem, so we skip writing to a log for one second
281                */
282
283               continue;
284           }
285
286           ngx_http_script_flush_no_cacheable_variables(r, log[l].format->flushes);
287
288           len = 0;
289           op = log[l].format->ops->elts;
290           for (i = 0; i < log[l].format->ops->nelts; i++) {
291               if (op[i].len == 0) {
292                   len += op[i].getlen(r, op[i].data);
293
294               } else {
295                   len += op[i].len;
296               }
297           }
298
299           if (log[l].syslog_peer) {
300
301               /* length of syslog's PRI and HEADER message parts */
302               len += sizeof("<255>Jan 01 00:00:00 ") - 1
303                      + ngx_cycle->hostname.len + 1
304                      + log[l].syslog_peer->tag.len + 2;
305
306               goto alloc_line;
307           }
308
309           len += NGX_LINEFEED_SIZE;
310
311           buffer = log[l].file ? log[l].file->data : NULL;
312
313           if (buffer) {
314
315               if (len > (size_t) (buffer->last - buffer->pos)) {
316
317                   ngx_http_log_write(r, &log[l], buffer->start,
318                                      buffer->pos - buffer->start);
319
320                   buffer->pos = buffer->start;
321               }
322
323               if (len <= (size_t) (buffer->last - buffer->pos)) {
324
325                   p = buffer->pos;
326
327                   if (buffer->event && p == buffer->start) {
328                       ngx_add_timer(buffer->event, buffer->flush);
329                   }
```

```
                for (i = 0; i < log[l].format->ops->nelts; i++) {
                    p = op[i].run(r, p, &op[i]);
                }

                ngx_linefeed(p);

                buffer->pos = p;

                continue;
            }

            if (buffer->event && buffer->event->timer_set) {
                ngx_del_timer(buffer->event);
            }
        }

    alloc_line:

        line = ngx_pnalloc(r->pool, len);
        if (line == NULL) {
            return NGX_ERROR;
        }

        p = line;

        if (log[l].syslog_peer) {
            p = ngx_syslog_add_header(log[l].syslog_peer, line);
        }

        for (i = 0; i < log[l].format->ops->nelts; i++) {
            p = op[i].run(r, p, &op[i]);
        }

        if (log[l].syslog_peer) {

            size = p - line;

            n = ngx_syslog_send(log[l].syslog_peer, line, size);

            if (n < 0) {
                ngx_log_error(NGX_LOG_WARN, r->connection->log, 0,
                              "send() to syslog failed");

            } else if ((size_t) n != size) {
                ngx_log_error(NGX_LOG_WARN, r->connection->log, 0,
                              "send() to syslog has written only %z of %uz",
                              n, size);
            }

            continue;
        }

        ngx_linefeed(p);

        ngx_http_log_write(r, &log[l], line, p - line);
    }

    return NGX_OK;
}


static void
ngx_http_log_write(ngx_http_request_t *r, ngx_http_log_t *log, u_char *buf,
    size_t len)
{
    u_char              *name;
    time_t               now;
    ssize_t              n;
    ngx_err_t            err;
#if (NGX_ZLIB)
    ngx_http_log_buf_t  *buffer;
#endif

    if (log->script == NULL) {
        name = log->file->name.data;
```

```
406
407 #if (NGX_ZLIB)
408         buffer = log->file->data;
409
410         if (buffer && buffer->gzip) {
411             n = ngx_http_log_gzip(log->file->fd, buf, len, buffer->gzip,
412                                   r->connection->log);
413         } else {
414             n = ngx_write_fd(log->file->fd, buf, len);
415         }
416 #else
417         n = ngx_write_fd(log->file->fd, buf, len);
418 #endif
419
420     } else {
421         name = NULL;
422         n = ngx_http_log_script_write(r, log->script, &name, buf, len);
423     }
424
425     if (n == (ssize_t) len) {
426         return;
427     }
428
429     now = ngx_time();
430
431     if (n == -1) {
432         err = ngx_errno;
433
434         if (err == NGX_ENOSPC) {
435             log->disk_full_time = now;
436         }
437
438         if (now - log->error_log_time > 59) {
439             ngx_log_error(NGX_LOG_ALERT, r->connection->log, err,
440                           ngx_write_fd_n " to \"%s\" failed", name);
441
442             log->error_log_time = now;
443         }
444
445         return;
446     }
447
448     if (now - log->error_log_time > 59) {
449         ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
450                       ngx_write_fd_n " to \"%s\" was incomplete: %z of %uz",
451                       name, n, len);
452
453         log->error_log_time = now;
454     }
455 }
456
457
458 static ssize_t
459 ngx_http_log_script_write(ngx_http_request_t *r, ngx_http_log_script_t *script,
460     u_char **name, u_char *buf, size_t len)
461 {
462     size_t                      root;
463     ssize_t                     n;
464     ngx_str_t                   log, path;
465     ngx_open_file_info_t        of;
466     ngx_http_log_loc_conf_t    *llcf;
467     ngx_http_core_loc_conf_t   *clcf;
468
469     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
470
471     if (!r->root_tested) {
472
473         /* test root directory existence */
474
475         if (ngx_http_map_uri_to_path(r, &path, &root, 0) == NULL) {
476             /* simulate successful logging */
477             return len;
478         }
479
480         path.data[root] = '\0';
481
```

```
        ngx_memzero(&of, sizeof(ngx_open_file_info_t));

        of.valid = clcf->open_file_cache_valid;
        of.min_uses = clcf->open_file_cache_min_uses;
        of.test_dir = 1;
        of.test_only = 1;
        of.errors = clcf->open_file_cache_errors;
        of.events = clcf->open_file_cache_events;

        if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
            /* simulate successful logging */
            return len;
        }

        if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
            != NGX_OK)
        {
            if (of.err == 0) {
                /* simulate successful logging */
                return len;
            }

            ngx_log_error(NGX_LOG_ERR, r->connection->log, of.err,
                          "testing \"%s\" existence failed", path.data);

            /* simulate successful logging */
            return len;
        }

        if (!of.is_dir) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, NGX_ENOTDIR,
                          "testing \"%s\" existence failed", path.data);

            /* simulate successful logging */
            return len;
        }
    }

    if (ngx_http_script_run(r, &log, script->lengths->elts, 1,
                            script->values->elts)
        == NULL)
    {
        /* simulate successful logging */
        return len;
    }

    log.data[log.len - 1] = '\0';
    *name = log.data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http log \"%s\"", log.data);

    llcf = ngx_http_get_module_loc_conf(r, ngx_http_log_module);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.log = 1;
    of.valid = llcf->open_file_cache_valid;
    of.min_uses = llcf->open_file_cache_min_uses;
    of.directio = NGX_OPEN_FILE_DIRECTIO_OFF;

    if (ngx_http_set_disable_symlinks(r, clcf, &log, &of) != NGX_OK) {
        /* simulate successful logging */
        return len;
    }

    if (ngx_open_cached_file(llcf->open_file_cache, &log, &of, r->pool)
        != NGX_OK)
    {
        ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
                      "%s \"%s\" failed", of.failed, log.data);
        /* simulate successful logging */
        return len;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
```

```
                   "http log #%d", of.fd);

    n = ngx_write_fd(of.fd, buf, len);

    return n;
}


#if (NGX_ZLIB)

static ssize_t
ngx_http_log_gzip(ngx_fd_t fd, u_char *buf, size_t len, ngx_int_t level,
    ngx_log_t *log)
{
    int          rc, wbits, memlevel;
    u_char      *out;
    size_t       size;
    ssize_t      n;
    z_stream     zstream;
    ngx_err_t    err;
    ngx_pool_t  *pool;

    wbits = MAX_WBITS;
    memlevel = MAX_MEM_LEVEL - 1;

    while ((ssize_t) len < ((1 << (wbits - 1)) - 262)) {
        wbits--;
        memlevel--;
    }

    /*
     * This is a formula from deflateBound() for conservative upper bound of
     * compressed data plus 18 bytes of gzip wrapper.
     */

    size = len + ((len + 7) >> 3) + ((len + 63) >> 6) + 5 + 18;

    ngx_memzero(&zstream, sizeof(z_stream));

    pool = ngx_create_pool(256, log);
    if (pool == NULL) {
        /* simulate successful logging */
        return len;
    }

    pool->log = log;

    zstream.zalloc = ngx_http_log_gzip_alloc;
    zstream.zfree = ngx_http_log_gzip_free;
    zstream.opaque = pool;

    out = ngx_pnalloc(pool, size);
    if (out == NULL) {
        goto done;
    }

    zstream.next_in = buf;
    zstream.avail_in = len;
    zstream.next_out = out;
    zstream.avail_out = size;

    rc = deflateInit2(&zstream, (int) level, Z_DEFLATED, wbits + 16, memlevel,
                      Z_DEFAULT_STRATEGY);

    if (rc != Z_OK) {
        ngx_log_error(NGX_LOG_ALERT, log, 0, "deflateInit2() failed: %d", rc);
        goto done;
    }

    ngx_log_debug4(NGX_LOG_DEBUG_HTTP, log, 0,
                   "deflate in: ni:%p no:%p ai:%ud ao:%ud",
                   zstream.next_in, zstream.next_out,
                   zstream.avail_in, zstream.avail_out);

    rc = deflate(&zstream, Z_FINISH);

```

```
    if (rc != Z_STREAM_END) {
        ngx_log_error(NGX_LOG_ALERT, log, 0,
                      "deflate(Z_FINISH) failed: %d", rc);
        goto done;
    }

    ngx_log_debug5(NGX_LOG_DEBUG_HTTP, log, 0,
                   "deflate out: ni:%p no:%p ai:%ud ao:%ud rc:%d",
                   zstream.next_in, zstream.next_out,
                   zstream.avail_in, zstream.avail_out,
                   rc);

    size -= zstream.avail_out;

    rc = deflateEnd(&zstream);

    if (rc != Z_OK) {
        ngx_log_error(NGX_LOG_ALERT, log, 0, "deflateEnd() failed: %d", rc);
        goto done;
    }

    n = ngx_write_fd(fd, out, size);

    if (n != (ssize_t) size) {
        err = (n == -1) ? ngx_errno : 0;

        ngx_destroy_pool(pool);

        ngx_set_errno(err);
        return -1;
    }

done:

    ngx_destroy_pool(pool);

    /* simulate successful logging */
    return len;
}


static void *
ngx_http_log_gzip_alloc(void *opaque, u_int items, u_int size)
{
    ngx_pool_t *pool = opaque;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pool->log, 0,
                   "gzip alloc: n:%ud s:%ud", items, size);

    return ngx_palloc(pool, items * size);
}


static void
ngx_http_log_gzip_free(void *opaque, void *address)
{
#if 0
    ngx_pool_t *pool = opaque;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pool->log, 0, "gzip free: %p", address);
#endif
}

#endif


static void
ngx_http_log_flush(ngx_open_file_t *file, ngx_log_t *log)
{
    size_t               len;
    ssize_t              n;
    ngx_http_log_buf_t  *buffer;

    buffer = file->data;

    len = buffer->pos - buffer->start;
```

```
710
711     if (len == 0) {
712         return;
713     }
714
715 #if (NGX_ZLIB)
716     if (buffer->gzip) {
717         n = ngx_http_log_gzip(file->fd, buffer->start, len, buffer->gzip, log);
718     } else {
719         n = ngx_write_fd(file->fd, buffer->start, len);
720     }
721 #else
722     n = ngx_write_fd(file->fd, buffer->start, len);
723 #endif
724
725     if (n == -1) {
726         ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
727                       ngx_write_fd_n " to \"%s\" failed",
728                       file->name.data);
729
730     } else if ((size_t) n != len) {
731         ngx_log_error(NGX_LOG_ALERT, log, 0,
732                       ngx_write_fd_n " to \"%s\" was incomplete: %z of %uz",
733                       file->name.data, n, len);
734     }
735
736     buffer->pos = buffer->start;
737
738     if (buffer->event && buffer->event->timer_set) {
739         ngx_del_timer(buffer->event);
740     }
741 }
742
743
744 static void
745 ngx_http_log_flush_handler(ngx_event_t *ev)
746 {
747     ngx_open_file_t     *file;
748     ngx_http_log_buf_t  *buffer;
749
750     ngx_log_debug0(NGX_LOG_DEBUG_EVENT, ev->log, 0,
751                    "http log buffer flush handler");
752
753     if (ev->timedout) {
754         ngx_http_log_flush(ev->data, ev->log);
755         return;
756     }
757
758     /* cancel the flush timer for graceful shutdown */
759
760     file = ev->data;
761     buffer = file->data;
762
763     buffer->event = NULL;
764 }
765
766
767 static u_char *
768 ngx_http_log_copy_short(ngx_http_request_t *r, u_char *buf,
769     ngx_http_log_op_t *op)
770 {
771     size_t     len;
772     uintptr_t  data;
773
774     len = op->len;
775     data = op->data;
776
777     while (len--) {
778         *buf++ = (u_char) (data & 0xff);
779         data >>= 8;
780     }
781
782     return buf;
783 }
784
785
```

```
786 static u_char *
787 ngx_http_log_copy_long(ngx_http_request_t *r, u_char *buf,
788     ngx_http_log_op_t *op)
789 {
790     return ngx_cpymem(buf, (u_char *) op->data, op->len);
791 }
792
793
794 static u_char *
795 ngx_http_log_pipe(ngx_http_request_t *r, u_char *buf, ngx_http_log_op_t *op)
796 {
797     if (r->pipeline) {
798         *buf = 'p';
799     } else {
800         *buf = '.';
801     }
802
803     return buf + 1;
804 }
805
806
807 static u_char *
808 ngx_http_log_time(ngx_http_request_t *r, u_char *buf, ngx_http_log_op_t *op)
809 {
810     return ngx_cpymem(buf, ngx_cached_http_log_time.data,
811                       ngx_cached_http_log_time.len);
812 }
813
814 static u_char *
815 ngx_http_log_iso8601(ngx_http_request_t *r, u_char *buf, ngx_http_log_op_t *op)
816 {
817     return ngx_cpymem(buf, ngx_cached_http_log_iso8601.data,
818                       ngx_cached_http_log_iso8601.len);
819 }
820
821 static u_char *
822 ngx_http_log_msec(ngx_http_request_t *r, u_char *buf, ngx_http_log_op_t *op)
823 {
824     ngx_time_t  *tp;
825
826     tp = ngx_timeofday();
827
828     return ngx_sprintf(buf, "%T.%03M", tp->sec, tp->msec);
829 }
830
831
832 static u_char *
833 ngx_http_log_request_time(ngx_http_request_t *r, u_char *buf,
834     ngx_http_log_op_t *op)
835 {
836     ngx_time_t      *tp;
837     ngx_msec_int_t   ms;
838
839     tp = ngx_timeofday();
840
841     ms = (ngx_msec_int_t)
842             ((tp->sec - r->start_sec) * 1000 + (tp->msec - r->start_msec));
843     ms = ngx_max(ms, 0);
844
845     return ngx_sprintf(buf, "%T.%03M", (time_t) ms / 1000, ms % 1000);
846 }
847
848
849 static u_char *
850 ngx_http_log_status(ngx_http_request_t *r, u_char *buf, ngx_http_log_op_t *op)
851 {
852     ngx_uint_t  status;
853
854     if (r->err_status) {
855         status = r->err_status;
856
857     } else if (r->headers_out.status) {
858         status = r->headers_out.status;
859
860     } else if (r->http_version == NGX_HTTP_VERSION_9) {
861         status = 9;
```

```
862
863     } else {
864         status = 0;
865     }
866
867     return ngx_sprintf(buf, "%03ui", status);
868 }
869
870
871 static u_char *
872 ngx_http_log_bytes_sent(ngx_http_request_t *r, u_char *buf,
873     ngx_http_log_op_t *op)
874 {
875     return ngx_sprintf(buf, "%O", r->connection->sent);
876 }
877
878
879 /*
880 * although there is a real $body_bytes_sent variable,
881 * this log operation code function is more optimized for logging
882 */
883
884 static u_char *
885 ngx_http_log_body_bytes_sent(ngx_http_request_t *r, u_char *buf,
886     ngx_http_log_op_t *op)
887 {
888     off_t  length;
889
890     length = r->connection->sent - r->header_size;
891
892     if (length > 0) {
893         return ngx_sprintf(buf, "%O", length);
894     }
895
896     *buf = '0';
897
898     return buf + 1;
899 }
900
901
902 static u_char *
903 ngx_http_log_request_length(ngx_http_request_t *r, u_char *buf,
904     ngx_http_log_op_t *op)
905 {
906     return ngx_sprintf(buf, "%O", r->request_length);
907 }
908
909
910 static ngx_int_t
911 ngx_http_log_variable_compile(ngx_conf_t *cf, ngx_http_log_op_t *op,
912     ngx_str_t *value)
913 {
914     ngx_int_t  index;
915
916     index = ngx_http_get_variable_index(cf, value);
917     if (index == NGX_ERROR) {
918         return NGX_ERROR;
919     }
920
921     op->len = 0;
922     op->getlen = ngx_http_log_variable_getlen;
923     op->run = ngx_http_log_variable;
924     op->data = index;
925
926     return NGX_OK;
927 }
928
929
930 static size_t
931 ngx_http_log_variable_getlen(ngx_http_request_t *r, uintptr_t data)
932 {
933     uintptr_t                   len;
934     ngx_http_variable_value_t  *value;
935
936     value = ngx_http_get_indexed_variable(r, data);
937
```

```
    if (value == NULL || value->not_found) {
        return 1;
    }

    len = ngx_http_log_escape(NULL, value->data, value->len);

    value->escape = len ? 1 : 0;

    return value->len + len * 3;
}


static u_char *
ngx_http_log_variable(ngx_http_request_t *r, u_char *buf, ngx_http_log_op_t *op)
{
    ngx_http_variable_value_t  *value;

    value = ngx_http_get_indexed_variable(r, op->data);

    if (value == NULL || value->not_found) {
        *buf = '-';
        return buf + 1;
    }

    if (value->escape == 0) {
        return ngx_cpymem(buf, value->data, value->len);

    } else {
        return (u_char *) ngx_http_log_escape(buf, value->data, value->len);
    }
}


static uintptr_t
ngx_http_log_escape(u_char *dst, u_char *src, size_t size)
{
    ngx_uint_t      n;
    static u_char   hex[] = "0123456789ABCDEF";

    static uint32_t   escape[] = {
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */

                    /* ?>=< ;:98 7654 3210  /.-, +*)( '&%$ #"!  */
        0x00000004, /* 0000 0000 0000 0000  0000 0000 0000 0100 */

                    /* _^]\ [ZYX WVUT SRQP  ONML KJIH GFED CBA@ */
        0x10000000, /* 0001 0000 0000 0000  0000 0000 0000 0000 */

                    /*  ~}| {zyx wvut srqp  onml kjih gfed cba` */
        0x80000000, /* 1000 0000 0000 0000  0000 0000 0000 0000 */

        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
        0xffffffff, /* 1111 1111 1111 1111  1111 1111 1111 1111 */
    };


    if (dst == NULL) {

        /* find the number of the characters to be escaped */

         n = 0;

         while (size) {
             if (escape[*src >> 5] & (1 << (*src & 0x1f))) {
                 n++;
             }
             src++;
             size--;
         }

         return (uintptr_t) n;
     }

     while (size) {
```

```
        if (escape[*src >> 5] & (1 << (*src & 0x1f))) {
            *dst++ = '\\';
            *dst++ = 'x';
            *dst++ = hex[*src >> 4];
            *dst++ = hex[*src & 0xf];
            src++;

        } else {
            *dst++ = *src++;
        }
        size--;
    }

    return (uintptr_t) dst;
}


static void *
ngx_http_log_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_log_main_conf_t  *conf;

    ngx_http_log_fmt_t  *fmt;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_log_main_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    if (ngx_array_init(&conf->formats, cf->pool, 4, sizeof(ngx_http_log_fmt_t))
        != NGX_OK)
    {
        return NULL;
    }

    fmt = ngx_array_push(&conf->formats);
    if (fmt == NULL) {
        return NULL;
    }

    ngx_str_set(&fmt->name, "combined");

    fmt->flushes = NULL;

    fmt->ops = ngx_array_create(cf->pool, 16, sizeof(ngx_http_log_op_t));
    if (fmt->ops == NULL) {
        return NULL;
    }

    return conf;
}


static void *
ngx_http_log_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_log_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_log_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->open_file_cache = NGX_CONF_UNSET_PTR;

    return conf;
}


static char *
ngx_http_log_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_log_loc_conf_t *prev = parent;
    ngx_http_log_loc_conf_t *conf = child;

    ngx_http_log_t            *log;
```

```
1090      ngx_http_log_fmt_t        *fmt;
1091      ngx_http_log_main_conf_t  *lmcf;
1092
1093      if (conf->open_file_cache == NGX_CONF_UNSET_PTR) {
1094
1095          conf->open_file_cache = prev->open_file_cache;
1096          conf->open_file_cache_valid = prev->open_file_cache_valid;
1097          conf->open_file_cache_min_uses = prev->open_file_cache_min_uses;
1098
1099          if (conf->open_file_cache == NGX_CONF_UNSET_PTR) {
1100              conf->open_file_cache = NULL;
1101          }
1102      }
1103
1104      if (conf->logs || conf->off) {
1105          return NGX_CONF_OK;
1106      }
1107
1108      conf->logs = prev->logs;
1109      conf->off = prev->off;
1110
1111      if (conf->logs || conf->off) {
1112          return NGX_CONF_OK;
1113      }
1114
1115      conf->logs = ngx_array_create(cf->pool, 2, sizeof(ngx_http_log_t));
1116      if (conf->logs == NULL) {
1117          return NGX_CONF_ERROR;
1118      }
1119
1120      log = ngx_array_push(conf->logs);
1121      if (log == NULL) {
1122          return NGX_CONF_ERROR;
1123      }
1124
1125      ngx_memzero(log, sizeof(ngx_http_log_t));
1126
1127      log->file = ngx_conf_open_file(cf->cycle, &ngx_http_access_log);
1128      if (log->file == NULL) {
1129          return NGX_CONF_ERROR;
1130      }
1131
1132      lmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_log_module);
1133      fmt = lmcf->formats.elts;
1134
1135      /* the default "combined" format */
1136      log->format = &fmt[0];
1137      lmcf->combined_used = 1;
1138
1139      return NGX_CONF_OK;
1140  }
1141
1142
1143  static char *
1144  ngx_http_log_set_log(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1145  {
1146      ngx_http_log_loc_conf_t *llcf = conf;
1147
1148      ssize_t                          size;
1149      ngx_int_t                        gzip;
1150      ngx_uint_t                       i, n;
1151      ngx_msec_t                       flush;
1152      ngx_str_t                       *value, name, s;
1153      ngx_http_log_t                  *log;
1154      ngx_syslog_peer_t               *peer;
1155      ngx_http_log_buf_t              *buffer;
1156      ngx_http_log_fmt_t              *fmt;
1157      ngx_http_log_main_conf_t        *lmcf;
1158      ngx_http_script_compile_t        sc;
1159      ngx_http_compile_complex_value_t ccv;
1160
1161      value = cf->args->elts;
1162
1163      if (ngx_strcmp(value[1].data, "off") == 0) {
1164          llcf->off = 1;
1165          if (cf->args->nelts == 2) {
```

```
            return NGX_CONF_OK;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid parameter \"%V\"", &value[2]);
        return NGX_CONF_ERROR;
    }

    if (llcf->logs == NULL) {
        llcf->logs = ngx_array_create(cf->pool, 2, sizeof(ngx_http_log_t));
        if (llcf->logs == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    lmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_log_module);

    log = ngx_array_push(llcf->logs);
    if (log == NULL) {
        return NGX_CONF_ERROR;
    }

    ngx_memzero(log, sizeof(ngx_http_log_t));


    if (ngx_strncmp(value[1].data, "syslog:", 7) == 0) {

        peer = ngx_pcalloc(cf->pool, sizeof(ngx_syslog_peer_t));
        if (peer == NULL) {
            return NGX_CONF_ERROR;
        }

        if (ngx_syslog_process_conf(cf, peer) != NGX_CONF_OK) {
            return NGX_CONF_ERROR;
        }

        log->syslog_peer = peer;

        goto process_formats;
    }

    n = ngx_http_script_variables_count(&value[1]);

    if (n == 0) {
        log->file = ngx_conf_open_file(cf->cycle, &value[1]);
        if (log->file == NULL) {
            return NGX_CONF_ERROR;
        }

    } else {
        if (ngx_conf_full_name(cf->cycle, &value[1], 0) != NGX_OK) {
            return NGX_CONF_ERROR;
        }

        log->script = ngx_pcalloc(cf->pool, sizeof(ngx_http_log_script_t));
        if (log->script == NULL) {
            return NGX_CONF_ERROR;
        }

        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

        sc.cf = cf;
        sc.source = &value[1];
        sc.lengths = &log->script->lengths;
        sc.values = &log->script->values;
        sc.variables = n;
        sc.complete_lengths = 1;
        sc.complete_values = 1;

        if (ngx_http_script_compile(&sc) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

process_formats:

```

```
    if (cf->args->nelts >= 3) {
        name = value[2];

        if (ngx_strcmp(name.data, "combined") == 0) {
            lmcf->combined_used = 1;
        }

    } else {
        ngx_str_set(&name, "combined");
        lmcf->combined_used = 1;
    }

    fmt = lmcf->formats.elts;
    for (i = 0; i < lmcf->formats.nelts; i++) {
        if (fmt[i].name.len == name.len
            && ngx_strcasecmp(fmt[i].name.data, name.data) == 0)
        {
            log->format = &fmt[i];
            break;
        }
    }

    if (log->format == NULL) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "unknown log format \"%V\"", &name);
        return NGX_CONF_ERROR;
    }

    size = 0;
    flush = 0;
    gzip = 0;

    for (i = 3; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "buffer=", 7) == 0) {
            s.len = value[i].len - 7;
            s.data = value[i].data + 7;

            size = ngx_parse_size(&s);

            if (size == NGX_ERROR || size == 0) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid buffer size \"%V\"", &s);
                return NGX_CONF_ERROR;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "flush=", 6) == 0) {
            s.len = value[i].len - 6;
            s.data = value[i].data + 6;

            flush = ngx_parse_time(&s, 0);

            if (flush == (ngx_msec_t) NGX_ERROR || flush == 0) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "invalid flush time \"%V\"", &s);
                return NGX_CONF_ERROR;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "gzip", 4) == 0
            && (value[i].len == 4 || value[i].data[4] == '='))
        {
#if (NGX_ZLIB)
            if (size == 0) {
                size = 64 * 1024;
            }

            if (value[i].len == 4) {
                gzip = Z_BEST_SPEED;
                continue;
            }
```

```
1318
1319            s.len = value[i].len - 5;
1320            s.data = value[i].data + 5;
1321
1322            gzip = ngx_atoi(s.data, s.len);
1323
1324            if (gzip < 1 || gzip > 9) {
1325                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1326                                   "invalid compression level \"%V\"", &s);
1327                return NGX_CONF_ERROR;
1328            }
1329
1330            continue;
1331
1332 #else
1333            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1334                               "nginx was built without zlib support");
1335            return NGX_CONF_ERROR;
1336 #endif
1337        }
1338
1339        if (ngx_strncmp(value[i].data, "if=", 3) == 0) {
1340            s.len = value[i].len - 3;
1341            s.data = value[i].data + 3;
1342
1343            ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
1344
1345            ccv.cf = cf;
1346            ccv.value = &s;
1347            ccv.complex_value = ngx_palloc(cf->pool,
1348                                           sizeof(ngx_http_complex_value_t));
1349            if (ccv.complex_value == NULL) {
1350                return NGX_CONF_ERROR;
1351            }
1352
1353            if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
1354                return NGX_CONF_ERROR;
1355            }
1356
1357            log->filter = ccv.complex_value;
1358
1359            continue;
1360        }
1361
1362        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1363                           "invalid parameter \"%V\"", &value[i]);
1364        return NGX_CONF_ERROR;
1365    }
1366
1367    if (flush && size == 0) {
1368        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1369                           "no buffer is defined for access_log \"%V\"",
1370                           &value[1]);
1371        return NGX_CONF_ERROR;
1372    }
1373
1374    if (size) {
1375
1376        if (log->script) {
1377            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1378                               "buffered logs cannot have variables in name");
1379            return NGX_CONF_ERROR;
1380        }
1381
1382        if (log->syslog_peer) {
1383            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1384                               "logs to syslog cannot be buffered");
1385            return NGX_CONF_ERROR;
1386        }
1387
1388        if (log->file->data) {
1389            buffer = log->file->data;
1390
1391            if (buffer->last - buffer->start != size
1392                || buffer->flush != flush
1393                || buffer->gzip != gzip)
```

```
            {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "access_log \"%V\" already defined "
                                   "with conflicting parameters",
                                   &value[1]);
                return NGX_CONF_ERROR;
            }

            return NGX_CONF_OK;
        }

        buffer = ngx_pcalloc(cf->pool, sizeof(ngx_http_log_buf_t));
        if (buffer == NULL) {
            return NGX_CONF_ERROR;
        }

        buffer->start = ngx_pnalloc(cf->pool, size);
        if (buffer->start == NULL) {
            return NGX_CONF_ERROR;
        }

        buffer->pos = buffer->start;
        buffer->last = buffer->start + size;

        if (flush) {
            buffer->event = ngx_pcalloc(cf->pool, sizeof(ngx_event_t));
            if (buffer->event == NULL) {
                return NGX_CONF_ERROR;
            }

            buffer->event->data = log->file;
            buffer->event->handler = ngx_http_log_flush_handler;
            buffer->event->log = &cf->cycle->new_log;
            buffer->event->cancelable = 1;

            buffer->flush = flush;
        }

        buffer->gzip = gzip;

        log->file->flush = ngx_http_log_flush;
        log->file->data = buffer;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_log_set_format(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_log_main_conf_t *lmcf = conf;

    ngx_str_t           *value;
    ngx_uint_t           i;
    ngx_http_log_fmt_t  *fmt;

    value = cf->args->elts;

    fmt = lmcf->formats.elts;
    for (i = 0; i < lmcf->formats.nelts; i++) {
        if (fmt[i].name.len == value[1].len
            && ngx_strcmp(fmt[i].name.data, value[1].data) == 0)
        {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "duplicate \"log_format\" name \"%V\"",
                               &value[1]);
            return NGX_CONF_ERROR;
        }
    }

    fmt = ngx_array_push(&lmcf->formats);
    if (fmt == NULL) {
        return NGX_CONF_ERROR;
    }

```

```
1470      fmt->name = value[1];
1471
1472      fmt->flushes = ngx_array_create(cf->pool, 4, sizeof(ngx_int_t));
1473      if (fmt->flushes == NULL) {
1474          return NGX_CONF_ERROR;
1475      }
1476
1477      fmt->ops = ngx_array_create(cf->pool, 16, sizeof(ngx_http_log_op_t));
1478      if (fmt->ops == NULL) {
1479          return NGX_CONF_ERROR;
1480      }
1481
1482      return ngx_http_log_compile_format(cf, fmt->flushes, fmt->ops, cf->args, 2);
1483  }
1484
1485
1486  static char *
1487  ngx_http_log_compile_format(ngx_conf_t *cf, ngx_array_t *flushes,
1488      ngx_array_t *ops, ngx_array_t *args, ngx_uint_t s)
1489  {
1490      u_char              *data, *p, ch;
1491      size_t               i, len;
1492      ngx_str_t           *value, var;
1493      ngx_int_t           *flush;
1494      ngx_uint_t           bracket;
1495      ngx_http_log_op_t   *op;
1496      ngx_http_log_var_t  *v;
1497
1498      value = args->elts;
1499
1500      for ( /* void */ ; s < args->nelts; s++) {
1501
1502          i = 0;
1503
1504          while (i < value[s].len) {
1505
1506              op = ngx_array_push(ops);
1507              if (op == NULL) {
1508                  return NGX_CONF_ERROR;
1509              }
1510
1511              data = &value[s].data[i];
1512
1513              if (value[s].data[i] == '$') {
1514
1515                  if (++i == value[s].len) {
1516                      goto invalid;
1517                  }
1518
1519                  if (value[s].data[i] == '{') {
1520                      bracket = 1;
1521
1522                      if (++i == value[s].len) {
1523                          goto invalid;
1524                      }
1525
1526                      var.data = &value[s].data[i];
1527
1528                  } else {
1529                      bracket = 0;
1530                      var.data = &value[s].data[i];
1531                  }
1532
1533                  for (var.len = 0; i < value[s].len; i++, var.len++) {
1534                      ch = value[s].data[i];
1535
1536                      if (ch == '}' && bracket) {
1537                          i++;
1538                          bracket = 0;
1539                          break;
1540                      }
1541
1542                      if ((ch >= 'A' && ch <= 'Z')
1543                          || (ch >= 'a' && ch <= 'z')
1544                          || (ch >= '0' && ch <= '9')
1545                          || ch == '_')
```

```
                {
                    continue;
                }

                break;
            }

            if (bracket) {
                ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                                   "the closing bracket in \"%V\" "
                                   "variable is missing", &var);
                return NGX_CONF_ERROR;
            }

            if (var.len == 0) {
                goto invalid;
            }

            for (v = ngx_http_log_vars; v->name.len; v++) {

                if (v->name.len == var.len
                    && ngx_strncmp(v->name.data, var.data, var.len) == 0)
                {
                    op->len = v->len;
                    op->getlen = NULL;
                    op->run = v->run;
                    op->data = 0;

                    goto found;
                }
            }

            if (ngx_http_log_variable_compile(cf, op, &var) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

            if (flushes) {

                flush = ngx_array_push(flushes);
                if (flush == NULL) {
                    return NGX_CONF_ERROR;
                }

                *flush = op->data; /* variable index */
            }

        found:

            continue;
        }

        i++;

        while (i < value[s].len && value[s].data[i] != '$') {
            i++;
        }

        len = &value[s].data[i] - data;

        if (len) {

            op->len = len;
            op->getlen = NULL;

            if (len <= sizeof(uintptr_t)) {
                op->run = ngx_http_log_copy_short;
                op->data = 0;

                while (len--) {
                    op->data <<= 8;
                    op->data |= data[len];
                }

            } else {
                op->run = ngx_http_log_copy_long;

```

```
                    p = ngx_pnalloc(cf->pool, len);
                    if (p == NULL) {
                        return NGX_CONF_ERROR;
                    }

                    ngx_memcpy(p, data, len);
                    op->data = (uintptr_t) p;
                }
            }
        }
    }

    return NGX_CONF_OK;

invalid:

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid parameter \"%s\"", data);

    return NGX_CONF_ERROR;
}


static char *
ngx_http_log_open_file_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_log_loc_conf_t *llcf = conf;

    time_t       inactive, valid;
    ngx_str_t   *value, s;
    ngx_int_t    max, min_uses;
    ngx_uint_t   i;

    if (llcf->open_file_cache != NGX_CONF_UNSET_PTR) {
        return "is duplicate";
    }

    value = cf->args->elts;

    max = 0;
    inactive = 10;
    valid = 60;
    min_uses = 1;

    for (i = 1; i < cf->args->nelts; i++) {

        if (ngx_strncmp(value[i].data, "max=", 4) == 0) {

            max = ngx_atoi(value[i].data + 4, value[i].len - 4);
            if (max == NGX_ERROR) {
                goto failed;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "inactive=", 9) == 0) {

            s.len = value[i].len - 9;
            s.data = value[i].data + 9;

            inactive = ngx_parse_time(&s, 1);
            if (inactive == (time_t) NGX_ERROR) {
                goto failed;
            }

            continue;
        }

        if (ngx_strncmp(value[i].data, "min_uses=", 9) == 0) {

            min_uses = ngx_atoi(value[i].data + 9, value[i].len - 9);
            if (min_uses == NGX_ERROR) {
                goto failed;
            }

            continue;
```

```
        }

        if (ngx_strncmp(value[i].data, "valid=", 6) == 0) {

            s.len = value[i].len - 6;
            s.data = value[i].data + 6;

            valid = ngx_parse_time(&s, 1);
            if (valid == (time_t) NGX_ERROR) {
                goto failed;
            }

            continue;
        }

        if (ngx_strcmp(value[i].data, "off") == 0) {

            llcf->open_file_cache = NULL;

            continue;
        }

    failed:

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid \"open_log_file_cache\" parameter \"%V\"",
                           &value[i]);
        return NGX_CONF_ERROR;
    }

    if (llcf->open_file_cache == NULL) {
        return NGX_CONF_OK;
    }

    if (max == 0) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                        "\"open_log_file_cache\" must have \"max\" parameter");
        return NGX_CONF_ERROR;
    }

    llcf->open_file_cache = ngx_open_file_cache_init(cf->pool, max, inactive);

    if (llcf->open_file_cache) {

        llcf->open_file_cache_valid = valid;
        llcf->open_file_cache_min_uses = min_uses;

        return NGX_CONF_OK;
    }

    return NGX_CONF_ERROR;
}


static ngx_int_t
ngx_http_log_init(ngx_conf_t *cf)
{
    ngx_str_t                  *value;
    ngx_array_t                 a;
    ngx_http_handler_pt        *h;
    ngx_http_log_fmt_t         *fmt;
    ngx_http_log_main_conf_t   *lmcf;
    ngx_http_core_main_conf_t  *cmcf;

    lmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_log_module);

    if (lmcf->combined_used) {
        if (ngx_array_init(&a, cf->pool, 1, sizeof(ngx_str_t)) != NGX_OK) {
            return NGX_ERROR;
        }

        value = ngx_array_push(&a);
        if (value == NULL) {
            return NGX_ERROR;
        }

```

```
1774            *value = ngx_http_combined_fmt;
1775            fmt = lmcf->formats.elts;
1776
1777            if (ngx_http_log_compile_format(cf, NULL, fmt->ops, &a, 0)
1778                != NGX_CONF_OK)
1779            {
1780                return NGX_ERROR;
1781            }
1782        }
1783
1784        cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
1785
1786        h = ngx_array_push(&cmcf->phases[NGX_HTTP_LOG_PHASE].handlers);
1787        if (h == NULL) {
1788            return NGX_ERROR;
1789        }
1790
1791        *h = ngx_http_log_handler;
1792
1793        return NGX_OK;
1794    }
```

One Level Up Top Level

src/http/modules/ngx_http_map_module.c - nginx-1.7.10

Global variables defined

- ngx_http_map_commands
- ngx_http_map_module
- ngx_http_map_module_ctx

Data types defined

- ngx_http_map_conf_ctx_t
- ngx_http_map_conf_t
- ngx_http_map_ctx_t

Functions defined

- ngx_http_map
- ngx_http_map_block
- ngx_http_map_cmp_dns_wildcards
- ngx_http_map_create_conf
- ngx_http_map_variable

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_uint_t                  hash_max_size;
15      ngx_uint_t                  hash_bucket_size;
16  } ngx_http_map_conf_t;
17
18
19  typedef struct {
20      ngx_hash_keys_arrays_t      keys;
21
22      ngx_array_t                *values_hash;
23      ngx_array_t                 var_values;
24  #if (NGX_PCRE)
25      ngx_array_t                 regexes;
26  #endif
27
28      ngx_http_variable_value_t  *default_value;
29      ngx_conf_t                 *cf;
30      ngx_uint_t                  hostnames;      /* unsigned  hostnames:1 */
31  } ngx_http_map_conf_ctx_t;
32
33
```

```
typedef struct {
    ngx_http_map_t              map;
    ngx_http_complex_value_t    value;
    ngx_http_variable_value_t  *default_value;
    ngx_uint_t                  hostnames;      /* unsigned  hostnames:1 */
} ngx_http_map_ctx_t;


static int ngx_libc_cdecl ngx_http_map_cmp_dns_wildcards(const void *one,
    const void *two);
static void *ngx_http_map_create_conf(ngx_conf_t *cf);
static char *ngx_http_map_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static char *ngx_http_map(ngx_conf_t *cf, ngx_command_t *dummy, void *conf);


static ngx_command_t  ngx_http_map_commands[] = {

    { ngx_string("map"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_TAKE2,
      ngx_http_map_block,
      NGX_HTTP_MAIN_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("map_hash_max_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_map_conf_t, hash_max_size),
      NULL },

    { ngx_string("map_hash_bucket_size"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_map_conf_t, hash_bucket_size),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_map_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_map_create_conf,              /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_map_module = {
    NGX_MODULE_V1,
    &ngx_http_map_module_ctx,              /* module context */
    ngx_http_map_commands,                 /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
 };


 static ngx_int_t
 ngx_http_map_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
     uintptr_t data)
```

```
{
    ngx_http_map_ctx_t  *map = (ngx_http_map_ctx_t *) data;

    ngx_str_t                   val;
    ngx_http_variable_value_t  *value;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http map started");

    if (ngx_http_complex_value(r, &map->value, &val) != NGX_OK) {
        return NGX_ERROR;
    }

    if (map->hostnames && val.len > 0 && val.data[val.len - 1] == '.') {
        val.len--;
    }

    value = ngx_http_map_find(r, &map->map, &val);

    if (value == NULL) {
        value = map->default_value;
    }

    if (!value->valid) {
        value = ngx_http_get_flushed_variable(r, (uintptr_t) value->data);

        if (value == NULL || value->not_found) {
            value = &ngx_http_variable_null_value;
        }
    }

    *v = *value;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http map: \"%v\" \"%v\"", &val, v);

    return NGX_OK;
}


static void *
ngx_http_map_create_conf(ngx_conf_t *cf)
{
    ngx_http_map_conf_t  *mcf;

    mcf = ngx_palloc(cf->pool, sizeof(ngx_http_map_conf_t));
    if (mcf == NULL) {
        return NULL;
    }

    mcf->hash_max_size = NGX_CONF_UNSET_UINT;
    mcf->hash_bucket_size = NGX_CONF_UNSET_UINT;

    return mcf;
}


static char *
ngx_http_map_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_map_conf_t  *mcf = conf;

    char                            *rv;
    ngx_str_t                       *value, name;
    ngx_conf_t                       save;
    ngx_pool_t                      *pool;
    ngx_hash_init_t                  hash;
    ngx_http_map_ctx_t              *map;
    ngx_http_variable_t             *var;
    ngx_http_map_conf_ctx_t          ctx;
    ngx_http_compile_complex_value_t  ccv;

    if (mcf->hash_max_size == NGX_CONF_UNSET_UINT) {
        mcf->hash_max_size = 2048;
    }

```

```
    if (mcf->hash_bucket_size == NGX_CONF_UNSET_UINT) {
        mcf->hash_bucket_size = ngx_cacheline_size;

    } else {
        mcf->hash_bucket_size = ngx_align(mcf->hash_bucket_size,
                                          ngx_cacheline_size);
    }

    map = ngx_pcalloc(cf->pool, sizeof(ngx_http_map_ctx_t));
    if (map == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &map->value;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    name = value[2];

    if (name.data[0] != '$') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid variable name \"%V\"", &name);
        return NGX_CONF_ERROR;
    }

    name.len--;
    name.data++;

    var = ngx_http_add_variable(cf, &name, NGX_HTTP_VAR_CHANGEABLE);
    if (var == NULL) {
        return NGX_CONF_ERROR;
    }

    var->get_handler = ngx_http_map_variable;
    var->data = (uintptr_t) map;

    pool = ngx_create_pool(NGX_DEFAULT_POOL_SIZE, cf->log);
    if (pool == NULL) {
        return NGX_CONF_ERROR;
    }

    ctx.keys.pool = cf->pool;
    ctx.keys.temp_pool = pool;

    if (ngx_hash_keys_array_init(&ctx.keys, NGX_HASH_LARGE) != NGX_OK) {
        ngx_destroy_pool(pool);
        return NGX_CONF_ERROR;
    }

    ctx.values_hash = ngx_pcalloc(pool, sizeof(ngx_array_t) * ctx.keys.hsize);
    if (ctx.values_hash == NULL) {
        ngx_destroy_pool(pool);
        return NGX_CONF_ERROR;
    }

    if (ngx_array_init(&ctx.var_values, cf->pool, 2,
                       sizeof(ngx_http_variable_value_t))
        != NGX_OK)
    {
        ngx_destroy_pool(pool);
        return NGX_CONF_ERROR;
    }

#if (NGX_PCRE)
    if (ngx_array_init(&ctx.regexes, cf->pool, 2, sizeof(ngx_http_map_regex_t))
        != NGX_OK)
    {
        ngx_destroy_pool(pool);
```

```
        return NGX_CONF_ERROR;
    }
#endif

    ctx.default_value = NULL;
    ctx.cf = &save;
    ctx.hostnames = 0;

    save = *cf;
    cf->pool = pool;
    cf->ctx = &ctx;
    cf->handler = ngx_http_map;
    cf->handler_conf = conf;

    rv = ngx_conf_parse(cf, NULL);

    *cf = save;

    if (rv != NGX_CONF_OK) {
        ngx_destroy_pool(pool);
        return rv;
    }

    map->default_value = ctx.default_value ? ctx.default_value:
                                             &ngx_http_variable_null_value;

    map->hostnames = ctx.hostnames;

    hash.key = ngx_hash_key_lc;
    hash.max_size = mcf->hash_max_size;
    hash.bucket_size = mcf->hash_bucket_size;
    hash.name = "map_hash";
    hash.pool = cf->pool;

    if (ctx.keys.keys.nelts) {
        hash.hash = &map->map.hash.hash;
        hash.temp_pool = NULL;

        if (ngx_hash_init(&hash, ctx.keys.keys.elts, ctx.keys.keys.nelts)
            != NGX_OK)
        {
            ngx_destroy_pool(pool);
            return NGX_CONF_ERROR;
        }
    }

    if (ctx.keys.dns_wc_head.nelts) {

        ngx_qsort(ctx.keys.dns_wc_head.elts,
                  (size_t) ctx.keys.dns_wc_head.nelts,
                  sizeof(ngx_hash_key_t), ngx_http_map_cmp_dns_wildcards);

        hash.hash = NULL;
        hash.temp_pool = pool;

        if (ngx_hash_wildcard_init(&hash, ctx.keys.dns_wc_head.elts,
                                   ctx.keys.dns_wc_head.nelts)
            != NGX_OK)
        {
            ngx_destroy_pool(pool);
            return NGX_CONF_ERROR;
        }

        map->map.hash.wc_head = (ngx_hash_wildcard_t *) hash.hash;
    }

    if (ctx.keys.dns_wc_tail.nelts) {

        ngx_qsort(ctx.keys.dns_wc_tail.elts,
                  (size_t) ctx.keys.dns_wc_tail.nelts,
                  sizeof(ngx_hash_key_t), ngx_http_map_cmp_dns_wildcards);

        hash.hash = NULL;
        hash.temp_pool = pool;

        if (ngx_hash_wildcard_init(&hash, ctx.keys.dns_wc_tail.elts,
```

```
                                  ctx.keys.dns_wc_tail.nelts)
            != NGX_OK)
        {
            ngx_destroy_pool(pool);
            return NGX_CONF_ERROR;
        }

        map->map.hash.wc_tail = (ngx_hash_wildcard_t *) hash.hash;
    }

#if (NGX_PCRE)

    if (ctx.regexes.nelts) {
        map->map.regex = ctx.regexes.elts;
        map->map.nregex = ctx.regexes.nelts;
    }

#endif

    ngx_destroy_pool(pool);

    return rv;
}


static int ngx_libc_cdecl
ngx_http_map_cmp_dns_wildcards(const void *one, const void *two)
{
    ngx_hash_key_t  *first, *second;

    first = (ngx_hash_key_t *) one;
    second = (ngx_hash_key_t *) two;

    return ngx_dns_strcmp(first->key.data, second->key.data);
}


static char *
ngx_http_map(ngx_conf_t *cf, ngx_command_t *dummy, void *conf)
{
    ngx_int_t                   rc, index;
    ngx_str_t                  *value, name;
    ngx_uint_t                  i, key;
    ngx_http_map_conf_ctx_t    *ctx;
    ngx_http_variable_value_t  *var, **vp;

    ctx = cf->ctx;

    value = cf->args->elts;

    if (cf->args->nelts == 1
        && ngx_strcmp(value[0].data, "hostnames") == 0)
    {
        ctx->hostnames = 1;
        return NGX_CONF_OK;

    } else if (cf->args->nelts != 2) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid number of the map parameters");
        return NGX_CONF_ERROR;
    }

    if (ngx_strcmp(value[0].data, "include") == 0) {
        return ngx_conf_include(cf, dummy, conf);
    }

    if (value[1].data[0] == '$') {
        name = value[1];
        name.len--;
        name.data++;

        index = ngx_http_get_variable_index(ctx->cf, &name);
        if (index == NGX_ERROR) {
            return NGX_CONF_ERROR;
        }

```

```
414         var = ctx->var_values.elts;
415
416         for (i = 0; i < ctx->var_values.nelts; i++) {
417             if (index == (intptr_t) var[i].data) {
418                 var = &var[i];
419                 goto found;
420             }
421         }
422
423         var = ngx_array_push(&ctx->var_values);
424         if (var == NULL) {
425             return NGX_CONF_ERROR;
426         }
427
428         var->valid = 0;
429         var->no_cacheable = 0;
430         var->not_found = 0;
431         var->len = 0;
432         var->data = (u_char *) (intptr_t) index;
433
434         goto found;
435     }
436
437     key = 0;
438
439     for (i = 0; i < value[1].len; i++) {
440         key = ngx_hash(key, value[1].data[i]);
441     }
442
443     key %= ctx->keys.hsize;
444
445     vp = ctx->values_hash[key].elts;
446
447     if (vp) {
448         for (i = 0; i < ctx->values_hash[key].nelts; i++) {
449             if (value[1].len != (size_t) vp[i]->len) {
450                 continue;
451             }
452
453             if (ngx_strncmp(value[1].data, vp[i]->data, value[1].len) == 0) {
454                 var = vp[i];
455                 goto found;
456             }
457         }
458
459     } else {
460         if (ngx_array_init(&ctx->values_hash[key], cf->pool, 4,
461                            sizeof(ngx_http_variable_value_t *))
462             != NGX_OK)
463         {
464             return NGX_CONF_ERROR;
465         }
466     }
467
468     var = ngx_palloc(ctx->keys.pool, sizeof(ngx_http_variable_value_t));
469     if (var == NULL) {
470         return NGX_CONF_ERROR;
471     }
472
473     var->len = value[1].len;
474     var->data = ngx_pstrdup(ctx->keys.pool, &value[1]);
475     if (var->data == NULL) {
476         return NGX_CONF_ERROR;
477     }
478
479     var->valid = 1;
480     var->no_cacheable = 0;
481     var->not_found = 0;
482
483     vp = ngx_array_push(&ctx->values_hash[key]);
484     if (vp == NULL) {
485         return NGX_CONF_ERROR;
486     }
487
488     *vp = var;
489
```

```
found:

    if (ngx_strcmp(value[0].data, "default") == 0) {

        if (ctx->default_value) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "duplicate default map parameter");
            return NGX_CONF_ERROR;
        }

        ctx->default_value = var;

        return NGX_CONF_OK;
    }

#if (NGX_PCRE)

    if (value[0].len && value[0].data[0] == '~') {
        ngx_regex_compile_t    rc;
        ngx_http_map_regex_t  *regex;
        u_char                 errstr[NGX_MAX_CONF_ERRSTR];

        regex = ngx_array_push(&ctx->regexes);
        if (regex == NULL) {
            return NGX_CONF_ERROR;
        }

        value[0].len--;
        value[0].data++;

        ngx_memzero(&rc, sizeof(ngx_regex_compile_t));

        if (value[0].data[0] == '*') {
            value[0].len--;
            value[0].data++;
            rc.options = NGX_REGEX_CASELESS;
        }

        rc.pattern = value[0];
        rc.err.len = NGX_MAX_CONF_ERRSTR;
        rc.err.data = errstr;

        regex->regex = ngx_http_regex_compile(ctx->cf, &rc);
        if (regex->regex == NULL) {
            return NGX_CONF_ERROR;
        }

        regex->value = var;

        return NGX_CONF_OK;
    }

#endif

    if (value[0].len && value[0].data[0] == '\\') {
        value[0].len--;
        value[0].data++;
    }

    rc = ngx_hash_add_key(&ctx->keys, &value[0], var,
                          (ctx->hostnames) ? NGX_HASH_WILDCARD_KEY : 0);

    if (rc == NGX_OK) {
        return NGX_CONF_OK;
    }

    if (rc == NGX_DECLINED) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid hostname or wildcard \"%V\"", &value[0]);
    }

    if (rc == NGX_BUSY) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "conflicting parameter \"%V\"", &value[0]);
    }

```

```
566     return NGX_CONF_ERROR;
567 }
```

One Level Up Top Level

src/http/modules/ngx_http_memcached_module.c - nginx-1.7.10

Global variables defined

- ngx_http_memcached_commands
- ngx_http_memcached_end
- ngx_http_memcached_key
- ngx_http_memcached_module
- ngx_http_memcached_module_ctx
- ngx_http_memcached_next_upstream_masks

Data types defined

- ngx_http_memcached_ctx_t
- ngx_http_memcached_loc_conf_t

Functions defined

- ngx_http_memcached_abort_request
- ngx_http_memcached_create_loc_conf
- ngx_http_memcached_create_request
- ngx_http_memcached_filter
- ngx_http_memcached_filter_init
- ngx_http_memcached_finalize_request
- ngx_http_memcached_handler
- ngx_http_memcached_merge_loc_conf
- ngx_http_memcached_pass
- ngx_http_memcached_process_header
- ngx_http_memcached_reinit_request

Macros defined

- NGX_HTTP_MEMCACHED_END

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
```

```c
#include <ngx_http.h>


typedef struct {
    ngx_http_upstream_conf_t   upstream;
    ngx_int_t                  index;
    ngx_uint_t                 gzip_flag;
} ngx_http_memcached_loc_conf_t;


typedef struct {
    size_t                     rest;
    ngx_http_request_t        *request;
    ngx_str_t                  key;
} ngx_http_memcached_ctx_t;


static ngx_int_t ngx_http_memcached_create_request(ngx_http_request_t *r);
static ngx_int_t ngx_http_memcached_reinit_request(ngx_http_request_t *r);
static ngx_int_t ngx_http_memcached_process_header(ngx_http_request_t *r);
static ngx_int_t ngx_http_memcached_filter_init(void *data);
static ngx_int_t ngx_http_memcached_filter(void *data, ssize_t bytes);
static void ngx_http_memcached_abort_request(ngx_http_request_t *r);
static void ngx_http_memcached_finalize_request(ngx_http_request_t *r,
    ngx_int_t rc);

static void *ngx_http_memcached_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_memcached_merge_loc_conf(ngx_conf_t *cf,
    void *parent, void *child);

static char *ngx_http_memcached_pass(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


static ngx_conf_bitmask_t  ngx_http_memcached_next_upstream_masks[] = {
    { ngx_string("error"), NGX_HTTP_UPSTREAM_FT_ERROR },
    { ngx_string("timeout"), NGX_HTTP_UPSTREAM_FT_TIMEOUT },
    { ngx_string("invalid_response"), NGX_HTTP_UPSTREAM_FT_INVALID_HEADER },
    { ngx_string("not_found"), NGX_HTTP_UPSTREAM_FT_HTTP_404 },
    { ngx_string("off"), NGX_HTTP_UPSTREAM_FT_OFF },
    { ngx_null_string, 0 }
};


static ngx_command_t  ngx_http_memcached_commands[] = {

    { ngx_string("memcached_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_memcached_pass,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("memcached_bind"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_upstream_bind_set_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.local),
      NULL },

    { ngx_string("memcached_connect_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.connect_timeout),
      NULL },

    { ngx_string("memcached_send_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.send_timeout),
      NULL },

    { ngx_string("memcached_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
```

```
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.buffer_size),
      NULL },

    { ngx_string("memcached_read_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.read_timeout),
      NULL },

    { ngx_string("memcached_next_upstream"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.next_upstream),
      &ngx_http_memcached_next_upstream_masks },

    { ngx_string("memcached_next_upstream_tries"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.next_upstream_tries),
      NULL },

    { ngx_string("memcached_next_upstream_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, upstream.next_upstream_timeout),
      NULL },

    { ngx_string("memcached_gzip_flag"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_memcached_loc_conf_t, gzip_flag),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_memcached_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_memcached_create_loc_conf,    /* create location configuration */
    ngx_http_memcached_merge_loc_conf      /* merge location configuration */
};


ngx_module_t  ngx_http_memcached_module = {
    NGX_MODULE_V1,
    &ngx_http_memcached_module_ctx,        /* module context */
    ngx_http_memcached_commands,           /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t  ngx_http_memcached_key = ngx_string("memcached_key");
```

```
162
163
164 #define NGX_HTTP_MEMCACHED_END   (sizeof(ngx_http_memcached_end) - 1)
165 static u_char  ngx_http_memcached_end[] = CRLF "END" CRLF;
166
167
168 static ngx_int_t
169 ngx_http_memcached_handler(ngx_http_request_t *r)
170 {
171     ngx_int_t                      rc;
172     ngx_http_upstream_t           *u;
173     ngx_http_memcached_ctx_t      *ctx;
174     ngx_http_memcached_loc_conf_t  *mlcf;
175
176     if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD))) {
177         return NGX_HTTP_NOT_ALLOWED;
178     }
179
180     rc = ngx_http_discard_request_body(r);
181
182     if (rc != NGX_OK) {
183         return rc;
184     }
185
186     if (ngx_http_set_content_type(r) != NGX_OK) {
187         return NGX_HTTP_INTERNAL_SERVER_ERROR;
188     }
189
190     if (ngx_http_upstream_create(r) != NGX_OK) {
191         return NGX_HTTP_INTERNAL_SERVER_ERROR;
192     }
193
194     u = r->upstream;
195
196     ngx_str_set(&u->schema, "memcached://");
197     u->output.tag = (ngx_buf_tag_t) &ngx_http_memcached_module;
198
199     mlcf = ngx_http_get_module_loc_conf(r, ngx_http_memcached_module);
200
201     u->conf = &mlcf->upstream;
202
203     u->create_request = ngx_http_memcached_create_request;
204     u->reinit_request = ngx_http_memcached_reinit_request;
205     u->process_header = ngx_http_memcached_process_header;
206     u->abort_request = ngx_http_memcached_abort_request;
207     u->finalize_request = ngx_http_memcached_finalize_request;
208
209     ctx = ngx_palloc(r->pool, sizeof(ngx_http_memcached_ctx_t));
210     if (ctx == NULL) {
211         return NGX_HTTP_INTERNAL_SERVER_ERROR;
212     }
213
214     ctx->request = r;
215
216     ngx_http_set_ctx(r, ctx, ngx_http_memcached_module);
217
218     u->input_filter_init = ngx_http_memcached_filter_init;
219     u->input_filter = ngx_http_memcached_filter;
220     u->input_filter_ctx = ctx;
221
222     r->main->count++;
223
224     ngx_http_upstream_init(r);
225
226     return NGX_DONE;
227 }
228
229
230 static ngx_int_t
231 ngx_http_memcached_create_request(ngx_http_request_t *r)
232 {
233     size_t                         len;
234     uintptr_t                      escape;
235     ngx_buf_t                     *b;
236     ngx_chain_t                   *cl;
237     ngx_http_memcached_ctx_t      *ctx;
```

```
    ngx_http_variable_value_t      *vv;
    ngx_http_memcached_loc_conf_t  *mlcf;

    mlcf = ngx_http_get_module_loc_conf(r, ngx_http_memcached_module);

    vv = ngx_http_get_indexed_variable(r, mlcf->index);

    if (vv == NULL || vv->not_found || vv->len == 0) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "the \"$memcached_key\" variable is not set");
        return NGX_ERROR;
    }

    escape = 2 * ngx_escape_uri(NULL, vv->data, vv->len, NGX_ESCAPE_MEMCACHED);

    len = sizeof("get ") - 1 + vv->len + escape + sizeof(CRLF) - 1;

    b = ngx_create_temp_buf(r->pool, len);
    if (b == NULL) {
        return NGX_ERROR;
    }

    cl = ngx_alloc_chain_link(r->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    cl->buf = b;
    cl->next = NULL;

    r->upstream->request_bufs = cl;

    *b->last++ = 'g'; *b->last++ = 'e'; *b->last++ = 't'; *b->last++ = ' ';

    ctx = ngx_http_get_module_ctx(r, ngx_http_memcached_module);

    ctx->key.data = b->last;

    if (escape == 0) {
        b->last = ngx_copy(b->last, vv->data, vv->len);

    } else {
        b->last = (u_char *) ngx_escape_uri(b->last, vv->data, vv->len,
                                            NGX_ESCAPE_MEMCACHED);
    }

    ctx->key.len = b->last - ctx->key.data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http memcached request: \"%V\"", &ctx->key);

    *b->last++ = CR; *b->last++ = LF;

    return NGX_OK;
}


static ngx_int_t
ngx_http_memcached_reinit_request(ngx_http_request_t *r)
{
    return NGX_OK;
}


static ngx_int_t
ngx_http_memcached_process_header(ngx_http_request_t *r)
{
    u_char                         *p, *start;
    ngx_str_t                       line;
    ngx_uint_t                      flags;
    ngx_table_elt_t                *h;
    ngx_http_upstream_t            *u;
    ngx_http_memcached_ctx_t       *ctx;
    ngx_http_memcached_loc_conf_t  *mlcf;

    u = r->upstream;
```

```
314
315     for (p = u->buffer.pos; p < u->buffer.last; p++) {
316         if (*p == LF) {
317             goto found;
318         }
319     }
320
321     return NGX_AGAIN;
322
323 found:
324
325     line.data = u->buffer.pos;
326     line.len = p - u->buffer.pos;
327
328     if (line.len == 0 || *(p - 1) != CR) {
329         goto no_valid;
330     }
331
332     *p = '\0';
333     line.len--;
334
335     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
336                    "memcached: \"%V\"", &line);
337
338     p = u->buffer.pos;
339
340     ctx = ngx_http_get_module_ctx(r, ngx_http_memcached_module);
341     mlcf = ngx_http_get_module_loc_conf(r, ngx_http_memcached_module);
342
343     if (ngx_strncmp(p, "VALUE ", sizeof("VALUE ") - 1) == 0) {
344
345         p += sizeof("VALUE ") - 1;
346
347         if (ngx_strncmp(p, ctx->key.data, ctx->key.len) != 0) {
348             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
349                           "memcached sent invalid key in response \"%V\" "
350                           "for key \"%V\"",
351                           &line, &ctx->key);
352
353             return NGX_HTTP_UPSTREAM_INVALID_HEADER;
354         }
355
356         p += ctx->key.len;
357
358         if (*p++ != ' ') {
359             goto no_valid;
360         }
361
362         /* flags */
363
364         start = p;
365
366         while (*p) {
367             if (*p++ == ' ') {
368                 if (mlcf->gzip_flag) {
369                     goto flags;
370                 } else {
371                     goto length;
372                 }
373             }
374         }
375
376         goto no_valid;
377
378     flags:
379
380         flags = ngx_atoi(start, p - start - 1);
381
382         if (flags == (ngx_uint_t) NGX_ERROR) {
383             ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
384                           "memcached sent invalid flags in response \"%V\" "
385                           "for key \"%V\"",
386                           &line, &ctx->key);
387             return NGX_HTTP_UPSTREAM_INVALID_HEADER;
388         }
389
```

```
        if (flags & mlcf->gzip_flag) {
            h = ngx_list_push(&r->headers_out.headers);
            if (h == NULL) {
                return NGX_ERROR;
            }

            h->hash = 1;
            ngx_str_set(&h->key, "Content-Encoding");
            ngx_str_set(&h->value, "gzip");
            r->headers_out.content_encoding = h;
        }

    length:

        start = p;
        p = line.data + line.len;

        u->headers_in.content_length_n = ngx_atoof(start, p - start);
        if (u->headers_in.content_length_n == NGX_ERROR) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "memcached sent invalid length in response \"%V\" "
                          "for key \"%V\"",
                          &line, &ctx->key);
            return NGX_HTTP_UPSTREAM_INVALID_HEADER;
        }

        u->headers_in.status_n = 200;
        u->state->status = 200;
        u->buffer.pos = p + sizeof(CRLF) - 1;

        return NGX_OK;
    }

    if (ngx_strcmp(p, "END\x0d") == 0) {
        ngx_log_error(NGX_LOG_INFO, r->connection->log, 0,
                      "key: \"%V\" was not found by memcached", &ctx->key);

        u->headers_in.content_length_n = 0;
        u->headers_in.status_n = 404;
        u->state->status = 404;
        u->buffer.pos = p + sizeof("END" CRLF) - 1;
        u->keepalive = 1;

        return NGX_OK;
    }

no_valid:

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "memcached sent invalid response: \"%V\"", &line);

    return NGX_HTTP_UPSTREAM_INVALID_HEADER;
}


static ngx_int_t
ngx_http_memcached_filter_init(void *data)
{
    ngx_http_memcached_ctx_t  *ctx = data;

    ngx_http_upstream_t  *u;

    u = ctx->request->upstream;

    if (u->headers_in.status_n != 404) {
        u->length = u->headers_in.content_length_n + NGX_HTTP_MEMCACHED_END;
        ctx->rest = NGX_HTTP_MEMCACHED_END;

    } else {
        u->length = 0;
    }

    return NGX_OK;
}
```

```
466 static ngx_int_t
467 ngx_http_memcached_filter(void *data, ssize_t bytes)
468 {
469     ngx_http_memcached_ctx_t  *ctx = data;
470
471     u_char                    *last;
472     ngx_buf_t                 *b;
473     ngx_chain_t               *cl, **ll;
474     ngx_http_upstream_t       *u;
475
476     u = ctx->request->upstream;
477     b = &u->buffer;
478
479     if (u->length == (ssize_t) ctx->rest) {
480
481         if (ngx_strncmp(b->last,
482                         ngx_http_memcached_end + NGX_HTTP_MEMCACHED_END - ctx->rest,
483                         bytes)
484             != 0)
485         {
486             ngx_log_error(NGX_LOG_ERR, ctx->request->connection->log, 0,
487                           "memcached sent invalid trailer");
488
489             u->length = 0;
490             ctx->rest = 0;
491
492             return NGX_OK;
493         }
494
495         u->length -= bytes;
496         ctx->rest -= bytes;
497
498         if (u->length == 0) {
499             u->keepalive = 1;
500         }
501
502         return NGX_OK;
503     }
504
505     for (cl = u->out_bufs, ll = &u->out_bufs; cl; cl = cl->next) {
506         ll = &cl->next;
507     }
508
509     cl = ngx_chain_get_free_buf(ctx->request->pool, &u->free_bufs);
510     if (cl == NULL) {
511         return NGX_ERROR;
512     }
513
514     cl->buf->flush = 1;
515     cl->buf->memory = 1;
516
517     *ll = cl;
518
519     last = b->last;
520     cl->buf->pos = last;
521     b->last += bytes;
522     cl->buf->last = b->last;
523     cl->buf->tag = u->output.tag;
524
525     ngx_log_debug4(NGX_LOG_DEBUG_HTTP, ctx->request->connection->log, 0,
526                    "memcached filter bytes:%z size:%z length:%z rest:%z",
527                    bytes, b->last - b->pos, u->length, ctx->rest);
528
529     if (bytes <= (ssize_t) (u->length - NGX_HTTP_MEMCACHED_END)) {
530         u->length -= bytes;
531         return NGX_OK;
532     }
533
534     last += (size_t) (u->length - NGX_HTTP_MEMCACHED_END);
535
536     if (ngx_strncmp(last, ngx_http_memcached_end, b->last - last) != 0) {
537         ngx_log_error(NGX_LOG_ERR, ctx->request->connection->log, 0,
538                       "memcached sent invalid trailer");
539
540         b->last = last;
541         cl->buf->last = last;
```

```
        u->length = 0;
        ctx->rest = 0;

        return NGX_OK;
    }

    ctx->rest -= b->last - last;
    b->last = last;
    cl->buf->last = last;
    u->length = ctx->rest;

    if (u->length == 0) {
        u->keepalive = 1;
    }

    return NGX_OK;
}


static void
ngx_http_memcached_abort_request(ngx_http_request_t *r)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "abort http memcached request");
    return;
}


static void
ngx_http_memcached_finalize_request(ngx_http_request_t *r, ngx_int_t rc)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "finalize http memcached request");
    return;
}


static void *
ngx_http_memcached_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_memcached_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_memcached_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->upstream.bufs.num = 0;
     *     conf->upstream.next_upstream = 0;
     *     conf->upstream.temp_path = NULL;
     *     conf->upstream.uri = { 0, NULL };
     *     conf->upstream.location = NULL;
     */

    conf->upstream.local = NGX_CONF_UNSET_PTR;
    conf->upstream.next_upstream_tries = NGX_CONF_UNSET_UINT;
    conf->upstream.connect_timeout = NGX_CONF_UNSET_MSEC;
    conf->upstream.send_timeout = NGX_CONF_UNSET_MSEC;
    conf->upstream.read_timeout = NGX_CONF_UNSET_MSEC;
    conf->upstream.next_upstream_timeout = NGX_CONF_UNSET_MSEC;

    conf->upstream.buffer_size = NGX_CONF_UNSET_SIZE;

    /* the hardcoded values */
    conf->upstream.cyclic_temp_file = 0;
    conf->upstream.buffering = 0;
    conf->upstream.ignore_client_abort = 0;
    conf->upstream.send_lowat = 0;
    conf->upstream.bufs.num = 0;
    conf->upstream.busy_buffers_size = 0;
    conf->upstream.max_temp_file_size = 0;
    conf->upstream.temp_file_write_size = 0;
    conf->upstream.intercept_errors = 1;
```

```
618     conf->upstream.intercept_404 = 1;
619     conf->upstream.pass_request_headers = 0;
620     conf->upstream.pass_request_body = 0;
621
622     conf->index = NGX_CONF_UNSET;
623     conf->gzip_flag = NGX_CONF_UNSET_UINT;
624
625     return conf;
626 }
627
628
629 static char *
630 ngx_http_memcached_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
631 {
632     ngx_http_memcached_loc_conf_t *prev = parent;
633     ngx_http_memcached_loc_conf_t *conf = child;
634
635     ngx_conf_merge_ptr_value(conf->upstream.local,
636                              prev->upstream.local, NULL);
637
638     ngx_conf_merge_uint_value(conf->upstream.next_upstream_tries,
639                              prev->upstream.next_upstream_tries, 0);
640
641     ngx_conf_merge_msec_value(conf->upstream.connect_timeout,
642                              prev->upstream.connect_timeout, 60000);
643
644     ngx_conf_merge_msec_value(conf->upstream.send_timeout,
645                              prev->upstream.send_timeout, 60000);
646
647     ngx_conf_merge_msec_value(conf->upstream.read_timeout,
648                              prev->upstream.read_timeout, 60000);
649
650     ngx_conf_merge_msec_value(conf->upstream.next_upstream_timeout,
651                              prev->upstream.next_upstream_timeout, 0);
652
653     ngx_conf_merge_size_value(conf->upstream.buffer_size,
654                              prev->upstream.buffer_size,
655                              (size_t) ngx_pagesize);
656
657     ngx_conf_merge_bitmask_value(conf->upstream.next_upstream,
658                              prev->upstream.next_upstream,
659                              (NGX_CONF_BITMASK_SET
660                               |NGX_HTTP_UPSTREAM_FT_ERROR
661                               |NGX_HTTP_UPSTREAM_FT_TIMEOUT));
662
663     if (conf->upstream.next_upstream & NGX_HTTP_UPSTREAM_FT_OFF) {
664         conf->upstream.next_upstream = NGX_CONF_BITMASK_SET
665                                        |NGX_HTTP_UPSTREAM_FT_OFF;
666     }
667
668     if (conf->upstream.upstream == NULL) {
669         conf->upstream.upstream = prev->upstream.upstream;
670     }
671
672     if (conf->index == NGX_CONF_UNSET) {
673         conf->index = prev->index;
674     }
675
676     ngx_conf_merge_uint_value(conf->gzip_flag, prev->gzip_flag, 0);
677
678     return NGX_CONF_OK;
679 }
680
681
682 static char *
683 ngx_http_memcached_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
684 {
685     ngx_http_memcached_loc_conf_t *mlcf = conf;
686
687     ngx_str_t                 *value;
688     ngx_url_t                  u;
689     ngx_http_core_loc_conf_t  *clcf;
690
691     if (mlcf->upstream.upstream) {
692         return "is duplicate";
693     }
```

```
694
695     value = cf->args->elts;
696
697     ngx_memzero(&u, sizeof(ngx_url_t));
698
699     u.url = value[1];
700     u.no_resolve = 1;
701
702     mlcf->upstream.upstream = ngx_http_upstream_add(cf, &u, 0);
703     if (mlcf->upstream.upstream == NULL) {
704         return NGX_CONF_ERROR;
705     }
706
707     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
708
709     clcf->handler = ngx_http_memcached_handler;
710
711     if (clcf->name.data[clcf->name.len - 1] == '/') {
712         clcf->auto_redirect = 1;
713     }
714
715     mlcf->index = ngx_http_get_variable_index(cf, &ngx_http_memcached_key);
716
717     if (mlcf->index == NGX_ERROR) {
718         return NGX_CONF_ERROR;
719     }
720
721     return NGX_CONF_OK;
722 }
```

One Level Up Top Level

src/http/modules/ngx_http_mp4_module.c - nginx-1.7.10

Global variables defined

- ngx_http_mp4_atoms
- ngx_http_mp4_commands
- ngx_http_mp4_mdia_atoms
- ngx_http_mp4_minf_atoms
- ngx_http_mp4_module
- ngx_http_mp4_module_ctx
- ngx_http_mp4_moov_atoms
- ngx_http_mp4_stbl_atoms
- ngx_http_mp4_trak_atoms

Data types defined

- ngx_http_mp4_atom_handler_t
- ngx_http_mp4_conf_t
- ngx_http_mp4_file_t
- ngx_http_mp4_stss_atom_t
- ngx_http_mp4_trak_t
- ngx_mp4_atom_header64_t
- ngx_mp4_atom_header_t
- ngx_mp4_co64_atom_t
- ngx_mp4_ctts_atom_t
- ngx_mp4_ctts_entry_t
- ngx_mp4_mdhd64_atom_t
- ngx_mp4_mdhd_atom_t
- ngx_mp4_mvhd64_atom_t
- ngx_mp4_mvhd_atom_t
- ngx_mp4_stco_atom_t
- ngx_mp4_stsc_atom_t
- ngx_mp4_stsc_entry_t
- ngx_mp4_stsd_atom_t
- ngx_mp4_stsz_atom_t

- ngx_mp4_stts_atom_t
- ngx_mp4_stts_entry_t
- ngx_mp4_tkhd64_atom_t
- ngx_mp4_tkhd_atom_t

Functions defined

- ngx_http_mp4
- ngx_http_mp4_adjust_co64_atom
- ngx_http_mp4_adjust_stco_atom
- ngx_http_mp4_create_conf
- ngx_http_mp4_crop_ctts_data
- ngx_http_mp4_crop_stsc_data
- ngx_http_mp4_crop_stss_data
- ngx_http_mp4_crop_stts_data
- ngx_http_mp4_handler
- ngx_http_mp4_merge_conf
- ngx_http_mp4_process
- ngx_http_mp4_read
- ngx_http_mp4_read_atom
- ngx_http_mp4_read_cmov_atom
- ngx_http_mp4_read_co64_atom
- ngx_http_mp4_read_ctts_atom
- ngx_http_mp4_read_dinf_atom
- ngx_http_mp4_read_ftyp_atom
- ngx_http_mp4_read_hdlr_atom
- ngx_http_mp4_read_mdat_atom
- ngx_http_mp4_read_mdhd_atom
- ngx_http_mp4_read_mdia_atom
- ngx_http_mp4_read_minf_atom
- ngx_http_mp4_read_moov_atom
- ngx_http_mp4_read_mvhd_atom
- ngx_http_mp4_read_smhd_atom
- ngx_http_mp4_read_stbl_atom
- ngx_http_mp4_read_stco_atom

- ngx_http_mp4_read_stsc_atom
- ngx_http_mp4_read_stsd_atom
- ngx_http_mp4_read_stss_atom
- ngx_http_mp4_read_stsz_atom
- ngx_http_mp4_read_stts_atom
- ngx_http_mp4_read_tkhd_atom
- ngx_http_mp4_read_trak_atom
- ngx_http_mp4_read_vmhd_atom
- ngx_http_mp4_update_co64_atom
- ngx_http_mp4_update_ctts_atom
- ngx_http_mp4_update_mdat_atom
- ngx_http_mp4_update_mdia_atom
- ngx_http_mp4_update_minf_atom
- ngx_http_mp4_update_stbl_atom
- ngx_http_mp4_update_stco_atom
- ngx_http_mp4_update_stsc_atom
- ngx_http_mp4_update_stss_atom
- ngx_http_mp4_update_stsz_atom
- ngx_http_mp4_update_stts_atom
- ngx_http_mp4_update_trak_atom

Macros defined

- NGX_HTTP_MP4_CO64_ATOM
- NGX_HTTP_MP4_CO64_DATA
- NGX_HTTP_MP4_CTTS_ATOM
- NGX_HTTP_MP4_CTTS_DATA
- NGX_HTTP_MP4_DINF_ATOM
- NGX_HTTP_MP4_HDLR_ATOM
- NGX_HTTP_MP4_LAST_ATOM
- NGX_HTTP_MP4_MDHD_ATOM
- NGX_HTTP_MP4_MDIA_ATOM
- NGX_HTTP_MP4_MINF_ATOM
- NGX_HTTP_MP4_MOOV_BUFFER_EXCESS
- NGX_HTTP_MP4_SMHD_ATOM

- NGX_HTTP_MP4_STBL_ATOM
- NGX_HTTP_MP4_STCO_ATOM
- NGX_HTTP_MP4_STCO_DATA
- NGX_HTTP_MP4_STSC_ATOM
- NGX_HTTP_MP4_STSC_DATA
- NGX_HTTP_MP4_STSC_END
- NGX_HTTP_MP4_STSC_START
- NGX_HTTP_MP4_STSD_ATOM
- NGX_HTTP_MP4_STSS_ATOM
- NGX_HTTP_MP4_STSS_DATA
- NGX_HTTP_MP4_STSZ_ATOM
- NGX_HTTP_MP4_STSZ_DATA
- NGX_HTTP_MP4_STTS_ATOM
- NGX_HTTP_MP4_STTS_DATA
- NGX_HTTP_MP4_TKHD_ATOM
- NGX_HTTP_MP4_TRAK_ATOM
- NGX_HTTP_MP4_VMHD_ATOM
- ngx_mp4_atom_data
- ngx_mp4_atom_data_size
- ngx_mp4_atom_header
- ngx_mp4_atom_next
- ngx_mp4_get_32value
- ngx_mp4_get_64value
- ngx_mp4_last_trak
- ngx_mp4_set_32value
- ngx_mp4_set_64value
- ngx_mp4_set_atom_name

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9  #include <ngx_http.h>
10
11
```

```
#define NGX_HTTP_MP4_TRAK_ATOM     0
#define NGX_HTTP_MP4_TKHD_ATOM     1
#define NGX_HTTP_MP4_MDIA_ATOM     2
#define NGX_HTTP_MP4_MDHD_ATOM     3
#define NGX_HTTP_MP4_HDLR_ATOM     4
#define NGX_HTTP_MP4_MINF_ATOM     5
#define NGX_HTTP_MP4_VMHD_ATOM     6
#define NGX_HTTP_MP4_SMHD_ATOM     7
#define NGX_HTTP_MP4_DINF_ATOM     8
#define NGX_HTTP_MP4_STBL_ATOM     9
#define NGX_HTTP_MP4_STSD_ATOM    10
#define NGX_HTTP_MP4_STTS_ATOM    11
#define NGX_HTTP_MP4_STTS_DATA    12
#define NGX_HTTP_MP4_STSS_ATOM    13
#define NGX_HTTP_MP4_STSS_DATA    14
#define NGX_HTTP_MP4_CTTS_ATOM    15
#define NGX_HTTP_MP4_CTTS_DATA    16
#define NGX_HTTP_MP4_STSC_ATOM    17
#define NGX_HTTP_MP4_STSC_START   18
#define NGX_HTTP_MP4_STSC_DATA    19
#define NGX_HTTP_MP4_STSC_END     20
#define NGX_HTTP_MP4_STSZ_ATOM    21
#define NGX_HTTP_MP4_STSZ_DATA    22
#define NGX_HTTP_MP4_STCO_ATOM    23
#define NGX_HTTP_MP4_STCO_DATA    24
#define NGX_HTTP_MP4_CO64_ATOM    25
#define NGX_HTTP_MP4_CO64_DATA    26

#define NGX_HTTP_MP4_LAST_ATOM    NGX_HTTP_MP4_CO64_DATA


typedef struct {
    size_t                buffer_size;
    size_t                max_buffer_size;
} ngx_http_mp4_conf_t;


typedef struct {
    u_char                chunk[4];
    u_char                samples[4];
    u_char                id[4];
} ngx_mp4_stsc_entry_t;


typedef struct {
    uint32_t              timescale;
    uint32_t              time_to_sample_entries;
    uint32_t              sample_to_chunk_entries;
    uint32_t              sync_samples_entries;
    uint32_t              composition_offset_entries;
    uint32_t              sample_sizes_entries;
    uint32_t              chunks;

    ngx_uint_t            start_sample;
    ngx_uint_t            end_sample;
    ngx_uint_t            start_chunk;
    ngx_uint_t            end_chunk;
    ngx_uint_t            start_chunk_samples;
    ngx_uint_t            end_chunk_samples;
    uint64_t              start_chunk_samples_size;
    uint64_t              end_chunk_samples_size;
    off_t                 start_offset;
    off_t                 end_offset;

    size_t                tkhd_size;
    size_t                mdhd_size;
    size_t                hdlr_size;
    size_t                vmhd_size;
    size_t                smhd_size;
    size_t                dinf_size;
    size_t                size;

    ngx_chain_t           out[NGX_HTTP_MP4_LAST_ATOM + 1];

    ngx_buf_t             trak_atom_buf;
    ngx_buf_t             tkhd_atom_buf;
```

```
    ngx_buf_t             mdia_atom_buf;
    ngx_buf_t             mdhd_atom_buf;
    ngx_buf_t             hdlr_atom_buf;
    ngx_buf_t             minf_atom_buf;
    ngx_buf_t             vmhd_atom_buf;
    ngx_buf_t             smhd_atom_buf;
    ngx_buf_t             dinf_atom_buf;
    ngx_buf_t             stbl_atom_buf;
    ngx_buf_t             stsd_atom_buf;
    ngx_buf_t             stts_atom_buf;
    ngx_buf_t             stts_data_buf;
    ngx_buf_t             stss_atom_buf;
     ngx_buf_t             stss_data_buf;
     ngx_buf_t             ctts_atom_buf;
     ngx_buf_t             ctts_data_buf;
     ngx_buf_t             stsc_atom_buf;
     ngx_buf_t             stsc_start_chunk_buf;
     ngx_buf_t             stsc_end_chunk_buf;
     ngx_buf_t             stsc_data_buf;
     ngx_buf_t             stsz_atom_buf;
     ngx_buf_t             stsz_data_buf;
     ngx_buf_t             stco_atom_buf;
     ngx_buf_t             stco_data_buf;
     ngx_buf_t             co64_atom_buf;
     ngx_buf_t             co64_data_buf;

     ngx_mp4_stsc_entry_t  stsc_start_chunk_entry;
     ngx_mp4_stsc_entry_t  stsc_end_chunk_entry;
} ngx_http_mp4_trak_t;


typedef struct {
     ngx_file_t            file;

     u_char               *buffer;
     u_char               *buffer_start;
     u_char               *buffer_pos;
     u_char               *buffer_end;
     size_t                buffer_size;

     off_t                 offset;
     off_t                 end;
     off_t                 content_length;
     ngx_uint_t            start;
     ngx_uint_t            length;
     uint32_t              timescale;
     ngx_http_request_t   *request;
     ngx_array_t           trak;
     ngx_http_mp4_trak_t   traks[2];

     size_t                ftyp_size;
     size_t                moov_size;

     ngx_chain_t          *out;
     ngx_chain_t           ftyp_atom;
     ngx_chain_t           moov_atom;
     ngx_chain_t           mvhd_atom;
     ngx_chain_t           mdat_atom;
     ngx_chain_t           mdat_data;

     ngx_buf_t             ftyp_atom_buf;
     ngx_buf_t             moov_atom_buf;
     ngx_buf_t             mvhd_atom_buf;
     ngx_buf_t             mdat_atom_buf;
     ngx_buf_t             mdat_data_buf;

     u_char                moov_atom_header[8];
     u_char                mdat_atom_header[16];
} ngx_http_mp4_file_t;


typedef struct {
     char                 *name;
     ngx_int_t           (*handler)(ngx_http_mp4_file_t *mp4,
                                    uint64_t atom_data_size);
} ngx_http_mp4_atom_handler_t;
```

```
164
165
166 #define ngx_mp4_atom_header(mp4)   (mp4->buffer_pos - 8)
167 #define ngx_mp4_atom_data(mp4)     mp4->buffer_pos
168 #define ngx_mp4_atom_data_size(t)  (uint64_t) (sizeof(t) - 8)
169
170
171 #define ngx_mp4_atom_next(mp4, n)                                           \
172     mp4->buffer_pos += (size_t) n;                                          \
173     mp4->offset += n
174
175
176 #define ngx_mp4_set_atom_name(p, n1, n2, n3, n4)                            \
177     ((u_char *) (p))[4] = n1;                                               \
178     ((u_char *) (p))[5] = n2;                                               \
179     ((u_char *) (p))[6] = n3;                                               \
180     ((u_char *) (p))[7] = n4
181
182 #define ngx_mp4_get_32value(p)                                              \
183     ( ((uint32_t) ((u_char *) (p))[0] << 24)                                \
184     + (           ((u_char *) (p))[1] << 16)                                \
185     + (           ((u_char *) (p))[2] << 8)                                 \
186     + (           ((u_char *) (p))[3]) )
187
188 #define ngx_mp4_set_32value(p, n)                                           \
189     ((u_char *) (p))[0] = (u_char) ((n) >> 24);                             \
190     ((u_char *) (p))[1] = (u_char) ((n) >> 16);                             \
191     ((u_char *) (p))[2] = (u_char) ((n) >> 8);                              \
192     ((u_char *) (p))[3] = (u_char)  (n)
193
194 #define ngx_mp4_get_64value(p)                                              \
195     ( ((uint64_t) ((u_char *) (p))[0] << 56)                                \
196     + ((uint64_t) ((u_char *) (p))[1] << 48)                                \
197     + ((uint64_t) ((u_char *) (p))[2] << 40)                                \
198     + ((uint64_t) ((u_char *) (p))[3] << 32)                                \
199     + ((uint64_t) ((u_char *) (p))[4] << 24)                                \
200     + (           ((u_char *) (p))[5] << 16)                                \
201     + (           ((u_char *) (p))[6] << 8)                                 \
202     + (           ((u_char *) (p))[7]) )
203
204 #define ngx_mp4_set_64value(p, n)                                           \
205     ((u_char *) (p))[0] = (u_char) ((uint64_t) (n) >> 56);                  \
206     ((u_char *) (p))[1] = (u_char) ((uint64_t) (n) >> 48);                  \
207     ((u_char *) (p))[2] = (u_char) ((uint64_t) (n) >> 40);                  \
208     ((u_char *) (p))[3] = (u_char) ((uint64_t) (n) >> 32);                  \
209     ((u_char *) (p))[4] = (u_char) (           (n) >> 24);                  \
210     ((u_char *) (p))[5] = (u_char) (           (n) >> 16);                  \
211     ((u_char *) (p))[6] = (u_char) (           (n) >> 8);                   \
212     ((u_char *) (p))[7] = (u_char)             (n)
213
214 #define ngx_mp4_last_trak(mp4)                                              \
215     &((ngx_http_mp4_trak_t *) mp4->trak.elts)[mp4->trak.nelts - 1]
216
217
218 static ngx_int_t ngx_http_mp4_handler(ngx_http_request_t *r);
219
220 static ngx_int_t ngx_http_mp4_process(ngx_http_mp4_file_t *mp4);
221 static ngx_int_t ngx_http_mp4_read_atom(ngx_http_mp4_file_t *mp4,
222     ngx_http_mp4_atom_handler_t *atom, uint64_t atom_data_size);
223 static ngx_int_t ngx_http_mp4_read(ngx_http_mp4_file_t *mp4, size_t size);
224 static ngx_int_t ngx_http_mp4_read_ftyp_atom(ngx_http_mp4_file_t *mp4,
225     uint64_t atom_data_size);
226 static ngx_int_t ngx_http_mp4_read_moov_atom(ngx_http_mp4_file_t *mp4,
227     uint64_t atom_data_size);
228 static ngx_int_t ngx_http_mp4_read_mdat_atom(ngx_http_mp4_file_t *mp4,
229     uint64_t atom_data_size);
230 static size_t ngx_http_mp4_update_mdat_atom(ngx_http_mp4_file_t *mp4,
231     off_t start_offset, off_t end_offset);
232 static ngx_int_t ngx_http_mp4_read_mvhd_atom(ngx_http_mp4_file_t *mp4,
233     uint64_t atom_data_size);
234 static ngx_int_t ngx_http_mp4_read_trak_atom(ngx_http_mp4_file_t *mp4,
235     uint64_t atom_data_size);
236 static void ngx_http_mp4_update_trak_atom(ngx_http_mp4_file_t *mp4,
237     ngx_http_mp4_trak_t *trak);
238 static ngx_int_t ngx_http_mp4_read_cmov_atom(ngx_http_mp4_file_t *mp4,
239     uint64_t atom_data_size);
```

```
static ngx_int_t ngx_http_mp4_read_tkhd_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_mdia_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static void ngx_http_mp4_update_mdia_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static ngx_int_t ngx_http_mp4_read_mdhd_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_hdlr_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_minf_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static void ngx_http_mp4_update_minf_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static ngx_int_t ngx_http_mp4_read_dinf_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_vmhd_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_smhd_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_stbl_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static void ngx_http_mp4_update_stbl_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static ngx_int_t ngx_http_mp4_read_stsd_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_read_stts_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_update_stts_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static ngx_int_t ngx_http_mp4_crop_stts_data(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, ngx_uint_t start);
static ngx_int_t ngx_http_mp4_read_stss_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_update_stss_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static void ngx_http_mp4_crop_stss_data(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, ngx_uint_t start);
static ngx_int_t ngx_http_mp4_read_ctts_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static void ngx_http_mp4_update_ctts_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static void ngx_http_mp4_crop_ctts_data(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, ngx_uint_t start);
static ngx_int_t ngx_http_mp4_read_stsc_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_update_stsc_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static ngx_int_t ngx_http_mp4_crop_stsc_data(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, ngx_uint_t start);
static ngx_int_t ngx_http_mp4_read_stsz_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_update_stsz_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static ngx_int_t ngx_http_mp4_read_stco_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_update_stco_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static void ngx_http_mp4_adjust_stco_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, int32_t adjustment);
static ngx_int_t ngx_http_mp4_read_co64_atom(ngx_http_mp4_file_t *mp4,
    uint64_t atom_data_size);
static ngx_int_t ngx_http_mp4_update_co64_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak);
static void ngx_http_mp4_adjust_co64_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, off_t adjustment);

static char *ngx_http_mp4(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static void *ngx_http_mp4_create_conf(ngx_conf_t *cf);
static char *ngx_http_mp4_merge_conf(ngx_conf_t *cf, void *parent, void *child);


static ngx_command_t  ngx_http_mp4_commands[] = {

    { ngx_string("mp4"),
      NGX_HTTP_LOC_CONF|NGX_CONF_NOARGS,
```

```
      ngx_http_mp4,
      0,
      0,
      NULL },

    { ngx_string("mp4_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mp4_conf_t, buffer_size),
      NULL },

    { ngx_string("mp4_max_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mp4_conf_t, max_buffer_size),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_mp4_module_ctx = {
    NULL,                          /* preconfiguration */
    NULL,                          /* postconfiguration */

    NULL,                          /* create main configuration */
    NULL,                          /* init main configuration */

    NULL,                          /* create server configuration */
    NULL,                          /* merge server configuration */

    ngx_http_mp4_create_conf,      /* create location configuration */
    ngx_http_mp4_merge_conf        /* merge location configuration */
};


ngx_module_t  ngx_http_mp4_module = {
    NGX_MODULE_V1,
    &ngx_http_mp4_module_ctx,      /* module context */
    ngx_http_mp4_commands,         /* module directives */
    NGX_HTTP_MODULE,               /* module type */
    NULL,                          /* init master */
    NULL,                          /* init module */
    NULL,                          /* init process */
    NULL,                          /* init thread */
    NULL,                          /* exit thread */
    NULL,                          /* exit process */
    NULL,                          /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_mp4_atom_handler_t  ngx_http_mp4_atoms[] = {
    { "ftyp", ngx_http_mp4_read_ftyp_atom },
    { "moov", ngx_http_mp4_read_moov_atom },
    { "mdat", ngx_http_mp4_read_mdat_atom },
    { NULL, NULL }
};

static ngx_http_mp4_atom_handler_t  ngx_http_mp4_moov_atoms[] = {
    { "mvhd", ngx_http_mp4_read_mvhd_atom },
    { "trak", ngx_http_mp4_read_trak_atom },
    { "cmov", ngx_http_mp4_read_cmov_atom },
    { NULL, NULL }
};

static ngx_http_mp4_atom_handler_t  ngx_http_mp4_trak_atoms[] = {
    { "tkhd", ngx_http_mp4_read_tkhd_atom },
    { "mdia", ngx_http_mp4_read_mdia_atom },
    { NULL, NULL }
};

static ngx_http_mp4_atom_handler_t  ngx_http_mp4_mdia_atoms[] = {
    { "mdhd", ngx_http_mp4_read_mdhd_atom },
```

```
    { "hdlr", ngx_http_mp4_read_hdlr_atom },
    { "minf", ngx_http_mp4_read_minf_atom },
    { NULL, NULL }
};

static ngx_http_mp4_atom_handler_t  ngx_http_mp4_minf_atoms[] = {
    { "vmhd", ngx_http_mp4_read_vmhd_atom },
    { "smhd", ngx_http_mp4_read_smhd_atom },
    { "dinf", ngx_http_mp4_read_dinf_atom },
    { "stbl", ngx_http_mp4_read_stbl_atom },
    { NULL, NULL }
};

static ngx_http_mp4_atom_handler_t  ngx_http_mp4_stbl_atoms[] = {
    { "stsd", ngx_http_mp4_read_stsd_atom },
    { "stts", ngx_http_mp4_read_stts_atom },
    { "stss", ngx_http_mp4_read_stss_atom },
    { "ctts", ngx_http_mp4_read_ctts_atom },
    { "stsc", ngx_http_mp4_read_stsc_atom },
    { "stsz", ngx_http_mp4_read_stsz_atom },
    { "stco", ngx_http_mp4_read_stco_atom },
    { "co64", ngx_http_mp4_read_co64_atom },
    { NULL, NULL }
};


static ngx_int_t
ngx_http_mp4_handler(ngx_http_request_t *r)
{
    u_char                     *last;
    size_t                      root;
    ngx_int_t                   rc, start, end;
    ngx_uint_t                  level, length;
    ngx_str_t                   path, value;
    ngx_log_t                  *log;
    ngx_buf_t                  *b;
    ngx_chain_t                 out;
    ngx_http_mp4_file_t        *mp4;
    ngx_open_file_info_t        of;
    ngx_http_core_loc_conf_t   *clcf;

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD))) {
        return NGX_HTTP_NOT_ALLOWED;
    }

    if (r->uri.data[r->uri.len - 1] == '/') {
        return NGX_DECLINED;
    }

    rc = ngx_http_discard_request_body(r);

    if (rc != NGX_OK) {
        return rc;
    }

    last = ngx_http_map_uri_to_path(r, &path, &root, 0);
    if (last == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    log = r->connection->log;

    path.len = last - path.data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, log, 0,
                   "http mp4 filename: \"%V\"", &path);

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.read_ahead = clcf->read_ahead;
    of.directio = NGX_MAX_OFF_T_VALUE;
    of.valid = clcf->open_file_cache_valid;
    of.min_uses = clcf->open_file_cache_min_uses;
    of.errors = clcf->open_file_cache_errors;
```

```
    of.events = clcf->open_file_cache_events;

    if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
        != NGX_OK)
    {
        switch (of.err) {

        case 0:
            return NGX_HTTP_INTERNAL_SERVER_ERROR;

        case NGX_ENOENT:
        case NGX_ENOTDIR:
        case NGX_ENAMETOOLONG:

            level = NGX_LOG_ERR;
            rc = NGX_HTTP_NOT_FOUND;
            break;

        case NGX_EACCES:
#if (NGX_HAVE_OPENAT)
        case NGX_EMLINK:
        case NGX_ELOOP:
#endif

            level = NGX_LOG_ERR;
            rc = NGX_HTTP_FORBIDDEN;
            break;

        default:

            level = NGX_LOG_CRIT;
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
            break;
        }

        if (rc != NGX_HTTP_NOT_FOUND || clcf->log_not_found) {
            ngx_log_error(level, log, of.err,
                          "%s \"%s\" failed", of.failed, path.data);
        }

        return rc;
    }

    if (!of.is_file) {

        if (ngx_close_file(of.fd) == NGX_FILE_ERROR) {
            ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
                          ngx_close_file_n " \"%s\" failed", path.data);
        }

        return NGX_DECLINED;
    }

    r->root_tested = !r->error_page;
    r->allow_ranges = 1;

    start = -1;
    length = 0;
    r->headers_out.content_length_n = of.size;
    mp4 = NULL;
    b = NULL;

    if (r->args.len) {

        if (ngx_http_arg(r, (u_char *) "start", 5, &value) == NGX_OK) {

            /*
             * A Flash player may send start value with a lot of digits
             * after dot so strtod() is used instead of atofp().  NaNs and
             * infinities become negative numbers after (int) conversion.
             */

```

```
            ngx_set_errno(0);
            start = (int) (strtod((char *) value.data, NULL) * 1000);

            if (ngx_errno != 0) {
                start = -1;
            }
        }

        if (ngx_http_arg(r, (u_char *) "end", 3, &value) == NGX_OK) {

            ngx_set_errno(0);
            end = (int) (strtod((char *) value.data, NULL) * 1000);

            if (ngx_errno != 0) {
                end = -1;
            }

            if (end > 0) {
                if (start < 0) {
                    start = 0;
                }

                if (end > start) {
                    length = end - start;
                }
            }
        }
    }

    if (start >= 0) {
        r->single_range = 1;

        mp4 = ngx_pcalloc(r->pool, sizeof(ngx_http_mp4_file_t));
        if (mp4 == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        mp4->file.fd = of.fd;
        mp4->file.name = path;
        mp4->file.log = r->connection->log;
        mp4->end = of.size;
        mp4->start = (ngx_uint_t) start;
        mp4->length = length;
        mp4->request = r;

        switch (ngx_http_mp4_process(mp4)) {

        case NGX_DECLINED:
            if (mp4->buffer) {
                ngx_pfree(r->pool, mp4->buffer);
            }

            ngx_pfree(r->pool, mp4);
            mp4 = NULL;

            break;

        case NGX_OK:
            r->headers_out.content_length_n = mp4->content_length;
            break;

        default: /* NGX_ERROR */
            if (mp4->buffer) {
                ngx_pfree(r->pool, mp4->buffer);
            }

            ngx_pfree(r->pool, mp4);

            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }
    }

    log->action = "sending mp4 to client";

    if (clcf->directio <= of.size) {

```

```
620         /*
621          * DIRECTIO is set on transfer only
622          * to allow kernel to cache "moov" atom
623          */
624
625         if (ngx_directio_on(of.fd) == NGX_FILE_ERROR) {
626             ngx_log_error(NGX_LOG_ALERT, log, ngx_errno,
627                           ngx_directio_on_n " \"%s\" failed", path.data);
628         }
629
630         of.is_directio = 1;
631
632         if (mp4) {
633             mp4->file.directio = 1;
634         }
635     }
636
637     r->headers_out.status = NGX_HTTP_OK;
638     r->headers_out.last_modified_time = of.mtime;
639
640     if (ngx_http_set_etag(r) != NGX_OK) {
641         return NGX_HTTP_INTERNAL_SERVER_ERROR;
642     }
643
644     if (ngx_http_set_content_type(r) != NGX_OK) {
645         return NGX_HTTP_INTERNAL_SERVER_ERROR;
646     }
647
648     if (mp4 == NULL) {
649         b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
650         if (b == NULL) {
651             return NGX_HTTP_INTERNAL_SERVER_ERROR;
652         }
653
654         b->file = ngx_pcalloc(r->pool, sizeof(ngx_file_t));
655         if (b->file == NULL) {
656             return NGX_HTTP_INTERNAL_SERVER_ERROR;
657         }
658     }
659
660     rc = ngx_http_send_header(r);
661
662     if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
663         return rc;
664     }
665
666     if (mp4) {
667         return ngx_http_output_filter(r, mp4->out);
668     }
669
670     b->file_pos = 0;
671     b->file_last = of.size;
672
673     b->in_file = b->file_last ? 1 : 0;
674     b->last_buf = (r == r->main) ? 1 : 0;
675     b->last_in_chain = 1;
676
677     b->file->fd = of.fd;
678     b->file->name = path;
679     b->file->log = log;
680     b->file->directio = of.is_directio;
681
682     out.buf = b;
683     out.next = NULL;
684
685     return ngx_http_output_filter(r, &out);
686 }
687
688
689 static ngx_int_t
690 ngx_http_mp4_process(ngx_http_mp4_file_t *mp4)
691 {
692     off_t                   start_offset, end_offset, adjustment;
693     ngx_int_t               rc;
694     ngx_uint_t              i, j;
695     ngx_chain_t           **prev;
```

```
    ngx_http_mp4_trak_t   *trak;
    ngx_http_mp4_conf_t   *conf;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mp4 start:%ui, length:%ui", mp4->start, mp4->length);

    conf = ngx_http_get_module_loc_conf(mp4->request, ngx_http_mp4_module);

    mp4->buffer_size = conf->buffer_size;

    rc = ngx_http_mp4_read_atom(mp4, ngx_http_mp4_atoms, mp4->end);
    if (rc != NGX_OK) {
        return rc;
    }

    if (mp4->trak.nelts == 0) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "no mp4 trak atoms were found in \"%s\"",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    if (mp4->mdat_atom.buf == NULL) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "no mp4 mdat atom was found in \"%s\"",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    prev = &mp4->out;

    if (mp4->ftyp_atom.buf) {
        *prev = &mp4->ftyp_atom;
        prev = &mp4->ftyp_atom.next;
    }

    *prev = &mp4->moov_atom;
    prev = &mp4->moov_atom.next;

    if (mp4->mvhd_atom.buf) {
        mp4->moov_size += mp4->mvhd_atom_buf.last - mp4->mvhd_atom_buf.pos;
        *prev = &mp4->mvhd_atom;
        prev = &mp4->mvhd_atom.next;
    }

    start_offset = mp4->end;
    end_offset = 0;
    trak = mp4->trak.elts;

    for (i = 0; i < mp4->trak.nelts; i++) {

        if (ngx_http_mp4_update_stts_atom(mp4, &trak[i]) != NGX_OK) {
            return NGX_ERROR;
        }

        if (ngx_http_mp4_update_stss_atom(mp4, &trak[i]) != NGX_OK) {
            return NGX_ERROR;
        }

        ngx_http_mp4_update_ctts_atom(mp4, &trak[i]);

        if (ngx_http_mp4_update_stsc_atom(mp4, &trak[i]) != NGX_OK) {
            return NGX_ERROR;
        }

        if (ngx_http_mp4_update_stsz_atom(mp4, &trak[i]) != NGX_OK) {
            return NGX_ERROR;
        }

        if (trak[i].out[NGX_HTTP_MP4_CO64_DATA].buf) {
            if (ngx_http_mp4_update_co64_atom(mp4, &trak[i]) != NGX_OK) {
                return NGX_ERROR;
            }

        } else {
            if (ngx_http_mp4_update_stco_atom(mp4, &trak[i]) != NGX_OK) {
```

```
                return NGX_ERROR;
            }
        }

        ngx_http_mp4_update_stbl_atom(mp4, &trak[i]);
        ngx_http_mp4_update_minf_atom(mp4, &trak[i]);
        trak[i].size += trak[i].mdhd_size;
        trak[i].size += trak[i].hdlr_size;
        ngx_http_mp4_update_mdia_atom(mp4, &trak[i]);
        trak[i].size += trak[i].tkhd_size;
        ngx_http_mp4_update_trak_atom(mp4, &trak[i]);

        mp4->moov_size += trak[i].size;

        if (start_offset > trak[i].start_offset) {
            start_offset = trak[i].start_offset;
        }

        if (end_offset < trak[i].end_offset) {
            end_offset = trak[i].end_offset;
        }

        *prev = &trak[i].out[NGX_HTTP_MP4_TRAK_ATOM];
        prev = &trak[i].out[NGX_HTTP_MP4_TRAK_ATOM].next;

        for (j = 0; j < NGX_HTTP_MP4_LAST_ATOM + 1; j++) {
            if (trak[i].out[j].buf) {
                *prev = &trak[i].out[j];
                prev = &trak[i].out[j].next;
            }
        }
    }

    if (end_offset < start_offset) {
        end_offset = start_offset;
    }

    mp4->moov_size += 8;

    ngx_mp4_set_32value(mp4->moov_atom_header, mp4->moov_size);
    ngx_mp4_set_atom_name(mp4->moov_atom_header, 'm', 'o', 'o', 'v');
    mp4->content_length += mp4->moov_size;

    *prev = &mp4->mdat_atom;

    if (start_offset > mp4->mdat_data.buf->file_last) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "start time is out mp4 mdat atom in \"%s\"",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    adjustment = mp4->ftyp_size + mp4->moov_size
                 + ngx_http_mp4_update_mdat_atom(mp4, start_offset, end_offset)
                 - start_offset;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mp4 adjustment:%O", adjustment);

    for (i = 0; i < mp4->trak.nelts; i++) {
        if (trak[i].out[NGX_HTTP_MP4_CO64_DATA].buf) {
            ngx_http_mp4_adjust_co64_atom(mp4, &trak[i], adjustment);
        } else {
            ngx_http_mp4_adjust_stco_atom(mp4, &trak[i], (int32_t) adjustment);
        }
    }

    return NGX_OK;
}


typedef struct {
    u_char    size[4];
    u_char    name[4];
} ngx_mp4_atom_header_t;

```

```
typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    size64[8];
} ngx_mp4_atom_header64_t;


static ngx_int_t
ngx_http_mp4_read_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_atom_handler_t *atom, uint64_t atom_data_size)
{
    off_t        end;
    size_t       atom_header_size;
    u_char      *atom_header, *atom_name;
    uint64_t     atom_size;
    ngx_int_t    rc;
    ngx_uint_t   n;

    end = mp4->offset + atom_data_size;

    while (mp4->offset < end) {

        if (ngx_http_mp4_read(mp4, sizeof(uint32_t)) != NGX_OK) {
            return NGX_ERROR;
        }

        atom_header = mp4->buffer_pos;
        atom_size = ngx_mp4_get_32value(atom_header);
        atom_header_size = sizeof(ngx_mp4_atom_header_t);

        if (atom_size == 0) {
            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                           "mp4 atom end");
            return NGX_OK;
        }

        if (atom_size < sizeof(ngx_mp4_atom_header_t)) {

            if (atom_size == 1) {

                if (ngx_http_mp4_read(mp4, sizeof(ngx_mp4_atom_header64_t))
                    != NGX_OK)
                {
                    return NGX_ERROR;
                }

                /* 64-bit atom size */
                atom_header = mp4->buffer_pos;
                atom_size = ngx_mp4_get_64value(atom_header + 8);
                atom_header_size = sizeof(ngx_mp4_atom_header64_t);

            } else {
                ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                              "\"%s\" mp4 atom is too small:%uL",
                              mp4->file.name.data, atom_size);
                return NGX_ERROR;
            }
        }

        if (ngx_http_mp4_read(mp4, sizeof(ngx_mp4_atom_header_t)) != NGX_OK) {
            return NGX_ERROR;
        }

        atom_header = mp4->buffer_pos;
        atom_name = atom_header + sizeof(uint32_t);

        ngx_log_debug4(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                       "mp4 atom: %*s @%O:%uL",
                       4, atom_name, mp4->offset, atom_size);

        if (atom_size > (uint64_t) (NGX_MAX_OFF_T_VALUE - mp4->offset)
            || mp4->offset + (off_t) atom_size > end)
        {
            ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                          "\"%s\" mp4 atom too large:%uL",
                          mp4->file.name.data, atom_size);
```

```
            return NGX_ERROR;
        }

        for (n = 0; atom[n].name; n++) {

            if (ngx_strncmp(atom_name, atom[n].name, 4) == 0) {

                ngx_mp4_atom_next(mp4, atom_header_size);

                rc = atom[n].handler(mp4, atom_size - atom_header_size);
                if (rc != NGX_OK) {
                    return rc;
                }

                goto next;
            }
        }

        ngx_mp4_atom_next(mp4, atom_size);

    next:
        continue;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_read(ngx_http_mp4_file_t *mp4, size_t size)
{
    ssize_t  n;

    if (mp4->buffer_pos + size <= mp4->buffer_end) {
        return NGX_OK;
    }

    if (mp4->offset + (off_t) mp4->buffer_size > mp4->end) {
        mp4->buffer_size = (size_t) (mp4->end - mp4->offset);
    }

    if (mp4->buffer_size < size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 file truncated", mp4->file.name.data);
        return NGX_ERROR;
    }

    if (mp4->buffer == NULL) {
        mp4->buffer = ngx_palloc(mp4->request->pool, mp4->buffer_size);
        if (mp4->buffer == NULL) {
            return NGX_ERROR;
        }

        mp4->buffer_start = mp4->buffer;
    }

    n = ngx_read_file(&mp4->file, mp4->buffer_start, mp4->buffer_size,
                      mp4->offset);

    if (n == NGX_ERROR) {
        return NGX_ERROR;
    }

    if ((size_t) n != mp4->buffer_size) {
        ngx_log_error(NGX_LOG_CRIT, mp4->file.log, 0,
                      ngx_read_file_n " read only %z of %z from \"%s\"",
                      n, mp4->buffer_size, mp4->file.name.data);
        return NGX_ERROR;
    }

    mp4->buffer_pos = mp4->buffer_start;
    mp4->buffer_end = mp4->buffer_start + mp4->buffer_size;

    return NGX_OK;
}
```

```
1000
1001 static ngx_int_t
1002 ngx_http_mp4_read_ftyp_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1003 {
1004     u_char     *ftyp_atom;
1005     size_t      atom_size;
1006     ngx_buf_t  *atom;
1007
1008     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 ftyp atom");
1009
1010     if (atom_data_size > 1024
1011         || ngx_mp4_atom_data(mp4) + (size_t) atom_data_size > mp4->buffer_end)
1012     {
1013         ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
1014                       "\"%s\" mp4 ftyp atom is too large:%uL",
1015                       mp4->file.name.data, atom_data_size);
1016         return NGX_ERROR;
1017     }
1018
1019     atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
1020
1021     ftyp_atom = ngx_palloc(mp4->request->pool, atom_size);
1022     if (ftyp_atom == NULL) {
1023         return NGX_ERROR;
1024     }
1025
1026     ngx_mp4_set_32value(ftyp_atom, atom_size);
1027     ngx_mp4_set_atom_name(ftyp_atom, 'f', 't', 'y', 'p');
1028
1029     /*
1030      * only moov atom content is guaranteed to be in mp4->buffer
1031      * during sending response, so ftyp atom content should be copied
1032      */
1033     ngx_memcpy(ftyp_atom + sizeof(ngx_mp4_atom_header_t),
1034                ngx_mp4_atom_data(mp4), (size_t) atom_data_size);
1035
1036     atom = &mp4->ftyp_atom_buf;
1037     atom->temporary = 1;
1038     atom->pos = ftyp_atom;
1039     atom->last = ftyp_atom + atom_size;
1040
1041     mp4->ftyp_atom.buf = atom;
1042     mp4->ftyp_size = atom_size;
1043     mp4->content_length = atom_size;
1044
1045     ngx_mp4_atom_next(mp4, atom_data_size);
1046
1047     return NGX_OK;
1048 }
1049
1050
1051 /*
1052 * Small excess buffer to process atoms after moov atom, mp4->buffer_start
1053 * will be set to this buffer part after moov atom processing.
1054 */
1055 #define NGX_HTTP_MP4_MOOV_BUFFER_EXCESS  (4 * 1024)
1056
1057 static ngx_int_t
1058 ngx_http_mp4_read_moov_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1059 {
1060     ngx_int_t             rc;
1061     ngx_uint_t            no_mdat;
1062     ngx_buf_t            *atom;
1063     ngx_http_mp4_conf_t  *conf;
1064
1065     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 moov atom");
1066
1067     no_mdat = (mp4->mdat_atom.buf == NULL);
1068
1069     if (no_mdat && mp4->start == 0 && mp4->length == 0) {
1070         /*
1071          * send original file if moov atom resides before
1072          * mdat atom and client requests integral file
1073          */
1074         return NGX_DECLINED;
1075     }
```

```
1076
1077        conf = ngx_http_get_module_loc_conf(mp4->request, ngx_http_mp4_module);
1078
1079        if (atom_data_size > mp4->buffer_size) {
1080
1081            if (atom_data_size > conf->max_buffer_size) {
1082                ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
1083                              "\"%s\" mp4 moov atom is too large:%uL, "
1084                              "you may want to increase mp4_max_buffer_size",
1085                              mp4->file.name.data, atom_data_size);
1086                return NGX_ERROR;
1087            }
1088
1089            ngx_pfree(mp4->request->pool, mp4->buffer);
1090            mp4->buffer = NULL;
1091            mp4->buffer_pos = NULL;
1092            mp4->buffer_end = NULL;
1093
1094            mp4->buffer_size = (size_t) atom_data_size
1095                             + NGX_HTTP_MP4_MOOV_BUFFER_EXCESS * no_mdat;
1096        }
1097
1098        if (ngx_http_mp4_read(mp4, (size_t) atom_data_size) != NGX_OK) {
1099            return NGX_ERROR;
1100        }
1101
1102        mp4->trak.elts = &mp4->traks;
1103        mp4->trak.size = sizeof(ngx_http_mp4_trak_t);
1104        mp4->trak.nalloc = 2;
1105        mp4->trak.pool = mp4->request->pool;
1106
1107        atom = &mp4->moov_atom_buf;
1108        atom->temporary = 1;
1109        atom->pos = mp4->moov_atom_header;
1110        atom->last = mp4->moov_atom_header + 8;
1111
1112        mp4->moov_atom.buf = &mp4->moov_atom_buf;
1113
1114        rc = ngx_http_mp4_read_atom(mp4, ngx_http_mp4_moov_atoms, atom_data_size);
1115
1116        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 moov atom done");
1117
1118        if (no_mdat) {
1119            mp4->buffer_start = mp4->buffer_pos;
1120            mp4->buffer_size = NGX_HTTP_MP4_MOOV_BUFFER_EXCESS;
1121
1122            if (mp4->buffer_start + mp4->buffer_size > mp4->buffer_end) {
1123                mp4->buffer = NULL;
1124                mp4->buffer_pos = NULL;
1125                mp4->buffer_end = NULL;
1126            }
1127
1128        } else {
1129            /* skip atoms after moov atom */
1130            mp4->offset = mp4->end;
1131        }
1132
1133        return rc;
1134    }
1135
1136
1137    static ngx_int_t
1138    ngx_http_mp4_read_mdat_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1139    {
1140        ngx_buf_t  *data;
1141
1142        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 mdat atom");
1143
1144        data = &mp4->mdat_data_buf;
1145        data->file = &mp4->file;
1146        data->in_file = 1;
1147        data->last_buf = 1;
1148        data->last_in_chain = 1;
1149        data->file_last = mp4->offset + atom_data_size;
1150
1151        mp4->mdat_atom.buf = &mp4->mdat_atom_buf;
```

```
    mp4->mdat_atom.next = &mp4->mdat_data;
    mp4->mdat_data.buf = data;

    if (mp4->trak.nelts) {
        /* skip atoms after mdat atom */
        mp4->offset = mp4->end;

    } else {
        ngx_mp4_atom_next(mp4, atom_data_size);
    }

    return NGX_OK;
}


static size_t
ngx_http_mp4_update_mdat_atom(ngx_http_mp4_file_t *mp4, off_t start_offset,
    off_t end_offset)
{
    off_t       atom_data_size;
    u_char     *atom_header;
    uint32_t    atom_header_size;
    uint64_t    atom_size;
    ngx_buf_t  *atom;

    atom_data_size = end_offset - start_offset;
    mp4->mdat_data.buf->file_pos = start_offset;
    mp4->mdat_data.buf->file_last = end_offset;

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mdat new offset @%O:%O", start_offset, atom_data_size);

    atom_header = mp4->mdat_atom_header;

    if ((uint64_t) atom_data_size > (uint64_t) 0xffffffff) {
        atom_size = 1;
        atom_header_size = sizeof(ngx_mp4_atom_header64_t);
        ngx_mp4_set_64value(atom_header + sizeof(ngx_mp4_atom_header_t),
                            sizeof(ngx_mp4_atom_header64_t) + atom_data_size);
    } else {
        atom_size = sizeof(ngx_mp4_atom_header_t) + atom_data_size;
        atom_header_size = sizeof(ngx_mp4_atom_header_t);
    }

    mp4->content_length += atom_header_size + atom_data_size;

    ngx_mp4_set_32value(atom_header, atom_size);
    ngx_mp4_set_atom_name(atom_header, 'm', 'd', 'a', 't');

    atom = &mp4->mdat_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + atom_header_size;

    return atom_header_size;
}


typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    creation_time[4];
    u_char    modification_time[4];
    u_char    timescale[4];
    u_char    duration[4];
    u_char    rate[4];
    u_char    volume[2];
    u_char    reserved[10];
    u_char    matrix[36];
    u_char    preview_time[4];
    u_char    preview_duration[4];
    u_char    poster_time[4];
    u_char    selection_time[4];
    u_char    selection_duration[4];
```

```
    u_char    current_time[4];
    u_char    next_track_id[4];
} ngx_mp4_mvhd_atom_t;

typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    creation_time[8];
    u_char    modification_time[8];
    u_char    timescale[4];
    u_char    duration[8];
    u_char    rate[4];
    u_char    volume[2];
    u_char    reserved[10];
    u_char    matrix[36];
    u_char    preview_time[4];
    u_char    preview_duration[4];
    u_char    poster_time[4];
    u_char    selection_time[4];
    u_char    selection_duration[4];
    u_char    current_time[4];
    u_char    next_track_id[4];
} ngx_mp4_mvhd64_atom_t;


static ngx_int_t
ngx_http_mp4_read_mvhd_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char                  *atom_header;
    size_t                   atom_size;
    uint32_t                 timescale;
    uint64_t                 duration, start_time, length_time;
    ngx_buf_t               *atom;
    ngx_mp4_mvhd_atom_t     *mvhd_atom;
    ngx_mp4_mvhd64_atom_t   *mvhd64_atom;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 mvhd atom");

    atom_header = ngx_mp4_atom_header(mp4);
    mvhd_atom = (ngx_mp4_mvhd_atom_t *) atom_header;
    mvhd64_atom = (ngx_mp4_mvhd64_atom_t *) atom_header;
    ngx_mp4_set_atom_name(atom_header, 'm', 'v', 'h', 'd');

    if (ngx_mp4_atom_data_size(ngx_mp4_mvhd_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 mvhd atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    if (mvhd_atom->version[0] == 0) {
        /* version 0: 32-bit duration */
        timescale = ngx_mp4_get_32value(mvhd_atom->timescale);
        duration = ngx_mp4_get_32value(mvhd_atom->duration);

    } else {
        /* version 1: 64-bit duration */

        if (ngx_mp4_atom_data_size(ngx_mp4_mvhd64_atom_t) > atom_data_size) {
            ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                          "\"%s\" mp4 mvhd atom too small",
                          mp4->file.name.data);
            return NGX_ERROR;
        }

        timescale = ngx_mp4_get_32value(mvhd64_atom->timescale);
        duration = ngx_mp4_get_64value(mvhd64_atom->duration);
    }

    mp4->timescale = timescale;

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mvhd timescale:%uD, duration:%uL, time:%.3fs",
                   timescale, duration, (double) duration / timescale);

```

```
    start_time = (uint64_t) mp4->start * timescale / 1000;

    if (duration < start_time) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 start time exceeds file duration",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    duration -= start_time;

    if (mp4->length) {
        length_time = (uint64_t) mp4->length * timescale / 1000;

        if (duration > length_time) {
            duration = length_time;
        }
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mvhd new duration:%uL, time:%.3fs",
                   duration, (double) duration / timescale);

    atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
    ngx_mp4_set_32value(mvhd_atom->size, atom_size);

    if (mvhd_atom->version[0] == 0) {
        ngx_mp4_set_32value(mvhd_atom->duration, duration);

    } else {
        ngx_mp4_set_64value(mvhd64_atom->duration, duration);
    }

    atom = &mp4->mvhd_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + atom_size;

    mp4->mvhd_atom.buf = atom;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_read_trak_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char               *atom_header, *atom_end;
    off_t                 atom_file_end;
    ngx_int_t             rc;
    ngx_buf_t            *atom;
    ngx_http_mp4_trak_t  *trak;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 trak atom");

    trak = ngx_array_push(&mp4->trak);
    if (trak == NULL) {
        return NGX_ERROR;
    }

    ngx_memzero(trak, sizeof(ngx_http_mp4_trak_t));

    atom_header = ngx_mp4_atom_header(mp4);
    ngx_mp4_set_atom_name(atom_header, 't', 'r', 'a', 'k');

    atom = &trak->trak_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + sizeof(ngx_mp4_atom_header_t);

    trak->out[NGX_HTTP_MP4_TRAK_ATOM].buf = atom;

    atom_end = mp4->buffer_pos + (size_t) atom_data_size;
    atom_file_end = mp4->offset + atom_data_size;
```

```
1380
1381        rc = ngx_http_mp4_read_atom(mp4, ngx_http_mp4_trak_atoms, atom_data_size);
1382
1383        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
1384                       "mp4 trak atom: %i", rc);
1385
1386        if (rc == NGX_DECLINED) {
1387            /* skip this trak */
1388            ngx_memzero(trak, sizeof(ngx_http_mp4_trak_t));
1389            mp4->trak.nelts--;
1390            mp4->buffer_pos = atom_end;
1391            mp4->offset = atom_file_end;
1392            return NGX_OK;
1393        }
1394
1395        return rc;
1396    }
1397
1398
1399    static void
1400    ngx_http_mp4_update_trak_atom(ngx_http_mp4_file_t *mp4,
1401        ngx_http_mp4_trak_t *trak)
1402    {
1403        ngx_buf_t  *atom;
1404
1405        trak->size += sizeof(ngx_mp4_atom_header_t);
1406        atom = &trak->trak_atom_buf;
1407        ngx_mp4_set_32value(atom->pos, trak->size);
1408    }
1409
1410
1411    static ngx_int_t
1412    ngx_http_mp4_read_cmov_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1413    {
1414        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
1415                      "\"%s\" mp4 compressed moov atom (cmov) is not supported",
1416                      mp4->file.name.data);
1417
1418        return NGX_ERROR;
1419    }
1420
1421
1422    typedef struct {
1423        u_char    size[4];
1424        u_char    name[4];
1425        u_char    version[1];
1426        u_char    flags[3];
1427        u_char    creation_time[4];
1428        u_char    modification_time[4];
1429        u_char    track_id[4];
1430        u_char    reserved1[4];
1431        u_char    duration[4];
1432        u_char    reserved2[8];
1433        u_char    layer[2];
1434        u_char    group[2];
1435        u_char    volume[2];
1436        u_char    reverved3[2];
1437        u_char    matrix[36];
1438        u_char    width[4];
1439        u_char    heigth[4];
1440    } ngx_mp4_tkhd_atom_t;
1441
1442    typedef struct {
1443        u_char    size[4];
1444        u_char    name[4];
1445        u_char    version[1];
1446        u_char    flags[3];
1447        u_char    creation_time[8];
1448        u_char    modification_time[8];
1449        u_char    track_id[4];
1450        u_char    reserved1[4];
1451        u_char    duration[8];
1452        u_char    reserved2[8];
1453        u_char    layer[2];
1454        u_char    group[2];
1455        u_char    volume[2];
```

```
    u_char    reverved3[2];
    u_char    matrix[36];
    u_char    width[4];
    u_char    heigth[4];
} ngx_mp4_tkhd64_atom_t;


static ngx_int_t
ngx_http_mp4_read_tkhd_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char                 *atom_header;
    size_t                  atom_size;
    uint64_t                duration, start_time, length_time;
    ngx_buf_t              *atom;
    ngx_http_mp4_trak_t    *trak;
    ngx_mp4_tkhd_atom_t    *tkhd_atom;
    ngx_mp4_tkhd64_atom_t  *tkhd64_atom;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 tkhd atom");

    atom_header = ngx_mp4_atom_header(mp4);
    tkhd_atom = (ngx_mp4_tkhd_atom_t *) atom_header;
    tkhd64_atom = (ngx_mp4_tkhd64_atom_t *) atom_header;
    ngx_mp4_set_atom_name(tkhd_atom, 't', 'k', 'h', 'd');

    if (ngx_mp4_atom_data_size(ngx_mp4_tkhd_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 tkhd atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    if (tkhd_atom->version[0] == 0) {
        /* version 0: 32-bit duration */
        duration = ngx_mp4_get_32value(tkhd_atom->duration);

    } else {
        /* version 1: 64-bit duration */

        if (ngx_mp4_atom_data_size(ngx_mp4_tkhd64_atom_t) > atom_data_size) {
            ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                          "\"%s\" mp4 tkhd atom too small",
                          mp4->file.name.data);
            return NGX_ERROR;
        }

        duration = ngx_mp4_get_64value(tkhd64_atom->duration);
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "tkhd duration:%uL, time:%.3fs",
                   duration, (double) duration / mp4->timescale);

    start_time = (uint64_t) mp4->start * mp4->timescale / 1000;

    if (duration <= start_time) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                       "tkhd duration is less than start time");
        return NGX_DECLINED;
    }

    duration -= start_time;

    if (mp4->length) {
        length_time = (uint64_t) mp4->length * mp4->timescale / 1000;

        if (duration > length_time) {
            duration = length_time;
        }
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "tkhd new duration:%uL, time:%.3fs",
                   duration, (double) duration / mp4->timescale);

    atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;

```

```
    trak = ngx_mp4_last_trak(mp4);
    trak->tkhd_size = atom_size;

    ngx_mp4_set_32value(tkhd_atom->size, atom_size);

    if (tkhd_atom->version[0] == 0) {
        ngx_mp4_set_32value(tkhd_atom->duration, duration);

    } else {
        ngx_mp4_set_64value(tkhd64_atom->duration, duration);
    }

    atom = &trak->tkhd_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + atom_size;

    trak->out[NGX_HTTP_MP4_TKHD_ATOM].buf = atom;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_read_mdia_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char               *atom_header;
    ngx_buf_t            *atom;
    ngx_http_mp4_trak_t  *trak;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "process mdia atom");

    atom_header = ngx_mp4_atom_header(mp4);
    ngx_mp4_set_atom_name(atom_header, 'm', 'd', 'i', 'a');

    trak = ngx_mp4_last_trak(mp4);

    atom = &trak->mdia_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + sizeof(ngx_mp4_atom_header_t);

    trak->out[NGX_HTTP_MP4_MDIA_ATOM].buf = atom;

    return ngx_http_mp4_read_atom(mp4, ngx_http_mp4_mdia_atoms, atom_data_size);
}


static void
ngx_http_mp4_update_mdia_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak)
{
    ngx_buf_t  *atom;

    trak->size += sizeof(ngx_mp4_atom_header_t);
    atom = &trak->mdia_atom_buf;
    ngx_mp4_set_32value(atom->pos, trak->size);
}


typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    creation_time[4];
    u_char    modification_time[4];
    u_char    timescale[4];
    u_char    duration[4];
    u_char    language[2];
    u_char    quality[2];
} ngx_mp4_mdhd_atom_t;

typedef struct {
```

```
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    creation_time[8];
    u_char    modification_time[8];
    u_char    timescale[4];
    u_char    duration[8];
    u_char    language[2];
    u_char    quality[2];
} ngx_mp4_mdhd64_atom_t;


static ngx_int_t
ngx_http_mp4_read_mdhd_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char                 *atom_header;
    size_t                  atom_size;
    uint32_t                timescale;
    uint64_t                duration, start_time, length_time;
    ngx_buf_t              *atom;
    ngx_http_mp4_trak_t    *trak;
    ngx_mp4_mdhd_atom_t    *mdhd_atom;
    ngx_mp4_mdhd64_atom_t  *mdhd64_atom;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 mdhd atom");

    atom_header = ngx_mp4_atom_header(mp4);
    mdhd_atom = (ngx_mp4_mdhd_atom_t *) atom_header;
    mdhd64_atom = (ngx_mp4_mdhd64_atom_t *) atom_header;
    ngx_mp4_set_atom_name(mdhd_atom, 'm', 'd', 'h', 'd');

    if (ngx_mp4_atom_data_size(ngx_mp4_mdhd_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 mdhd atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    if (mdhd_atom->version[0] == 0) {
        /* version 0: everything is 32-bit */
        timescale = ngx_mp4_get_32value(mdhd_atom->timescale);
        duration = ngx_mp4_get_32value(mdhd_atom->duration);

    } else {
        /* version 1: 64-bit duration and 32-bit timescale */

        if (ngx_mp4_atom_data_size(ngx_mp4_mdhd64_atom_t) > atom_data_size) {
            ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                          "\"%s\" mp4 mdhd atom too small",
                          mp4->file.name.data);
            return NGX_ERROR;
        }

        timescale = ngx_mp4_get_32value(mdhd64_atom->timescale);
        duration = ngx_mp4_get_64value(mdhd64_atom->duration);
    }

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mdhd timescale:%uD, duration:%uL, time:%.3fs",
                   timescale, duration, (double) duration / timescale);

    start_time = (uint64_t) mp4->start * timescale / 1000;

    if (duration <= start_time) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                       "mdhd duration is less than start time");
        return NGX_DECLINED;
    }

    duration -= start_time;

    if (mp4->length) {
        length_time = (uint64_t) mp4->length * timescale / 1000;

        if (duration > length_time) {
            duration = length_time;
```

```
        }
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mdhd new duration:%uL, time:%.3fs",
                   duration, (double) duration / timescale);

    atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;

    trak = ngx_mp4_last_trak(mp4);
    trak->mdhd_size = atom_size;
    trak->timescale = timescale;

    ngx_mp4_set_32value(mdhd_atom->size, atom_size);

    if (mdhd_atom->version[0] == 0) {
        ngx_mp4_set_32value(mdhd_atom->duration, duration);

    } else {
        ngx_mp4_set_64value(mdhd64_atom->duration, duration);
    }

    atom = &trak->mdhd_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + atom_size;

    trak->out[NGX_HTTP_MP4_MDHD_ATOM].buf = atom;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_read_hdlr_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char              *atom_header;
    size_t               atom_size;
    ngx_buf_t            *atom;
    ngx_http_mp4_trak_t  *trak;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 hdlr atom");

    atom_header = ngx_mp4_atom_header(mp4);
    atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
    ngx_mp4_set_32value(atom_header, atom_size);
    ngx_mp4_set_atom_name(atom_header, 'h', 'd', 'l', 'r');

    trak = ngx_mp4_last_trak(mp4);

    atom = &trak->hdlr_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_header + atom_size;

    trak->hdlr_size = atom_size;
    trak->out[NGX_HTTP_MP4_HDLR_ATOM].buf = atom;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_read_minf_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char              *atom_header;
    ngx_buf_t            *atom;
    ngx_http_mp4_trak_t  *trak;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "process minf atom");

    atom_header = ngx_mp4_atom_header(mp4);
```

```
1760      ngx_mp4_set_atom_name(atom_header, 'm', 'i', 'n', 'f');
1761
1762      trak = ngx_mp4_last_trak(mp4);
1763
1764      atom = &trak->minf_atom_buf;
1765      atom->temporary = 1;
1766      atom->pos = atom_header;
1767      atom->last = atom_header + sizeof(ngx_mp4_atom_header_t);
1768
1769      trak->out[NGX_HTTP_MP4_MINF_ATOM].buf = atom;
1770
1771      return ngx_http_mp4_read_atom(mp4, ngx_http_mp4_minf_atoms, atom_data_size);
1772  }
1773
1774
1775  static void
1776  ngx_http_mp4_update_minf_atom(ngx_http_mp4_file_t *mp4,
1777      ngx_http_mp4_trak_t *trak)
1778  {
1779      ngx_buf_t  *atom;
1780
1781      trak->size += sizeof(ngx_mp4_atom_header_t)
1782                  + trak->vmhd_size
1783                  + trak->smhd_size
1784                  + trak->dinf_size;
1785      atom = &trak->minf_atom_buf;
1786      ngx_mp4_set_32value(atom->pos, trak->size);
1787  }
1788
1789
1790  static ngx_int_t
1791  ngx_http_mp4_read_vmhd_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1792  {
1793      u_char              *atom_header;
1794      size_t               atom_size;
1795      ngx_buf_t           *atom;
1796      ngx_http_mp4_trak_t  *trak;
1797
1798      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 vmhd atom");
1799
1800      atom_header = ngx_mp4_atom_header(mp4);
1801      atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
1802      ngx_mp4_set_32value(atom_header, atom_size);
1803      ngx_mp4_set_atom_name(atom_header, 'v', 'm', 'h', 'd');
1804
1805      trak = ngx_mp4_last_trak(mp4);
1806
1807      atom = &trak->vmhd_atom_buf;
1808      atom->temporary = 1;
1809      atom->pos = atom_header;
1810      atom->last = atom_header + atom_size;
1811
1812      trak->vmhd_size += atom_size;
1813      trak->out[NGX_HTTP_MP4_VMHD_ATOM].buf = atom;
1814
1815      ngx_mp4_atom_next(mp4, atom_data_size);
1816
1817      return NGX_OK;
1818  }
1819
1820
1821  static ngx_int_t
1822  ngx_http_mp4_read_smhd_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1823  {
1824      u_char              *atom_header;
1825      size_t               atom_size;
1826      ngx_buf_t           *atom;
1827      ngx_http_mp4_trak_t  *trak;
1828
1829      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 smhd atom");
1830
1831      atom_header = ngx_mp4_atom_header(mp4);
1832      atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
1833      ngx_mp4_set_32value(atom_header, atom_size);
1834      ngx_mp4_set_atom_name(atom_header, 's', 'm', 'h', 'd');
1835
```

```
1836      trak = ngx_mp4_last_trak(mp4);
1837
1838      atom = &trak->smhd_atom_buf;
1839      atom->temporary = 1;
1840      atom->pos = atom_header;
1841      atom->last = atom_header + atom_size;
1842
1843      trak->smhd_size += atom_size;
1844      trak->out[NGX_HTTP_MP4_SMHD_ATOM].buf = atom;
1845
1846      ngx_mp4_atom_next(mp4, atom_data_size);
1847
1848      return NGX_OK;
1849  }
1850
1851
1852  static ngx_int_t
1853  ngx_http_mp4_read_dinf_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1854  {
1855      u_char              *atom_header;
1856      size_t               atom_size;
1857      ngx_buf_t           *atom;
1858      ngx_http_mp4_trak_t  *trak;
1859
1860      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 dinf atom");
1861
1862      atom_header = ngx_mp4_atom_header(mp4);
1863      atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
1864      ngx_mp4_set_32value(atom_header, atom_size);
1865      ngx_mp4_set_atom_name(atom_header, 'd', 'i', 'n', 'f');
1866
1867      trak = ngx_mp4_last_trak(mp4);
1868
1869      atom = &trak->dinf_atom_buf;
1870      atom->temporary = 1;
1871      atom->pos = atom_header;
1872      atom->last = atom_header + atom_size;
1873
1874      trak->dinf_size += atom_size;
1875      trak->out[NGX_HTTP_MP4_DINF_ATOM].buf = atom;
1876
1877      ngx_mp4_atom_next(mp4, atom_data_size);
1878
1879      return NGX_OK;
1880  }
1881
1882
1883  static ngx_int_t
1884  ngx_http_mp4_read_stbl_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
1885  {
1886      u_char              *atom_header;
1887      ngx_buf_t           *atom;
1888      ngx_http_mp4_trak_t  *trak;
1889
1890      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "process stbl atom");
1891
1892      atom_header = ngx_mp4_atom_header(mp4);
1893      ngx_mp4_set_atom_name(atom_header, 's', 't', 'b', 'l');
1894
1895      trak = ngx_mp4_last_trak(mp4);
1896
1897      atom = &trak->stbl_atom_buf;
1898      atom->temporary = 1;
1899      atom->pos = atom_header;
1900      atom->last = atom_header + sizeof(ngx_mp4_atom_header_t);
1901
1902      trak->out[NGX_HTTP_MP4_STBL_ATOM].buf = atom;
1903
1904      return ngx_http_mp4_read_atom(mp4, ngx_http_mp4_stbl_atoms, atom_data_size);
1905  }
1906
1907
1908  static void
1909  ngx_http_mp4_update_stbl_atom(ngx_http_mp4_file_t *mp4,
1910      ngx_http_mp4_trak_t *trak)
1911  {
```

```
    ngx_buf_t  *atom;

    trak->size += sizeof(ngx_mp4_atom_header_t);
    atom = &trak->stbl_atom_buf;
    ngx_mp4_set_32value(atom->pos, trak->size);
}


typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    entries[4];

    u_char    media_size[4];
    u_char    media_name[4];
} ngx_mp4_stsd_atom_t;


static ngx_int_t
ngx_http_mp4_read_stsd_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char               *atom_header, *atom_table;
    size_t                atom_size;
    ngx_buf_t            *atom;
    ngx_mp4_stsd_atom_t  *stsd_atom;
    ngx_http_mp4_trak_t  *trak;

    /* sample description atom */

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 stsd atom");

    atom_header = ngx_mp4_atom_header(mp4);
    stsd_atom = (ngx_mp4_stsd_atom_t *) atom_header;
    atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
    atom_table = atom_header + atom_size;
    ngx_mp4_set_32value(stsd_atom->size, atom_size);
    ngx_mp4_set_atom_name(stsd_atom, 's', 't', 's', 'd');

    if (ngx_mp4_atom_data_size(ngx_mp4_stsd_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 stsd atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "stsd entries:%uD, media:%*s",
                   ngx_mp4_get_32value(stsd_atom->entries),
                   4, stsd_atom->media_name);

    trak = ngx_mp4_last_trak(mp4);

    atom = &trak->stsd_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_table;

    trak->out[NGX_HTTP_MP4_STSD_ATOM].buf = atom;
    trak->size += atom_size;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    entries[4];
} ngx_mp4_stts_atom_t;

typedef struct {
```

```
    u_char    count[4];
    u_char    duration[4];
} ngx_mp4_stts_entry_t;


static ngx_int_t
ngx_http_mp4_read_stts_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char               *atom_header, *atom_table, *atom_end;
    uint32_t              entries;
    ngx_buf_t            *atom, *data;
    ngx_mp4_stts_atom_t  *stts_atom;
    ngx_http_mp4_trak_t  *trak;

    /* time-to-sample atom */

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 stts atom");

    atom_header = ngx_mp4_atom_header(mp4);
    stts_atom = (ngx_mp4_stts_atom_t *) atom_header;
    ngx_mp4_set_atom_name(stts_atom, 's', 't', 't', 's');

    if (ngx_mp4_atom_data_size(ngx_mp4_stts_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 stts atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    entries = ngx_mp4_get_32value(stts_atom->entries);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mp4 time-to-sample entries:%uD", entries);

    if (ngx_mp4_atom_data_size(ngx_mp4_stts_atom_t)
        + entries * sizeof(ngx_mp4_stts_entry_t) > atom_data_size)
    {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 stts atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    atom_table = atom_header + sizeof(ngx_mp4_stts_atom_t);
    atom_end = atom_table + entries * sizeof(ngx_mp4_stts_entry_t);

    trak = ngx_mp4_last_trak(mp4);
    trak->time_to_sample_entries = entries;

    atom = &trak->stts_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_table;

    data = &trak->stts_data_buf;
    data->temporary = 1;
    data->pos = atom_table;
    data->last = atom_end;

    trak->out[NGX_HTTP_MP4_STTS_ATOM].buf = atom;
    trak->out[NGX_HTTP_MP4_STTS_DATA].buf = data;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_update_stts_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak)
{
    size_t               atom_size;
    ngx_buf_t            *atom, *data;
    ngx_mp4_stts_atom_t  *stts_atom;

    /*
     * mdia.minf.stbl.stts updating requires trak->timescale
```

```
2064        * from mdia.mdhd atom which may reside after mdia.minf
2065        */
2066
2067       ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2068                      "mp4 stts atom update");
2069
2070       data = trak->out[NGX_HTTP_MP4_STTS_DATA].buf;
2071
2072       if (data == NULL) {
2073           ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2074                         "no mp4 stts atoms were found in \"%s\"",
2075                         mp4->file.name.data);
2076           return NGX_ERROR;
2077       }
2078
2079       if (ngx_http_mp4_crop_stts_data(mp4, trak, 1) != NGX_OK) {
2080           return NGX_ERROR;
2081       }
2082
2083       if (ngx_http_mp4_crop_stts_data(mp4, trak, 0) != NGX_OK) {
2084           return NGX_ERROR;
2085       }
2086
2087       ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2088                      "time-to-sample entries:%uD", trak->time_to_sample_entries);
2089
2090       atom_size = sizeof(ngx_mp4_stts_atom_t) + (data->last - data->pos);
2091       trak->size += atom_size;
2092
2093       atom = trak->out[NGX_HTTP_MP4_STTS_ATOM].buf;
2094       stts_atom = (ngx_mp4_stts_atom_t *) atom->pos;
2095       ngx_mp4_set_32value(stts_atom->size, atom_size);
2096       ngx_mp4_set_32value(stts_atom->entries, trak->time_to_sample_entries);
2097
2098       return NGX_OK;
2099   }
2100
2101
2102   static ngx_int_t
2103   ngx_http_mp4_crop_stts_data(ngx_http_mp4_file_t *mp4,
2104       ngx_http_mp4_trak_t *trak, ngx_uint_t start)
2105   {
2106       uint32_t               count, duration, rest;
2107       uint64_t               start_time;
2108       ngx_buf_t             *data;
2109       ngx_uint_t             start_sample, entries, start_sec;
2110       ngx_mp4_stts_entry_t  *entry, *end;
2111
2112       if (start) {
2113           start_sec = mp4->start;
2114
2115           ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2116                          "mp4 stts crop start_time:%ui", start_sec);
2117
2118       } else if (mp4->length) {
2119           start_sec = mp4->length;
2120
2121           ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2122                          "mp4 stts crop end_time:%ui", start_sec);
2123
2124       } else {
2125           return NGX_OK;
2126       }
2127
2128       data = trak->out[NGX_HTTP_MP4_STTS_DATA].buf;
2129
2130       start_time = (uint64_t) start_sec * trak->timescale / 1000;
2131
2132       entries = trak->time_to_sample_entries;
2133       start_sample = 0;
2134       entry = (ngx_mp4_stts_entry_t *) data->pos;
2135       end = (ngx_mp4_stts_entry_t *) data->last;
2136
2137       while (entry < end) {
2138           count = ngx_mp4_get_32value(entry->count);
2139           duration = ngx_mp4_get_32value(entry->duration);
```

```
2140
2141        ngx_log_debug3(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2142                       "time:%uL, count:%uD, duration:%uD",
2143                       start_time, count, duration);
2144
2145        if (start_time < (uint64_t) count * duration) {
2146            start_sample += (ngx_uint_t) (start_time / duration);
2147            rest = (uint32_t) (start_time / duration);
2148            goto found;
2149        }
2150
2151        start_sample += count;
2152        start_time -= count * duration;
2153        entries--;
2154        entry++;
2155    }
2156
2157    if (start) {
2158        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2159                      "start time is out mp4 stts samples in \"%s\"",
2160                      mp4->file.name.data);
2161
2162        return NGX_ERROR;
2163
2164    } else {
2165        trak->end_sample = trak->start_sample + start_sample;
2166
2167        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2168                       "end_sample:%ui", trak->end_sample);
2169
2170        return NGX_OK;
2171    }
2172
2173found:
2174
2175    if (start) {
2176        ngx_mp4_set_32value(entry->count, count - rest);
2177        data->pos = (u_char *) entry;
2178        trak->time_to_sample_entries = entries;
2179        trak->start_sample = start_sample;
2180
2181        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2182                       "start_sample:%ui, new count:%uD",
2183                       trak->start_sample, count - rest);
2184
2185    } else {
2186        ngx_mp4_set_32value(entry->count, rest);
2187        data->last = (u_char *) (entry + 1);
2188        trak->time_to_sample_entries -= entries - 1;
2189        trak->end_sample = trak->start_sample + start_sample;
2190
2191        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2192                       "end_sample:%ui, new count:%uD",
2193                       trak->end_sample, rest);
2194    }
2195
2196    return NGX_OK;
2197}
2198
2199
2200typedef struct {
2201    u_char    size[4];
2202    u_char    name[4];
2203    u_char    version[1];
2204    u_char    flags[3];
2205    u_char    entries[4];
2206} ngx_http_mp4_stss_atom_t;
2207
2208
2209static ngx_int_t
2210ngx_http_mp4_read_stss_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
2211{
2212    u_char                  *atom_header, *atom_table, *atom_end;
2213    uint32_t                 entries;
2214    ngx_buf_t               *atom, *data;
2215    ngx_http_mp4_trak_t     *trak;
```

```
2216      ngx_http_mp4_stss_atom_t  *stss_atom;
2217
2218      /* sync samples atom */
2219
2220      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 stss atom");
2221
2222      atom_header = ngx_mp4_atom_header(mp4);
2223      stss_atom = (ngx_http_mp4_stss_atom_t *) atom_header;
2224      ngx_mp4_set_atom_name(stss_atom, 's', 't', 's', 's');
2225
2226      if (ngx_mp4_atom_data_size(ngx_http_mp4_stss_atom_t) > atom_data_size) {
2227          ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2228                        "\"%s\" mp4 stss atom too small", mp4->file.name.data);
2229          return NGX_ERROR;
2230      }
2231
2232      entries = ngx_mp4_get_32value(stss_atom->entries);
2233
2234      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2235                     "sync sample entries:%uD", entries);
2236
2237      trak = ngx_mp4_last_trak(mp4);
2238      trak->sync_samples_entries = entries;
2239
2240      atom_table = atom_header + sizeof(ngx_http_mp4_stss_atom_t);
2241
2242      atom = &trak->stss_atom_buf;
2243      atom->temporary = 1;
2244      atom->pos = atom_header;
2245      atom->last = atom_table;
2246
2247      if (ngx_mp4_atom_data_size(ngx_http_mp4_stss_atom_t)
2248          + entries * sizeof(uint32_t) > atom_data_size)
2249      {
2250          ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2251                        "\"%s\" mp4 stss atom too small", mp4->file.name.data);
2252          return NGX_ERROR;
2253      }
2254
2255      atom_end = atom_table + entries * sizeof(uint32_t);
2256
2257      data = &trak->stss_data_buf;
2258      data->temporary = 1;
2259      data->pos = atom_table;
2260      data->last = atom_end;
2261
2262      trak->out[NGX_HTTP_MP4_STSS_ATOM].buf = atom;
2263      trak->out[NGX_HTTP_MP4_STSS_DATA].buf = data;
2264
2265      ngx_mp4_atom_next(mp4, atom_data_size);
2266
2267      return NGX_OK;
2268  }
2269
2270
2271  static ngx_int_t
2272  ngx_http_mp4_update_stss_atom(ngx_http_mp4_file_t *mp4,
2273      ngx_http_mp4_trak_t *trak)
2274  {
2275      size_t                     atom_size;
2276      uint32_t                   sample, start_sample, *entry, *end;
2277      ngx_buf_t                 *atom, *data;
2278      ngx_http_mp4_stss_atom_t  *stss_atom;
2279
2280      /*
2281       * mdia.minf.stbl.stss updating requires trak->start_sample
2282       * from mdia.minf.stbl.stts which depends on value from mdia.mdhd
2283       * atom which may reside after mdia.minf
2284       */
2285
2286      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2287                     "mp4 stss atom update");
2288
2289      data = trak->out[NGX_HTTP_MP4_STSS_DATA].buf;
2290
2291      if (data == NULL) {
```

```
2292        return NGX_OK;
2293    }
2294
2295    ngx_http_mp4_crop_stss_data(mp4, trak, 1);
2296    ngx_http_mp4_crop_stss_data(mp4, trak, 0);
2297
2298    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2299                   "sync sample entries:%uD", trak->sync_samples_entries);
2300
2301    if (trak->sync_samples_entries) {
2302        entry = (uint32_t *) data->pos;
2303        end = (uint32_t *) data->last;
2304
2305        start_sample = trak->start_sample;
2306
2307        while (entry < end) {
2308            sample = ngx_mp4_get_32value(entry);
2309            sample -= start_sample;
2310            ngx_mp4_set_32value(entry, sample);
2311            entry++;
2312        }
2313
2314    } else {
2315        trak->out[NGX_HTTP_MP4_STSS_DATA].buf = NULL;
2316    }
2317
2318    atom_size = sizeof(ngx_http_mp4_stss_atom_t) + (data->last - data->pos);
2319    trak->size += atom_size;
2320
2321    atom = trak->out[NGX_HTTP_MP4_STSS_ATOM].buf;
2322    stss_atom = (ngx_http_mp4_stss_atom_t *) atom->pos;
2323
2324    ngx_mp4_set_32value(stss_atom->size, atom_size);
2325    ngx_mp4_set_32value(stss_atom->entries, trak->sync_samples_entries);
2326
2327    return NGX_OK;
2328}
2329
2330
2331static void
2332ngx_http_mp4_crop_stss_data(ngx_http_mp4_file_t *mp4,
2333    ngx_http_mp4_trak_t *trak, ngx_uint_t start)
2334{
2335    uint32_t     sample, start_sample, *entry, *end;
2336    ngx_buf_t   *data;
2337    ngx_uint_t   entries;
2338
2339    /* sync samples starts from 1 */
2340
2341    if (start) {
2342        start_sample = trak->start_sample + 1;
2343
2344        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2345                       "mp4 stss crop start_sample:%uD", start_sample);
2346
2347    } else if (mp4->length) {
2348        start_sample = trak->end_sample + 1;
2349
2350        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2351                       "mp4 stss crop end_sample:%uD", start_sample);
2352
2353    } else {
2354        return;
2355    }
2356
2357    data = trak->out[NGX_HTTP_MP4_STSS_DATA].buf;
2358
2359    entries = trak->sync_samples_entries;
2360    entry = (uint32_t *) data->pos;
2361    end = (uint32_t *) data->last;
2362
2363    while (entry < end) {
2364        sample = ngx_mp4_get_32value(entry);
2365
2366        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2367                       "sync:%uD", sample);
```

```
2368
2369            if (sample >= start_sample) {
2370                goto found;
2371            }
2372
2373            entries--;
2374            entry++;
2375        }
2376
2377        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2378                       "sample is out of mp4 stss atom");
2379
2380    found:
2381
2382        if (start) {
2383            data->pos = (u_char *) entry;
2384            trak->sync_samples_entries = entries;
2385
2386        } else {
2387            data->last = (u_char *) entry;
2388            trak->sync_samples_entries -= entries;
2389        }
2390    }
2391
2392
2393    typedef struct {
2394        u_char    size[4];
2395        u_char    name[4];
2396        u_char    version[1];
2397        u_char    flags[3];
2398        u_char    entries[4];
2399    } ngx_mp4_ctts_atom_t;
2400
2401    typedef struct {
2402        u_char    count[4];
2403        u_char    offset[4];
2404    } ngx_mp4_ctts_entry_t;
2405
2406
2407    static ngx_int_t
2408    ngx_http_mp4_read_ctts_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
2409    {
2410        u_char               *atom_header, *atom_table, *atom_end;
2411        uint32_t              entries;
2412        ngx_buf_t            *atom, *data;
2413        ngx_mp4_ctts_atom_t  *ctts_atom;
2414        ngx_http_mp4_trak_t  *trak;
2415
2416        /* composition offsets atom */
2417
2418        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 ctts atom");
2419
2420        atom_header = ngx_mp4_atom_header(mp4);
2421        ctts_atom = (ngx_mp4_ctts_atom_t *) atom_header;
2422        ngx_mp4_set_atom_name(ctts_atom, 'c', 't', 't', 's');
2423
2424        if (ngx_mp4_atom_data_size(ngx_mp4_ctts_atom_t) > atom_data_size) {
2425            ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2426                          "\"%s\" mp4 ctts atom too small", mp4->file.name.data);
2427            return NGX_ERROR;
2428        }
2429
2430        entries = ngx_mp4_get_32value(ctts_atom->entries);
2431
2432        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2433                       "composition offset entries:%uD", entries);
2434
2435        trak = ngx_mp4_last_trak(mp4);
2436        trak->composition_offset_entries = entries;
2437
2438        atom_table = atom_header + sizeof(ngx_mp4_ctts_atom_t);
2439
2440        atom = &trak->ctts_atom_buf;
2441        atom->temporary = 1;
2442        atom->pos = atom_header;
2443        atom->last = atom_table;
```

```
2444
2445        if (ngx_mp4_atom_data_size(ngx_mp4_ctts_atom_t)
2446            + entries * sizeof(ngx_mp4_ctts_entry_t) > atom_data_size)
2447        {
2448            ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2449                          "\"%s\" mp4 ctts atom too small", mp4->file.name.data);
2450            return NGX_ERROR;
2451        }
2452
2453        atom_end = atom_table + entries * sizeof(ngx_mp4_ctts_entry_t);
2454
2455        data = &trak->ctts_data_buf;
2456        data->temporary = 1;
2457        data->pos = atom_table;
2458        data->last = atom_end;
2459
2460        trak->out[NGX_HTTP_MP4_CTTS_ATOM].buf = atom;
2461        trak->out[NGX_HTTP_MP4_CTTS_DATA].buf = data;
2462
2463        ngx_mp4_atom_next(mp4, atom_data_size);
2464
2465        return NGX_OK;
2466    }
2467
2468
2469    static void
2470    ngx_http_mp4_update_ctts_atom(ngx_http_mp4_file_t *mp4,
2471        ngx_http_mp4_trak_t *trak)
2472    {
2473        size_t                atom_size;
2474        ngx_buf_t            *atom, *data;
2475        ngx_mp4_ctts_atom_t  *ctts_atom;
2476
2477        /*
2478         * mdia.minf.stbl.ctts updating requires trak->start_sample
2479         * from mdia.minf.stbl.stts which depends on value from mdia.mdhd
2480         * atom which may reside after mdia.minf
2481         */
2482
2483        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2484                       "mp4 ctts atom update");
2485
2486        data = trak->out[NGX_HTTP_MP4_CTTS_DATA].buf;
2487
2488        if (data == NULL) {
2489            return;
2490        }
2491
2492        ngx_http_mp4_crop_ctts_data(mp4, trak, 1);
2493        ngx_http_mp4_crop_ctts_data(mp4, trak, 0);
2494
2495        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2496                       "composition offset entries:%uD",
2497                       trak->composition_offset_entries);
2498
2499        if (trak->composition_offset_entries == 0) {
2500            trak->out[NGX_HTTP_MP4_CTTS_ATOM].buf = NULL;
2501            trak->out[NGX_HTTP_MP4_CTTS_DATA].buf = NULL;
2502            return;
2503        }
2504
2505        atom_size = sizeof(ngx_mp4_ctts_atom_t) + (data->last - data->pos);
2506        trak->size += atom_size;
2507
2508        atom = trak->out[NGX_HTTP_MP4_CTTS_ATOM].buf;
2509        ctts_atom = (ngx_mp4_ctts_atom_t *) atom->pos;
2510
2511        ngx_mp4_set_32value(ctts_atom->size, atom_size);
2512        ngx_mp4_set_32value(ctts_atom->entries, trak->composition_offset_entries);
2513
2514        return;
2515    }
2516
2517
2518    static void
2519    ngx_http_mp4_crop_ctts_data(ngx_http_mp4_file_t *mp4,
```

```
2520      ngx_http_mp4_trak_t *trak, ngx_uint_t start)
2521  {
2522      uint32_t               count, start_sample, rest;
2523      ngx_buf_t             *data;
2524      ngx_uint_t             entries;
2525      ngx_mp4_ctts_entry_t  *entry, *end;
2526
2527      /* sync samples starts from 1 */
2528
2529      if (start) {
2530          start_sample = trak->start_sample + 1;
2531
2532          ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2533                         "mp4 ctts crop start_sample:%uD", start_sample);
2534
2535      } else if (mp4->length) {
2536          start_sample = trak->end_sample - trak->start_sample + 1;
2537
2538          ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2539                         "mp4 ctts crop end_sample:%uD", start_sample);
2540
2541      } else {
2542          return;
2543      }
2544
2545      data = trak->out[NGX_HTTP_MP4_CTTS_DATA].buf;
2546
2547      entries = trak->composition_offset_entries;
2548      entry = (ngx_mp4_ctts_entry_t *) data->pos;
2549      end = (ngx_mp4_ctts_entry_t *) data->last;
2550
2551      while (entry < end) {
2552          count = ngx_mp4_get_32value(entry->count);
2553
2554          ngx_log_debug3(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2555                         "sample:%uD, count:%uD, offset:%uD",
2556                         start_sample, count, ngx_mp4_get_32value(entry->offset));
2557
2558           if (start_sample <= count) {
2559               rest = start_sample - 1;
2560               goto found;
2561           }
2562
2563           start_sample -= count;
2564           entries--;
2565           entry++;
2566      }
2567
2568      if (start) {
2569          data->pos = (u_char *) end;
2570          trak->composition_offset_entries = 0;
2571      }
2572
2573      return;
2574
2575  found:
2576
2577      if (start) {
2578          ngx_mp4_set_32value(entry->count, count - rest);
2579          data->pos = (u_char *) entry;
2580          trak->composition_offset_entries = entries;
2581
2582      } else {
2583          ngx_mp4_set_32value(entry->count, rest);
2584          data->last = (u_char *) (entry + 1);
2585          trak->composition_offset_entries -= entries - 1;
2586      }
2587  }
2588
2589
2590  typedef struct {
2591      u_char    size[4];
2592      u_char    name[4];
2593      u_char    version[1];
2594      u_char    flags[3];
2595      u_char    entries[4];
```

```
} ngx_mp4_stsc_atom_t;


static ngx_int_t
ngx_http_mp4_read_stsc_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char                  *atom_header, *atom_table, *atom_end;
    uint32_t                 entries;
    ngx_buf_t               *atom, *data;
    ngx_mp4_stsc_atom_t     *stsc_atom;
    ngx_http_mp4_trak_t     *trak;

    /* sample-to-chunk atom */

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 stsc atom");

    atom_header = ngx_mp4_atom_header(mp4);
    stsc_atom = (ngx_mp4_stsc_atom_t *) atom_header;
    ngx_mp4_set_atom_name(stsc_atom, 's', 't', 's', 'c');

    if (ngx_mp4_atom_data_size(ngx_mp4_stsc_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 stsc atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    entries = ngx_mp4_get_32value(stsc_atom->entries);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "sample-to-chunk entries:%uD", entries);

    if (ngx_mp4_atom_data_size(ngx_mp4_stsc_atom_t)
        + entries * sizeof(ngx_mp4_stsc_entry_t) > atom_data_size)
    {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 stsc atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    atom_table = atom_header + sizeof(ngx_mp4_stsc_atom_t);
    atom_end = atom_table + entries * sizeof(ngx_mp4_stsc_entry_t);

    trak = ngx_mp4_last_trak(mp4);
    trak->sample_to_chunk_entries = entries;

    atom = &trak->stsc_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_table;

    data = &trak->stsc_data_buf;
    data->temporary = 1;
    data->pos = atom_table;
    data->last = atom_end;

    trak->out[NGX_HTTP_MP4_STSC_ATOM].buf = atom;
    trak->out[NGX_HTTP_MP4_STSC_DATA].buf = data;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_update_stsc_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak)
{
    size_t                   atom_size;
    uint32_t                 chunk;
    ngx_buf_t               *atom, *data;
    ngx_mp4_stsc_atom_t     *stsc_atom;
    ngx_mp4_stsc_entry_t    *entry, *end;

    /*
     * mdia.minf.stbl.stsc updating requires trak->start_sample
```

```
     * from mdia.minf.stbl.stts which depends on value from mdia.mdhd
     * atom which may reside after mdia.minf
     */

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "mp4 stsc atom update");

    data = trak->out[NGX_HTTP_MP4_STSC_DATA].buf;

    if (data == NULL) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "no mp4 stsc atoms were found in \"%s\"",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    if (trak->sample_to_chunk_entries == 0) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "zero number of entries in stsc atom in \"%s\"",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    if (ngx_http_mp4_crop_stsc_data(mp4, trak, 1) != NGX_OK) {
        return NGX_ERROR;
    }

    if (ngx_http_mp4_crop_stsc_data(mp4, trak, 0) != NGX_OK) {
        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "sample-to-chunk entries:%uD",
                   trak->sample_to_chunk_entries);

    entry = (ngx_mp4_stsc_entry_t *) data->pos;
    end = (ngx_mp4_stsc_entry_t *) data->last;

    while (entry < end) {
        chunk = ngx_mp4_get_32value(entry->chunk);
        chunk -= trak->start_chunk;
        ngx_mp4_set_32value(entry->chunk, chunk);
        entry++;
    }

    atom_size = sizeof(ngx_mp4_stsc_atom_t)
                + trak->sample_to_chunk_entries * sizeof(ngx_mp4_stsc_entry_t);

    trak->size += atom_size;

    atom = trak->out[NGX_HTTP_MP4_STSC_ATOM].buf;
    stsc_atom = (ngx_mp4_stsc_atom_t *) atom->pos;

    ngx_mp4_set_32value(stsc_atom->size, atom_size);
    ngx_mp4_set_32value(stsc_atom->entries, trak->sample_to_chunk_entries);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_crop_stsc_data(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak, ngx_uint_t start)
{
    uint32_t               start_sample, chunk, samples, id, next_chunk, n,
                           prev_samples;
    ngx_buf_t             *data, *buf;
    ngx_uint_t             entries, target_chunk, chunk_samples;
    ngx_mp4_stsc_entry_t  *entry, *end, *first;

    entries = trak->sample_to_chunk_entries - 1;

    if (start) {
        start_sample = (uint32_t) trak->start_sample;

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
```

```
2748                    "mp4 stsc crop start_sample:%uD", start_sample);
2749
2750    } else if (mp4->length) {
2751        start_sample = (uint32_t) (trak->end_sample - trak->start_sample);
2752        samples = 0;
2753
2754        data = trak->out[NGX_HTTP_MP4_STSC_START].buf;
2755
2756        if (data) {
2757            entry = (ngx_mp4_stsc_entry_t *) data->pos;
2758            samples = ngx_mp4_get_32value(entry->samples);
2759            entries--;
2760
2761            if (samples > start_sample) {
2762                samples = start_sample;
2763                ngx_mp4_set_32value(entry->samples, samples);
2764            }
2765
2766            start_sample -= samples;
2767        }
2768
2769        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2770                       "mp4 stsc crop end_sample:%uD, ext_samples:%uD",
2771                       start_sample, samples);
2772
2773    } else {
2774        return NGX_OK;
2775    }
2776
2777    data = trak->out[NGX_HTTP_MP4_STSC_DATA].buf;
2778
2779    entry = (ngx_mp4_stsc_entry_t *) data->pos;
2780    end = (ngx_mp4_stsc_entry_t *) data->last;
2781
2782    chunk = ngx_mp4_get_32value(entry->chunk);
2783    samples = ngx_mp4_get_32value(entry->samples);
2784    id = ngx_mp4_get_32value(entry->id);
2785    prev_samples = 0;
2786    entry++;
2787
2788    while (entry < end) {
2789
2790        next_chunk = ngx_mp4_get_32value(entry->chunk);
2791
2792        ngx_log_debug5(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2793                       "sample:%uD, chunk:%uD, chunks:%uD, "
2794                       "samples:%uD, id:%uD",
2795                       start_sample, chunk, next_chunk - chunk, samples, id);
2796
2797        n = (next_chunk - chunk) * samples;
2798
2799        if (start_sample < n) {
2800            goto found;
2801        }
2802
2803        start_sample -= n;
2804
2805        prev_samples = samples;
2806        chunk = next_chunk;
2807        samples = ngx_mp4_get_32value(entry->samples);
2808        id = ngx_mp4_get_32value(entry->id);
2809        entries--;
2810        entry++;
2811    }
2812
2813    next_chunk = trak->chunks + 1;
2814
2815    ngx_log_debug4(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
2816                   "sample:%uD, chunk:%uD, chunks:%uD, samples:%uD",
2817                   start_sample, chunk, next_chunk - chunk, samples);
2818
2819    n = (next_chunk - chunk) * samples;
2820
2821    if (start_sample > n) {
2822        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2823                      "%s time is out mp4 stsc chunks in \"%s\"",
```

```
                      start ? "start" : "end", mp4->file.name.data);
        return NGX_ERROR;
    }

found:

    entries++;
    entry--;

    if (samples == 0) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "zero number of samples in \"%s\"",
                      mp4->file.name.data);
        return NGX_ERROR;
    }

    target_chunk = chunk - 1;
    target_chunk += start_sample / samples;
    chunk_samples = start_sample % samples;

    if (start) {
        data->pos = (u_char *) entry;

        trak->sample_to_chunk_entries = entries;
        trak->start_chunk = target_chunk;
        trak->start_chunk_samples = chunk_samples;

        ngx_mp4_set_32value(entry->chunk, trak->start_chunk + 1);

        samples -= chunk_samples;

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                       "start_chunk:%ui, start_chunk_samples:%ui",
                       trak->start_chunk, trak->start_chunk_samples);

    } else {
        if (start_sample) {
            data->last = (u_char *) (entry + 1);
            trak->sample_to_chunk_entries -= entries - 1;
            trak->end_chunk_samples = samples;

        } else {
            data->last = (u_char *) entry;
            trak->sample_to_chunk_entries -= entries;
            trak->end_chunk_samples = prev_samples;
        }

        if (chunk_samples) {
            trak->end_chunk = target_chunk + 1;
            trak->end_chunk_samples = chunk_samples;

        } else {
            trak->end_chunk = target_chunk;
        }

        samples = chunk_samples;
        next_chunk = chunk + 1;

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                       "end_chunk:%ui, end_chunk_samples:%ui",
                       trak->end_chunk, trak->end_chunk_samples);
    }

    if (chunk_samples && next_chunk - target_chunk == 2) {

        ngx_mp4_set_32value(entry->samples, samples);

    } else if (chunk_samples && start) {

        first = &trak->stsc_start_chunk_entry;
        ngx_mp4_set_32value(first->chunk, 1);
        ngx_mp4_set_32value(first->samples, samples);
        ngx_mp4_set_32value(first->id, id);

        buf = &trak->stsc_start_chunk_buf;
        buf->temporary = 1;
```

```
        buf->pos = (u_char *) first;
        buf->last = (u_char *) first + sizeof(ngx_mp4_stsc_entry_t);

        trak->out[NGX_HTTP_MP4_STSC_START].buf = buf;

        ngx_mp4_set_32value(entry->chunk, trak->start_chunk + 2);

        trak->sample_to_chunk_entries++;

    } else if (chunk_samples) {

        first = &trak->stsc_end_chunk_entry;
        ngx_mp4_set_32value(first->chunk, trak->end_chunk - trak->start_chunk);
        ngx_mp4_set_32value(first->samples, samples);
        ngx_mp4_set_32value(first->id, id);

        buf = &trak->stsc_end_chunk_buf;
        buf->temporary = 1;
        buf->pos = (u_char *) first;
        buf->last = (u_char *) first + sizeof(ngx_mp4_stsc_entry_t);

        trak->out[NGX_HTTP_MP4_STSC_END].buf = buf;

        trak->sample_to_chunk_entries++;
    }

    return NGX_OK;
}


typedef struct {
    u_char    size[4];
    u_char    name[4];
    u_char    version[1];
    u_char    flags[3];
    u_char    uniform_size[4];
    u_char    entries[4];
} ngx_mp4_stsz_atom_t;


static ngx_int_t
ngx_http_mp4_read_stsz_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char               *atom_header, *atom_table, *atom_end;
    size_t                atom_size;
    uint32_t              entries, size;
    ngx_buf_t            *atom, *data;
    ngx_mp4_stsz_atom_t  *stsz_atom;
    ngx_http_mp4_trak_t  *trak;

    /* sample sizes atom */

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 stsz atom");

    atom_header = ngx_mp4_atom_header(mp4);
    stsz_atom = (ngx_mp4_stsz_atom_t *) atom_header;
    ngx_mp4_set_atom_name(stsz_atom, 's', 't', 's', 'z');

    if (ngx_mp4_atom_data_size(ngx_mp4_stsz_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 stsz atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    size = ngx_mp4_get_32value(stsz_atom->uniform_size);
    entries = ngx_mp4_get_32value(stsz_atom->entries);

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
                   "sample uniform size:%uD, entries:%uD", size, entries);

    trak = ngx_mp4_last_trak(mp4);
    trak->sample_sizes_entries = entries;

    atom_table = atom_header + sizeof(ngx_mp4_stsz_atom_t);

    atom = &trak->stsz_atom_buf;
```

```
2976      atom->temporary = 1;
2977      atom->pos = atom_header;
2978      atom->last = atom_table;
2979
2980      trak->out[NGX_HTTP_MP4_STSZ_ATOM].buf = atom;
2981
2982      if (size == 0) {
2983          if (ngx_mp4_atom_data_size(ngx_mp4_stsz_atom_t)
2984              + entries * sizeof(uint32_t) > atom_data_size)
2985          {
2986              ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
2987                            "\"%s\" mp4 stsz atom too small",
2988                            mp4->file.name.data);
2989              return NGX_ERROR;
2990          }
2991
2992          atom_end = atom_table + entries * sizeof(uint32_t);
2993
2994          data = &trak->stsz_data_buf;
2995          data->temporary = 1;
2996          data->pos = atom_table;
2997          data->last = atom_end;
2998
2999          trak->out[NGX_HTTP_MP4_STSZ_DATA].buf = data;
3000
3001      } else {
3002          /* if size != 0 then all samples are the same size */
3003          /* TODO : chunk samples */
3004          atom_size = sizeof(ngx_mp4_atom_header_t) + (size_t) atom_data_size;
3005          ngx_mp4_set_32value(atom_header, atom_size);
3006          trak->size += atom_size;
3007      }
3008
3009      ngx_mp4_atom_next(mp4, atom_data_size);
3010
3011      return NGX_OK;
3012  }
3013
3014
3015  static ngx_int_t
3016  ngx_http_mp4_update_stsz_atom(ngx_http_mp4_file_t *mp4,
3017      ngx_http_mp4_trak_t *trak)
3018  {
3019      size_t                atom_size;
3020      uint32_t             *pos, *end, entries;
3021      ngx_buf_t            *atom, *data;
3022      ngx_mp4_stsz_atom_t  *stsz_atom;
3023
3024      /*
3025       * mdia.minf.stbl.stsz updating requires trak->start_sample
3026       * from mdia.minf.stbl.stts which depends on value from mdia.mdhd
3027       * atom which may reside after mdia.minf
3028       */
3029
3030      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3031                     "mp4 stsz atom update");
3032
3033      data = trak->out[NGX_HTTP_MP4_STSZ_DATA].buf;
3034
3035      if (data) {
3036          entries = trak->sample_sizes_entries;
3037
3038          if (trak->start_sample > entries) {
3039              ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3040                            "start time is out mp4 stsz samples in \"%s\"",
3041                            mp4->file.name.data);
3042              return NGX_ERROR;
3043          }
3044
3045          entries -= trak->start_sample;
3046          data->pos += trak->start_sample * sizeof(uint32_t);
3047          end = (uint32_t *) data->pos;
3048
3049          for (pos = end - trak->start_chunk_samples; pos < end; pos++) {
3050              trak->start_chunk_samples_size += ngx_mp4_get_32value(pos);
3051          }
```

```
3052
3053        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3054                       "chunk samples sizes:%uL",
3055                       trak->start_chunk_samples_size);
3056
3057        if (mp4->length) {
3058            if (trak->end_sample - trak->start_sample > entries) {
3059                ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3060                              "end time is out mp4 stsz samples in \"%s\"",
3061                              mp4->file.name.data);
3062                return NGX_ERROR;
3063            }
3064
3065            entries = trak->end_sample - trak->start_sample;
3066            data->last = data->pos + entries * sizeof(uint32_t);
3067            end = (uint32_t *) data->last;
3068
3069            for (pos = end - trak->end_chunk_samples; pos < end; pos++) {
3070                trak->end_chunk_samples_size += ngx_mp4_get_32value(pos);
3071            }
3072
3073            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3074                           "mp4 stsz end_chunk_samples_size:%uL",
3075                           trak->end_chunk_samples_size);
3076        }
3077
3078        atom_size = sizeof(ngx_mp4_stsz_atom_t) + (data->last - data->pos);
3079        trak->size += atom_size;
3080
3081        atom = trak->out[NGX_HTTP_MP4_STSZ_ATOM].buf;
3082        stsz_atom = (ngx_mp4_stsz_atom_t *) atom->pos;
3083
3084        ngx_mp4_set_32value(stsz_atom->size, atom_size);
3085        ngx_mp4_set_32value(stsz_atom->entries, entries);
3086    }
3087
3088    return NGX_OK;
3089 }
3090
3091
3092 typedef struct {
3093     u_char    size[4];
3094     u_char    name[4];
3095     u_char    version[1];
3096     u_char    flags[3];
3097     u_char    entries[4];
3098 } ngx_mp4_stco_atom_t;
3099
3100
3101 static ngx_int_t
3102 ngx_http_mp4_read_stco_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
3103 {
3104     u_char               *atom_header, *atom_table, *atom_end;
3105     uint32_t              entries;
3106     ngx_buf_t            *atom, *data;
3107     ngx_mp4_stco_atom_t  *stco_atom;
3108     ngx_http_mp4_trak_t  *trak;
3109
3110     /* chunk offsets atom */
3111
3112     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 stco atom");
3113
3114     atom_header = ngx_mp4_atom_header(mp4);
3115     stco_atom = (ngx_mp4_stco_atom_t *) atom_header;
3116     ngx_mp4_set_atom_name(stco_atom, 's', 't', 'c', 'o');
3117
3118     if (ngx_mp4_atom_data_size(ngx_mp4_stco_atom_t) > atom_data_size) {
3119         ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3120                       "\"%s\" mp4 stco atom too small", mp4->file.name.data);
3121         return NGX_ERROR;
3122     }
3123
3124     entries = ngx_mp4_get_32value(stco_atom->entries);
3125
3126     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "chunks:%uD", entries);
3127
```

```
3128      if (ngx_mp4_atom_data_size(ngx_mp4_stco_atom_t)
3129          + entries * sizeof(uint32_t) > atom_data_size)
3130      {
3131          ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3132                        "\"%s\" mp4 stco atom too small", mp4->file.name.data);
3133          return NGX_ERROR;
3134      }
3135
3136      atom_table = atom_header + sizeof(ngx_mp4_stco_atom_t);
3137      atom_end = atom_table + entries * sizeof(uint32_t);
3138
3139      trak = ngx_mp4_last_trak(mp4);
3140      trak->chunks = entries;
3141
3142      atom = &trak->stco_atom_buf;
3143      atom->temporary = 1;
3144      atom->pos = atom_header;
3145      atom->last = atom_table;
3146
3147      data = &trak->stco_data_buf;
3148      data->temporary = 1;
3149      data->pos = atom_table;
3150      data->last = atom_end;
3151
3152      trak->out[NGX_HTTP_MP4_STCO_ATOM].buf = atom;
3153      trak->out[NGX_HTTP_MP4_STCO_DATA].buf = data;
3154
3155      ngx_mp4_atom_next(mp4, atom_data_size);
3156
3157      return NGX_OK;
3158  }
3159
3160
3161  static ngx_int_t
3162  ngx_http_mp4_update_stco_atom(ngx_http_mp4_file_t *mp4,
3163      ngx_http_mp4_trak_t *trak)
3164  {
3165      size_t                atom_size;
3166      uint32_t              entries;
3167      ngx_buf_t            *atom, *data;
3168      ngx_mp4_stco_atom_t  *stco_atom;
3169
3170      /*
3171       * mdia.minf.stbl.stco updating requires trak->start_chunk
3172       * from mdia.minf.stbl.stsc which depends on value from mdia.mdhd
3173       * atom which may reside after mdia.minf
3174       */
3175
3176      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3177                     "mp4 stco atom update");
3178
3179      data = trak->out[NGX_HTTP_MP4_STCO_DATA].buf;
3180
3181      if (data == NULL) {
3182          ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3183                        "no mp4 stco atoms were found in \"%s\"",
3184                        mp4->file.name.data);
3185          return NGX_ERROR;
3186      }
3187
3188      if (trak->start_chunk > trak->chunks) {
3189          ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3190                        "start time is out mp4 stco chunks in \"%s\"",
3191                        mp4->file.name.data);
3192          return NGX_ERROR;
3193      }
3194
3195      data->pos += trak->start_chunk * sizeof(uint32_t);
3196
3197      trak->start_offset = ngx_mp4_get_32value(data->pos);
3198      trak->start_offset += trak->start_chunk_samples_size;
3199      ngx_mp4_set_32value(data->pos, trak->start_offset);
3200
3201      ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3202                     "start chunk offset:%O", trak->start_offset);
3203
```

```
3204      if (mp4->length) {
3205
3206          if (trak->end_chunk > trak->chunks) {
3207              ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3208                            "end time is out mp4 stco chunks in \"%s\"",
3209                            mp4->file.name.data);
3210              return NGX_ERROR;
3211          }
3212
3213          entries = trak->end_chunk - trak->start_chunk;
3214          data->last = data->pos + entries * sizeof(uint32_t);
3215
3216          if (entries) {
3217              trak->end_offset =
3218                              ngx_mp4_get_32value(data->last - sizeof(uint32_t));
3219              trak->end_offset += trak->end_chunk_samples_size;
3220
3221              ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3222                             "end chunk offset:%O", trak->end_offset);
3223          }
3224
3225      } else {
3226          entries = trak->chunks - trak->start_chunk;
3227          trak->end_offset = mp4->mdat_data.buf->file_last;
3228      }
3229
3230      if (entries == 0) {
3231          trak->start_offset = mp4->end;
3232          trak->end_offset = 0;
3233      }
3234
3235      atom_size = sizeof(ngx_mp4_stco_atom_t) + (data->last - data->pos);
3236      trak->size += atom_size;
3237
3238      atom = trak->out[NGX_HTTP_MP4_STCO_ATOM].buf;
3239      stco_atom = (ngx_mp4_stco_atom_t *) atom->pos;
3240
3241      ngx_mp4_set_32value(stco_atom->size, atom_size);
3242      ngx_mp4_set_32value(stco_atom->entries, entries);
3243
3244      return NGX_OK;
3245  }
3246
3247
3248  static void
3249  ngx_http_mp4_adjust_stco_atom(ngx_http_mp4_file_t *mp4,
3250      ngx_http_mp4_trak_t *trak, int32_t adjustment)
3251  {
3252      uint32_t    offset, *entry, *end;
3253      ngx_buf_t  *data;
3254
3255      /*
3256       * moov.trak.mdia.minf.stbl.stco adjustment requires
3257       * minimal start offset of all traks and new moov atom size
3258       */
3259
3260      ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3261                     "mp4 stco atom adjustment");
3262
3263      data = trak->out[NGX_HTTP_MP4_STCO_DATA].buf;
3264      entry = (uint32_t *) data->pos;
3265      end = (uint32_t *) data->last;
3266
3267      while (entry < end) {
3268          offset = ngx_mp4_get_32value(entry);
3269          offset += adjustment;
3270          ngx_mp4_set_32value(entry, offset);
3271          entry++;
3272      }
3273  }
3274
3275
3276  typedef struct {
3277      u_char    size[4];
3278      u_char    name[4];
3279      u_char    version[1];
```

```
    u_char    flags[3];
    u_char    entries[4];
} ngx_mp4_co64_atom_t;


static ngx_int_t
ngx_http_mp4_read_co64_atom(ngx_http_mp4_file_t *mp4, uint64_t atom_data_size)
{
    u_char               *atom_header, *atom_table, *atom_end;
    uint32_t              entries;
    ngx_buf_t            *atom, *data;
    ngx_mp4_co64_atom_t  *co64_atom;
    ngx_http_mp4_trak_t  *trak;

    /* chunk offsets atom */

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "mp4 co64 atom");

    atom_header = ngx_mp4_atom_header(mp4);
    co64_atom = (ngx_mp4_co64_atom_t *) atom_header;
    ngx_mp4_set_atom_name(co64_atom, 'c', 'o', '6', '4');

    if (ngx_mp4_atom_data_size(ngx_mp4_co64_atom_t) > atom_data_size) {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 co64 atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    entries = ngx_mp4_get_32value(co64_atom->entries);

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0, "chunks:%uD", entries);

    if (ngx_mp4_atom_data_size(ngx_mp4_co64_atom_t)
        + entries * sizeof(uint64_t) > atom_data_size)
    {
        ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
                      "\"%s\" mp4 co64 atom too small", mp4->file.name.data);
        return NGX_ERROR;
    }

    atom_table = atom_header + sizeof(ngx_mp4_co64_atom_t);
    atom_end = atom_table + entries * sizeof(uint64_t);

    trak = ngx_mp4_last_trak(mp4);
    trak->chunks = entries;

    atom = &trak->co64_atom_buf;
    atom->temporary = 1;
    atom->pos = atom_header;
    atom->last = atom_table;

    data = &trak->co64_data_buf;
    data->temporary = 1;
    data->pos = atom_table;
    data->last = atom_end;

    trak->out[NGX_HTTP_MP4_CO64_ATOM].buf = atom;
    trak->out[NGX_HTTP_MP4_CO64_DATA].buf = data;

    ngx_mp4_atom_next(mp4, atom_data_size);

    return NGX_OK;
}


static ngx_int_t
ngx_http_mp4_update_co64_atom(ngx_http_mp4_file_t *mp4,
    ngx_http_mp4_trak_t *trak)
{
    size_t                atom_size;
    uint64_t              entries;
    ngx_buf_t            *atom, *data;
    ngx_mp4_co64_atom_t  *co64_atom;

    /*
     * mdia.minf.stbl.co64 updating requires trak->start_chunk
```

```
3356        * from mdia.minf.stbl.stsc which depends on value from mdia.mdhd
3357        * atom which may reside after mdia.minf
3358        */
3359
3360       ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3361                      "mp4 co64 atom update");
3362
3363       data = trak->out[NGX_HTTP_MP4_CO64_DATA].buf;
3364
3365       if (data == NULL) {
3366           ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3367                         "no mp4 co64 atoms were found in \"%s\"",
3368                         mp4->file.name.data);
3369           return NGX_ERROR;
3370       }
3371
3372       if (trak->start_chunk > trak->chunks) {
3373           ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3374                         "start time is out mp4 co64 chunks in \"%s\"",
3375                         mp4->file.name.data);
3376           return NGX_ERROR;
3377       }
3378
3379       data->pos += trak->start_chunk * sizeof(uint64_t);
3380
3381       trak->start_offset = ngx_mp4_get_64value(data->pos);
3382       trak->start_offset += trak->start_chunk_samples_size;
3383       ngx_mp4_set_64value(data->pos, trak->start_offset);
3384
3385       ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3386                      "start chunk offset:%O", trak->start_offset);
3387
3388       if (mp4->length) {
3389
3390           if (trak->end_chunk > trak->chunks) {
3391               ngx_log_error(NGX_LOG_ERR, mp4->file.log, 0,
3392                             "end time is out mp4 co64 chunks in \"%s\"",
3393                             mp4->file.name.data);
3394               return NGX_ERROR;
3395           }
3396
3397           entries = trak->end_chunk - trak->start_chunk;
3398           data->last = data->pos + entries * sizeof(uint64_t);
3399
3400           if (entries) {
3401               trak->end_offset =
3402                               ngx_mp4_get_64value(data->last - sizeof(uint64_t));
3403               trak->end_offset += trak->end_chunk_samples_size;
3404
3405               ngx_log_debug1(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3406                              "end chunk offset:%O", trak->end_offset);
3407           }
3408
3409       } else {
3410           entries = trak->chunks - trak->start_chunk;
3411           trak->end_offset = mp4->mdat_data.buf->file_last;
3412       }
3413
3414       if (entries == 0) {
3415           trak->start_offset = mp4->end;
3416           trak->end_offset = 0;
3417       }
3418
3419       atom_size = sizeof(ngx_mp4_co64_atom_t) + (data->last - data->pos);
3420       trak->size += atom_size;
3421
3422       atom = trak->out[NGX_HTTP_MP4_CO64_ATOM].buf;
3423       co64_atom = (ngx_mp4_co64_atom_t *) atom->pos;
3424
3425       ngx_mp4_set_32value(co64_atom->size, atom_size);
3426       ngx_mp4_set_32value(co64_atom->entries, entries);
3427
3428       return NGX_OK;
3429   }
3430
3431
```

```
3432 static void
3433 ngx_http_mp4_adjust_co64_atom(ngx_http_mp4_file_t *mp4,
3434     ngx_http_mp4_trak_t *trak, off_t adjustment)
3435 {
3436     uint64_t    offset, *entry, *end;
3437     ngx_buf_t  *data;
3438
3439     /*
3440      * moov.trak.mdia.minf.stbl.co64 adjustment requires
3441      * minimal start offset of all traks and new moov atom size
3442      */
3443
3444     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, mp4->file.log, 0,
3445                    "mp4 co64 atom adjustment");
3446
3447     data = trak->out[NGX_HTTP_MP4_CO64_DATA].buf;
3448     entry = (uint64_t *) data->pos;
3449     end = (uint64_t *) data->last;
3450
3451     while (entry < end) {
3452         offset = ngx_mp4_get_64value(entry);
3453         offset += adjustment;
3454         ngx_mp4_set_64value(entry, offset);
3455         entry++;
3456     }
3457 }
3458
3459
3460 static char *
3461 ngx_http_mp4(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3462 {
3463     ngx_http_core_loc_conf_t  *clcf;
3464
3465     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
3466     clcf->handler = ngx_http_mp4_handler;
3467
3468     return NGX_CONF_OK;
3469 }
3470
3471
3472 static void *
3473 ngx_http_mp4_create_conf(ngx_conf_t *cf)
3474 {
3475     ngx_http_mp4_conf_t  *conf;
3476
3477     conf = ngx_palloc(cf->pool, sizeof(ngx_http_mp4_conf_t));
3478     if (conf == NULL) {
3479         return NULL;
3480     }
3481
3482     conf->buffer_size = NGX_CONF_UNSET_SIZE;
3483     conf->max_buffer_size = NGX_CONF_UNSET_SIZE;
3484
3485     return conf;
3486 }
3487
3488
3489 static char *
3490 ngx_http_mp4_merge_conf(ngx_conf_t *cf, void *parent, void *child)
3491 {
3492     ngx_http_mp4_conf_t *prev = parent;
3493     ngx_http_mp4_conf_t *conf = child;
3494
3495     ngx_conf_merge_size_value(conf->buffer_size, prev->buffer_size, 512 * 1024);
3496     ngx_conf_merge_size_value(conf->max_buffer_size, prev->max_buffer_size,
3497                               10 * 1024 * 1024);
3498
3499     return NGX_CONF_OK;
3500 }
```

One Level Up Top Level

src/http/modules/ngx_http_not_modified_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_header_filter
- ngx_http_not_modified_filter_module
- ngx_http_not_modified_filter_module_ctx

Functions defined

- ngx_http_not_modified_filter_init
- ngx_http_not_modified_header_filter
- ngx_http_test_if_match
- ngx_http_test_if_modified
- ngx_http_test_if_unmodified

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static ngx_uint_t ngx_http_test_if_unmodified(ngx_http_request_t *r);
14  static ngx_uint_t ngx_http_test_if_modified(ngx_http_request_t *r);
15  static ngx_uint_t ngx_http_test_if_match(ngx_http_request_t *r,
16      ngx_table_elt_t *header, ngx_uint_t weak);
17  static ngx_int_t ngx_http_not_modified_filter_init(ngx_conf_t *cf);
18
19
20  static ngx_http_module_t  ngx_http_not_modified_filter_module_ctx = {
21      NULL,                                  /* preconfiguration */
22      ngx_http_not_modified_filter_init,     /* postconfiguration */
23
24      NULL,                                  /* create main configuration */
25      NULL,                                  /* init main configuration */
26
27      NULL,                                  /* create server configuration */
28      NULL,                                  /* merge server configuration */
29
30      NULL,                                  /* create location configuration */
31      NULL                                   /* merge location configuration */
32  };
33
34
35  ngx_module_t  ngx_http_not_modified_filter_module = {
36      NGX_MODULE_V1,
37      &ngx_http_not_modified_filter_module_ctx, /* module context */
38      NULL,                                  /* module directives */
39      NGX_HTTP_MODULE,                       /* module type */
40      NULL,                                  /* init master */
41      NULL,                                  /* init module */
42      NULL,                                  /* init process */
43      NULL,                                  /* init thread */
```

```
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;


static ngx_int_t
ngx_http_not_modified_header_filter(ngx_http_request_t *r)
{
    if (r->headers_out.status != NGX_HTTP_OK
        || r != r->main
        || r->disable_not_modified)
    {
        return ngx_http_next_header_filter(r);
    }

    if (r->headers_in.if_unmodified_since
        && !ngx_http_test_if_unmodified(r))
    {
        return ngx_http_filter_finalize_request(r, NULL,
                                                NGX_HTTP_PRECONDITION_FAILED);
    }

    if (r->headers_in.if_match
        && !ngx_http_test_if_match(r, r->headers_in.if_match, 0))
    {
        return ngx_http_filter_finalize_request(r, NULL,
                                                NGX_HTTP_PRECONDITION_FAILED);
    }

    if (r->headers_in.if_modified_since || r->headers_in.if_none_match) {

        if (r->headers_in.if_modified_since
            && ngx_http_test_if_modified(r))
        {
            return ngx_http_next_header_filter(r);
        }

        if (r->headers_in.if_none_match
            && !ngx_http_test_if_match(r, r->headers_in.if_none_match, 1))
        {
            return ngx_http_next_header_filter(r);
        }

        /* not modified */

        r->headers_out.status = NGX_HTTP_NOT_MODIFIED;
        r->headers_out.status_line.len = 0;
        r->headers_out.content_type.len = 0;
        ngx_http_clear_content_length(r);
        ngx_http_clear_accept_ranges(r);

         if (r->headers_out.content_encoding) {
             r->headers_out.content_encoding->hash = 0;
             r->headers_out.content_encoding = NULL;
         }

         return ngx_http_next_header_filter(r);
     }

    return ngx_http_next_header_filter(r);
}


 static ngx_uint_t
 ngx_http_test_if_unmodified(ngx_http_request_t *r)
 {
    time_t  iums;

    if (r->headers_out.last_modified_time == (time_t) -1) {
        return 0;
    }
```

```
120
121     ims = ngx_http_parse_time(r->headers_in.if_unmodified_since->value.data,
122                               r->headers_in.if_unmodified_since->value.len);
123
124     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
125                    "http ims:%T lm:%T", ims, r->headers_out.last_modified_time);
126
127     if (ims >= r->headers_out.last_modified_time) {
128         return 1;
129     }
130
131     return 0;
132 }
133
134
135 static ngx_uint_t
136 ngx_http_test_if_modified(ngx_http_request_t *r)
137 {
138     time_t                     ims;
139     ngx_http_core_loc_conf_t  *clcf;
140
141     if (r->headers_out.last_modified_time == (time_t) -1) {
142         return 1;
143     }
144
145     clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);
146
147     if (clcf->if_modified_since == NGX_HTTP_IMS_OFF) {
148         return 1;
149     }
150
151     ims = ngx_http_parse_time(r->headers_in.if_modified_since->value.data,
152                               r->headers_in.if_modified_since->value.len);
153
154     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
155                    "http ims:%T lm:%T", ims, r->headers_out.last_modified_time);
156
157     if (ims == r->headers_out.last_modified_time) {
158         return 0;
159     }
160
161     if (clcf->if_modified_since == NGX_HTTP_IMS_EXACT
162         || ims < r->headers_out.last_modified_time)
163     {
164         return 1;
165     }
166
167     return 0;
168 }
169
170
171 static ngx_uint_t
172 ngx_http_test_if_match(ngx_http_request_t *r, ngx_table_elt_t *header,
173     ngx_uint_t weak)
174 {
175     u_char     *start, *end, ch;
176     ngx_str_t   etag, *list;
177
178     list = &header->value;
179
180     if (list->len == 1 && list->data[0] == '*') {
181         return 1;
182     }
183
184     if (r->headers_out.etag == NULL) {
185         return 0;
186     }
187
188     etag = r->headers_out.etag->value;
189
190     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
191                    "http im:\"%V\" etag:%V", list, &etag);
192
193     if (weak
194         && etag.len > 2
195         && etag.data[0] == 'W'
```

```
196         && etag.data[1] == '/')
197     {
198         etag.len -= 2;
199         etag.data += 2;
200     }
201 
202     start = list->data;
203     end = list->data + list->len;
204 
205     while (start < end) {
206 
207         if (weak
208             && end - start > 2
209             && start[0] == 'W'
210             && start[1] == '/')
211         {
212             start += 2;
213         }
214 
215         if (etag.len > (size_t) (end - start)) {
216             return 0;
217         }
218 
219         if (ngx_strncmp(start, etag.data, etag.len) != 0) {
220             goto skip;
221         }
222 
223         start += etag.len;
224 
225         while (start < end) {
226             ch = *start;
227 
228             if (ch == ' ' || ch == '\t') {
229                 start++;
230                 continue;
231             }
232 
233             break;
234         }
235 
236         if (start == end || *start == ',') {
237             return 1;
238         }
239 
240     skip:
241 
242         while (start < end && *start != ',') { start++; }
243         while (start < end) {
244             ch = *start;
245 
246             if (ch == ' ' || ch == '\t' || ch == ',') {
247                 start++;
248                 continue;
249             }
250 
251             break;
252         }
253     }
254 
255     return 0;
256 }
257 
258 
259 static ngx_int_t
260 ngx_http_not_modified_filter_init(ngx_conf_t *cf)
261 {
262     ngx_http_next_header_filter = ngx_http_top_header_filter;
263     ngx_http_top_header_filter = ngx_http_not_modified_header_filter;
264 
265     return NGX_OK;
266 }
```

One Level Up Top Level

src/http/modules/ngx_http_proxy_module.c - nginx-1.7.10

Global variables defined

- ngx_http_proxy_cache_headers
- ngx_http_proxy_commands
- ngx_http_proxy_headers
- ngx_http_proxy_hide_headers
- ngx_http_proxy_http_version
- ngx_http_proxy_lowat_post
- ngx_http_proxy_module
- ngx_http_proxy_module
- ngx_http_proxy_module_ctx
- ngx_http_proxy_next_upstream_masks
- ngx_http_proxy_ssl_protocols
- ngx_http_proxy_temp_path
- ngx_http_proxy_vars
- ngx_http_proxy_version
- ngx_http_proxy_version_11

Data types defined

- ngx_http_proxy_ctx_t
- ngx_http_proxy_headers_t
- ngx_http_proxy_loc_conf_t
- ngx_http_proxy_main_conf_t
- ngx_http_proxy_rewrite_pt
- ngx_http_proxy_rewrite_s
- ngx_http_proxy_rewrite_t
- ngx_http_proxy_vars_t

Functions defined

- ngx_http_proxy_abort_request
- ngx_http_proxy_add_variables
- ngx_http_proxy_add_x_forwarded_for_variable
- ngx_http_proxy_cache

- ngx_http_proxy_cache_key
- ngx_http_proxy_chunked_filter
- ngx_http_proxy_cookie_domain
- ngx_http_proxy_cookie_path
- ngx_http_proxy_copy_filter
- ngx_http_proxy_create_key
- ngx_http_proxy_create_loc_conf
- ngx_http_proxy_create_main_conf
- ngx_http_proxy_create_request
- ngx_http_proxy_eval
- ngx_http_proxy_finalize_request
- ngx_http_proxy_handler
- ngx_http_proxy_host_variable
- ngx_http_proxy_init_headers
- ngx_http_proxy_input_filter_init
- ngx_http_proxy_internal_body_length_variable
- ngx_http_proxy_lowat_check
- ngx_http_proxy_merge_loc_conf
- ngx_http_proxy_non_buffered_chunked_filter
- ngx_http_proxy_non_buffered_copy_filter
- ngx_http_proxy_pass
- ngx_http_proxy_port_variable
- ngx_http_proxy_process_header
- ngx_http_proxy_process_status_line
- ngx_http_proxy_redirect
- ngx_http_proxy_reinit_request
- ngx_http_proxy_rewrite
- ngx_http_proxy_rewrite_complex_handler
- ngx_http_proxy_rewrite_cookie
- ngx_http_proxy_rewrite_cookie_value
- ngx_http_proxy_rewrite_domain_handler
- ngx_http_proxy_rewrite_redirect
- ngx_http_proxy_rewrite_regex

- ngx_http_proxy_rewrite_regex_handler
- ngx_http_proxy_set_ssl
- ngx_http_proxy_set_vars
- ngx_http_proxy_ssl_password_file
- ngx_http_proxy_store

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_array_t                    caches;  /* ngx_http_file_cache_t * */
15  } ngx_http_proxy_main_conf_t;
16
17
18  typedef struct ngx_http_proxy_rewrite_s  ngx_http_proxy_rewrite_t;
19
20  typedef ngx_int_t (*ngx_http_proxy_rewrite_pt)(ngx_http_request_t *r,
21      ngx_table_elt_t *h, size_t prefix, size_t len,
22      ngx_http_proxy_rewrite_t *pr);
23
24  struct ngx_http_proxy_rewrite_s {
25      ngx_http_proxy_rewrite_pt      handler;
26
27      union {
28          ngx_http_complex_value_t   complex;
29  #if (NGX_PCRE)
30          ngx_http_regex_t          *regex;
31  #endif
32      } pattern;
33
34      ngx_http_complex_value_t       replacement;
35  };
36
37
38  typedef struct {
39      ngx_str_t                      key_start;
40      ngx_str_t                      schema;
41      ngx_str_t                      host_header;
42      ngx_str_t                      port;
43      ngx_str_t                      uri;
44  } ngx_http_proxy_vars_t;
45
46
47  typedef struct {
48      ngx_array_t                   *flushes;
49      ngx_array_t                   *lengths;
50      ngx_array_t                   *values;
51      ngx_hash_t                     hash;
52  } ngx_http_proxy_headers_t;
53
54
55  typedef struct {
56      ngx_http_upstream_conf_t       upstream;
57
58      ngx_array_t                   *body_flushes;
59      ngx_array_t                   *body_lengths;
60      ngx_array_t                   *body_values;
61      ngx_str_t                      body_source;
```

```
62
63      ngx_http_proxy_headers_t       headers;
64  #if (NGX_HTTP_CACHE)
65      ngx_http_proxy_headers_t       headers_cache;
66  #endif
67      ngx_array_t                   *headers_source;
68
69      ngx_array_t                   *proxy_lengths;
70      ngx_array_t                   *proxy_values;
71
72      ngx_array_t                   *redirects;
73      ngx_array_t                   *cookie_domains;
74      ngx_array_t                   *cookie_paths;
75
76      ngx_str_t                      method;
77      ngx_str_t                      location;
78      ngx_str_t                      url;
79
80  #if (NGX_HTTP_CACHE)
81      ngx_http_complex_value_t       cache_key;
82  #endif
83
84      ngx_http_proxy_vars_t          vars;
85
86      ngx_flag_t                     redirect;
87
88      ngx_uint_t                     http_version;
89
90      ngx_uint_t                     headers_hash_max_size;
91      ngx_uint_t                     headers_hash_bucket_size;
92
93  #if (NGX_HTTP_SSL)
94      ngx_uint_t                     ssl;
95      ngx_uint_t                     ssl_protocols;
96      ngx_str_t                      ssl_ciphers;
97      ngx_uint_t                     ssl_verify_depth;
98      ngx_str_t                      ssl_trusted_certificate;
99      ngx_str_t                      ssl_crl;
100      ngx_str_t                      ssl_certificate;
101      ngx_str_t                      ssl_certificate_key;
102      ngx_array_t                   *ssl_passwords;
103  #endif
104  } ngx_http_proxy_loc_conf_t;
105
106
107  typedef struct {
108      ngx_http_status_t              status;
109      ngx_http_chunked_t             chunked;
110      ngx_http_proxy_vars_t          vars;
111      off_t                          internal_body_length;
112
113      ngx_uint_t                     head;  /* unsigned  head:1 */
114  } ngx_http_proxy_ctx_t;
115
116
117  static ngx_int_t ngx_http_proxy_eval(ngx_http_request_t *r,
118      ngx_http_proxy_ctx_t *ctx, ngx_http_proxy_loc_conf_t *plcf);
119  #if (NGX_HTTP_CACHE)
120  static ngx_int_t ngx_http_proxy_create_key(ngx_http_request_t *r);
121  #endif
122  static ngx_int_t ngx_http_proxy_create_request(ngx_http_request_t *r);
123  static ngx_int_t ngx_http_proxy_reinit_request(ngx_http_request_t *r);
124  static ngx_int_t ngx_http_proxy_process_status_line(ngx_http_request_t *r);
125  static ngx_int_t ngx_http_proxy_process_header(ngx_http_request_t *r);
126  static ngx_int_t ngx_http_proxy_input_filter_init(void *data);
127  static ngx_int_t ngx_http_proxy_copy_filter(ngx_event_pipe_t *p,
128      ngx_buf_t *buf);
129  static ngx_int_t ngx_http_proxy_chunked_filter(ngx_event_pipe_t *p,
130      ngx_buf_t *buf);
131  static ngx_int_t ngx_http_proxy_non_buffered_copy_filter(void *data,
132      ssize_t bytes);
133  static ngx_int_t ngx_http_proxy_non_buffered_chunked_filter(void *data,
134      ssize_t bytes);
135  static void ngx_http_proxy_abort_request(ngx_http_request_t *r);
136  static void ngx_http_proxy_finalize_request(ngx_http_request_t *r,
137      ngx_int_t rc);
```

```
138
139 static ngx_int_t ngx_http_proxy_host_variable(ngx_http_request_t *r,
140     ngx_http_variable_value_t *v, uintptr_t data);
141 static ngx_int_t ngx_http_proxy_port_variable(ngx_http_request_t *r,
142     ngx_http_variable_value_t *v, uintptr_t data);
143 static ngx_int_t
144     ngx_http_proxy_add_x_forwarded_for_variable(ngx_http_request_t *r,
145     ngx_http_variable_value_t *v, uintptr_t data);
146 static ngx_int_t
147     ngx_http_proxy_internal_body_length_variable(ngx_http_request_t *r,
148     ngx_http_variable_value_t *v, uintptr_t data);
149 static ngx_int_t ngx_http_proxy_rewrite_redirect(ngx_http_request_t *r,
150     ngx_table_elt_t *h, size_t prefix);
151 static ngx_int_t ngx_http_proxy_rewrite_cookie(ngx_http_request_t *r,
152     ngx_table_elt_t *h);
153 static ngx_int_t ngx_http_proxy_rewrite_cookie_value(ngx_http_request_t *r,
154     ngx_table_elt_t *h, u_char *value, ngx_array_t *rewrites);
155 static ngx_int_t ngx_http_proxy_rewrite(ngx_http_request_t *r,
156     ngx_table_elt_t *h, size_t prefix, size_t len, ngx_str_t *replacement);
157
158 static ngx_int_t ngx_http_proxy_add_variables(ngx_conf_t *cf);
159 static void *ngx_http_proxy_create_main_conf(ngx_conf_t *cf);
160 static void *ngx_http_proxy_create_loc_conf(ngx_conf_t *cf);
161 static char *ngx_http_proxy_merge_loc_conf(ngx_conf_t *cf,
162     void *parent, void *child);
163 static ngx_int_t ngx_http_proxy_init_headers(ngx_conf_t *cf,
164     ngx_http_proxy_loc_conf_t *conf, ngx_http_proxy_headers_t *headers,
165     ngx_keyval_t *default_headers);
166
167 static char *ngx_http_proxy_pass(ngx_conf_t *cf, ngx_command_t *cmd,
168     void *conf);
169 static char *ngx_http_proxy_redirect(ngx_conf_t *cf, ngx_command_t *cmd,
170     void *conf);
171 static char *ngx_http_proxy_cookie_domain(ngx_conf_t *cf, ngx_command_t *cmd,
172     void *conf);
173 static char *ngx_http_proxy_cookie_path(ngx_conf_t *cf, ngx_command_t *cmd,
174     void *conf);
175 static char *ngx_http_proxy_store(ngx_conf_t *cf, ngx_command_t *cmd,
176     void *conf);
177 #if (NGX_HTTP_CACHE)
178 static char *ngx_http_proxy_cache(ngx_conf_t *cf, ngx_command_t *cmd,
179     void *conf);
180 static char *ngx_http_proxy_cache_key(ngx_conf_t *cf, ngx_command_t *cmd,
181     void *conf);
182 #endif
183 #if (NGX_HTTP_SSL)
184 static char *ngx_http_proxy_ssl_password_file(ngx_conf_t *cf,
185     ngx_command_t *cmd, void *conf);
186 #endif
187
188 static char *ngx_http_proxy_lowat_check(ngx_conf_t *cf, void *post, void *data);
189
190 static ngx_int_t ngx_http_proxy_rewrite_regex(ngx_conf_t *cf,
191     ngx_http_proxy_rewrite_t *pr, ngx_str_t *regex, ngx_uint_t caseless);
192
193 #if (NGX_HTTP_SSL)
194 static ngx_int_t ngx_http_proxy_set_ssl(ngx_conf_t *cf,
195     ngx_http_proxy_loc_conf_t *plcf);
196 #endif
197 static void ngx_http_proxy_set_vars(ngx_url_t *u, ngx_http_proxy_vars_t *v);
198
199
200 static ngx_conf_post_t  ngx_http_proxy_lowat_post =
201     { ngx_http_proxy_lowat_check };
202
203
204 static ngx_conf_bitmask_t  ngx_http_proxy_next_upstream_masks[] = {
205     { ngx_string("error"), NGX_HTTP_UPSTREAM_FT_ERROR },
206     { ngx_string("timeout"), NGX_HTTP_UPSTREAM_FT_TIMEOUT },
207     { ngx_string("invalid_header"), NGX_HTTP_UPSTREAM_FT_INVALID_HEADER },
208     { ngx_string("http_500"), NGX_HTTP_UPSTREAM_FT_HTTP_500 },
209     { ngx_string("http_502"), NGX_HTTP_UPSTREAM_FT_HTTP_502 },
210     { ngx_string("http_503"), NGX_HTTP_UPSTREAM_FT_HTTP_503 },
211     { ngx_string("http_504"), NGX_HTTP_UPSTREAM_FT_HTTP_504 },
212     { ngx_string("http_403"), NGX_HTTP_UPSTREAM_FT_HTTP_403 },
213     { ngx_string("http_404"), NGX_HTTP_UPSTREAM_FT_HTTP_404 },
```

```
    { ngx_string("updating"), NGX_HTTP_UPSTREAM_FT_UPDATING },
    { ngx_string("off"), NGX_HTTP_UPSTREAM_FT_OFF },
    { ngx_null_string, 0 }
};


#if (NGX_HTTP_SSL)

static ngx_conf_bitmask_t  ngx_http_proxy_ssl_protocols[] = {
    { ngx_string("SSLv2"), NGX_SSL_SSLv2 },
    { ngx_string("SSLv3"), NGX_SSL_SSLv3 },
    { ngx_string("TLSv1"), NGX_SSL_TLSv1 },
    { ngx_string("TLSv1.1"), NGX_SSL_TLSv1_1 },
    { ngx_string("TLSv1.2"), NGX_SSL_TLSv1_2 },
    { ngx_null_string, 0 }
};

#endif


static ngx_conf_enum_t  ngx_http_proxy_http_version[] = {
    { ngx_string("1.0"), NGX_HTTP_VERSION_10 },
    { ngx_string("1.1"), NGX_HTTP_VERSION_11 },
    { ngx_null_string, 0 }
};


ngx_module_t  ngx_http_proxy_module;


static ngx_command_t  ngx_http_proxy_commands[] = {

    { ngx_string("proxy_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_HTTP_LMT_CONF|NGX_CONF_TAKE1,
      ngx_http_proxy_pass,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("proxy_redirect"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
      ngx_http_proxy_redirect,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("proxy_cookie_domain"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
      ngx_http_proxy_cookie_domain,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("proxy_cookie_path"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE12,
      ngx_http_proxy_cookie_path,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("proxy_store"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_proxy_store,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("proxy_store_access"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
      ngx_conf_set_access_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.store_access),
      NULL },

    { ngx_string("proxy_buffering"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
```

```
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.buffering),
      NULL },

    { ngx_string("proxy_ignore_client_abort"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.ignore_client_abort),
      NULL },

    { ngx_string("proxy_bind"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_upstream_bind_set_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.local),
      NULL },

    { ngx_string("proxy_connect_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.connect_timeout),
      NULL },

    { ngx_string("proxy_send_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.send_timeout),
      NULL },

    { ngx_string("proxy_send_lowat"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.send_lowat),
      &ngx_http_proxy_lowat_post },

    { ngx_string("proxy_intercept_errors"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.intercept_errors),
      NULL },

    { ngx_string("proxy_set_header"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_conf_set_keyval_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, headers_source),
      NULL },

    { ngx_string("proxy_headers_hash_max_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, headers_hash_max_size),
      NULL },

    { ngx_string("proxy_headers_hash_bucket_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, headers_hash_bucket_size),
      NULL },

    { ngx_string("proxy_set_body"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, body_source),
      NULL },

    { ngx_string("proxy_method"),
```

```
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, method),
      NULL },

    { ngx_string("proxy_pass_request_headers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.pass_request_headers),
      NULL },

    { ngx_string("proxy_pass_request_body"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.pass_request_body),
      NULL },

    { ngx_string("proxy_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.buffer_size),
      NULL },

    { ngx_string("proxy_read_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.read_timeout),
      NULL },

    { ngx_string("proxy_buffers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_conf_set_bufs_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.bufs),
      NULL },

    { ngx_string("proxy_busy_buffers_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.busy_buffers_size_conf),
      NULL },

    { ngx_string("proxy_force_ranges"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.force_ranges),
      NULL },

    { ngx_string("proxy_limit_rate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.limit_rate),
      NULL },

#if (NGX_HTTP_CACHE)

    { ngx_string("proxy_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_proxy_cache,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("proxy_cache_key"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_proxy_cache_key,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
```

```
      NULL },

    { ngx_string("proxy_cache_path"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_2MORE,
      ngx_http_file_cache_set_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_proxy_main_conf_t, caches),
      &ngx_http_proxy_module },

    { ngx_string("proxy_cache_bypass"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_set_predicate_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_bypass),
      NULL },

    { ngx_string("proxy_no_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_set_predicate_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.no_cache),
      NULL },

    { ngx_string("proxy_cache_valid"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_file_cache_valid_set_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_valid),
      NULL },

    { ngx_string("proxy_cache_min_uses"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_min_uses),
      NULL },

    { ngx_string("proxy_cache_use_stale"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_use_stale),
      &ngx_http_proxy_next_upstream_masks },

    { ngx_string("proxy_cache_methods"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_methods),
      &ngx_http_upstream_cache_method_mask },

    { ngx_string("proxy_cache_lock"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_lock),
      NULL },

    { ngx_string("proxy_cache_lock_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_lock_timeout),
      NULL },

    { ngx_string("proxy_cache_lock_age"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_lock_age),
      NULL },

    { ngx_string("proxy_cache_revalidate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
```

```
      offsetof(ngx_http_proxy_loc_conf_t, upstream.cache_revalidate),
      NULL },

#endif

    { ngx_string("proxy_temp_path"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1234,
      ngx_conf_set_path_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.temp_path),
      NULL },

    { ngx_string("proxy_max_temp_file_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.max_temp_file_size_conf),
      NULL },

    { ngx_string("proxy_temp_file_write_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.temp_file_write_size_conf),
      NULL },

    { ngx_string("proxy_next_upstream"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.next_upstream),
      &ngx_http_proxy_next_upstream_masks },

    { ngx_string("proxy_next_upstream_tries"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.next_upstream_tries),
      NULL },

    { ngx_string("proxy_next_upstream_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.next_upstream_timeout),
      NULL },

    { ngx_string("proxy_pass_header"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_array_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.pass_headers),
      NULL },

    { ngx_string("proxy_hide_header"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_array_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.hide_headers),
      NULL },

    { ngx_string("proxy_ignore_headers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, upstream.ignore_headers),
      &ngx_http_upstream_ignore_headers_masks },

    { ngx_string("proxy_http_version"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_enum_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_proxy_loc_conf_t, http_version),
      &ngx_http_proxy_http_version },

#if (NGX_HTTP_SSL)
```

```
594
595     { ngx_string("proxy_ssl_session_reuse"),
596       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
597       ngx_conf_set_flag_slot,
598       NGX_HTTP_LOC_CONF_OFFSET,
599       offsetof(ngx_http_proxy_loc_conf_t, upstream.ssl_session_reuse),
600       NULL },
601
602     { ngx_string("proxy_ssl_protocols"),
603       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
604       ngx_conf_set_bitmask_slot,
605       NGX_HTTP_LOC_CONF_OFFSET,
606       offsetof(ngx_http_proxy_loc_conf_t, ssl_protocols),
607       &ngx_http_proxy_ssl_protocols },
608
609     { ngx_string("proxy_ssl_ciphers"),
610       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
611       ngx_conf_set_str_slot,
612       NGX_HTTP_LOC_CONF_OFFSET,
613       offsetof(ngx_http_proxy_loc_conf_t, ssl_ciphers),
614       NULL },
615
616     { ngx_string("proxy_ssl_name"),
617       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
618       ngx_http_set_complex_value_slot,
619       NGX_HTTP_LOC_CONF_OFFSET,
620       offsetof(ngx_http_proxy_loc_conf_t, upstream.ssl_name),
621       NULL },
622
623     { ngx_string("proxy_ssl_server_name"),
624       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
625       ngx_conf_set_flag_slot,
626       NGX_HTTP_LOC_CONF_OFFSET,
627       offsetof(ngx_http_proxy_loc_conf_t, upstream.ssl_server_name),
628       NULL },
629
630     { ngx_string("proxy_ssl_verify"),
631       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
632       ngx_conf_set_flag_slot,
633       NGX_HTTP_LOC_CONF_OFFSET,
634       offsetof(ngx_http_proxy_loc_conf_t, upstream.ssl_verify),
635       NULL },
636
637     { ngx_string("proxy_ssl_verify_depth"),
638       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
639       ngx_conf_set_num_slot,
640       NGX_HTTP_LOC_CONF_OFFSET,
641       offsetof(ngx_http_proxy_loc_conf_t, ssl_verify_depth),
642       NULL },
643
644     { ngx_string("proxy_ssl_trusted_certificate"),
645       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
646       ngx_conf_set_str_slot,
647       NGX_HTTP_LOC_CONF_OFFSET,
648       offsetof(ngx_http_proxy_loc_conf_t, ssl_trusted_certificate),
649       NULL },
650
651     { ngx_string("proxy_ssl_crl"),
652       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
653       ngx_conf_set_str_slot,
654       NGX_HTTP_LOC_CONF_OFFSET,
655       offsetof(ngx_http_proxy_loc_conf_t, ssl_crl),
656       NULL },
657
658     { ngx_string("proxy_ssl_certificate"),
659       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
660       ngx_conf_set_str_slot,
661       NGX_HTTP_LOC_CONF_OFFSET,
662       offsetof(ngx_http_proxy_loc_conf_t, ssl_certificate),
663       NULL },
664
665     { ngx_string("proxy_ssl_certificate_key"),
666       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
667       ngx_conf_set_str_slot,
668       NGX_HTTP_LOC_CONF_OFFSET,
669       offsetof(ngx_http_proxy_loc_conf_t, ssl_certificate_key),
```

```
      NULL },

    { ngx_string("proxy_ssl_password_file"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_proxy_ssl_password_file,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

#endif

      ngx_null_command
};


static ngx_http_module_t  ngx_http_proxy_module_ctx = {
    ngx_http_proxy_add_variables,          /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_proxy_create_main_conf,       /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_proxy_create_loc_conf,        /* create location configuration */
    ngx_http_proxy_merge_loc_conf          /* merge location configuration */
};


ngx_module_t  ngx_http_proxy_module = {
    NGX_MODULE_V1,
    &ngx_http_proxy_module_ctx,            /* module context */
    ngx_http_proxy_commands,               /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static char  ngx_http_proxy_version[] = " HTTP/1.0" CRLF;
static char  ngx_http_proxy_version_11[] = " HTTP/1.1" CRLF;


static ngx_keyval_t  ngx_http_proxy_headers[] = {
    { ngx_string("Host"), ngx_string("$proxy_host") },
    { ngx_string("Connection"), ngx_string("close") },
    { ngx_string("Content-Length"), ngx_string("$proxy_internal_body_length") },
    { ngx_string("TE"), ngx_string("") },
    { ngx_string("Transfer-Encoding"), ngx_string("") },
    { ngx_string("Keep-Alive"), ngx_string("") },
    { ngx_string("Expect"), ngx_string("") },
    { ngx_string("Upgrade"), ngx_string("") },
    { ngx_null_string, ngx_null_string }
};


static ngx_str_t  ngx_http_proxy_hide_headers[] = {
    ngx_string("Date"),
    ngx_string("Server"),
    ngx_string("X-Pad"),
    ngx_string("X-Accel-Expires"),
    ngx_string("X-Accel-Redirect"),
    ngx_string("X-Accel-Limit-Rate"),
    ngx_string("X-Accel-Buffering"),
    ngx_string("X-Accel-Charset"),
    ngx_null_string
};
```

```
#if (NGX_HTTP_CACHE)

static ngx_keyval_t  ngx_http_proxy_cache_headers[] = {
    { ngx_string("Host"), ngx_string("$proxy_host") },
    { ngx_string("Connection"), ngx_string("close") },
    { ngx_string("Content-Length"), ngx_string("$proxy_internal_body_length") },
    { ngx_string("TE"), ngx_string("") },
    { ngx_string("Transfer-Encoding"), ngx_string("") },
    { ngx_string("Keep-Alive"), ngx_string("") },
    { ngx_string("Expect"), ngx_string("") },
    { ngx_string("Upgrade"), ngx_string("") },
    { ngx_string("If-Modified-Since"),
      ngx_string("$upstream_cache_last_modified") },
    { ngx_string("If-Unmodified-Since"), ngx_string("") },
    { ngx_string("If-None-Match"), ngx_string("$upstream_cache_etag") },
    { ngx_string("If-Match"), ngx_string("") },
    { ngx_string("Range"), ngx_string("") },
    { ngx_string("If-Range"), ngx_string("") },
    { ngx_null_string, ngx_null_string }
};

#endif


static ngx_http_variable_t  ngx_http_proxy_vars[] = {

    { ngx_string("proxy_host"), NULL, ngx_http_proxy_host_variable, 0,
      NGX_HTTP_VAR_CHANGEABLE|NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

    { ngx_string("proxy_port"), NULL, ngx_http_proxy_port_variable, 0,
      NGX_HTTP_VAR_CHANGEABLE|NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

    { ngx_string("proxy_add_x_forwarded_for"), NULL,
      ngx_http_proxy_add_x_forwarded_for_variable, 0, NGX_HTTP_VAR_NOHASH, 0 },

#if 0
    { ngx_string("proxy_add_via"), NULL, NULL, 0, NGX_HTTP_VAR_NOHASH, 0 },
#endif

    { ngx_string("proxy_internal_body_length"), NULL,
      ngx_http_proxy_internal_body_length_variable, 0,
      NGX_HTTP_VAR_NOCACHEABLE|NGX_HTTP_VAR_NOHASH, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static ngx_path_init_t  ngx_http_proxy_temp_path = {
    ngx_string(NGX_HTTP_PROXY_TEMP_PATH), { 1, 2, 0 }
};


static ngx_int_t
ngx_http_proxy_handler(ngx_http_request_t *r)
{
    ngx_int_t                   rc;
    ngx_http_upstream_t        *u;
    ngx_http_proxy_ctx_t       *ctx;
    ngx_http_proxy_loc_conf_t  *plcf;
#if (NGX_HTTP_CACHE)
    ngx_http_proxy_main_conf_t  *pmcf;
#endif

    if (ngx_http_upstream_create(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_proxy_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_proxy_module);

    plcf = ngx_http_get_module_loc_conf(r, ngx_http_proxy_module);

```

```
    u = r->upstream;

    if (plcf->proxy_lengths == NULL) {
        ctx->vars = plcf->vars;
        u->schema = plcf->vars.schema;
#if (NGX_HTTP_SSL)
        u->ssl = (plcf->upstream.ssl != NULL);
#endif

    } else {
        if (ngx_http_proxy_eval(r, ctx, plcf) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }
    }

    u->output.tag = (ngx_buf_tag_t) &ngx_http_proxy_module;

    u->conf = &plcf->upstream;

#if (NGX_HTTP_CACHE)
    pmcf = ngx_http_get_module_main_conf(r, ngx_http_proxy_module);

    u->caches = &pmcf->caches;
    u->create_key = ngx_http_proxy_create_key;
#endif

    u->create_request = ngx_http_proxy_create_request;
    u->reinit_request = ngx_http_proxy_reinit_request;
    u->process_header = ngx_http_proxy_process_status_line;
    u->abort_request = ngx_http_proxy_abort_request;
    u->finalize_request = ngx_http_proxy_finalize_request;
    r->state = 0;

    if (plcf->redirects) {
        u->rewrite_redirect = ngx_http_proxy_rewrite_redirect;
    }

    if (plcf->cookie_domains || plcf->cookie_paths) {
        u->rewrite_cookie = ngx_http_proxy_rewrite_cookie;
    }

    u->buffering = plcf->upstream.buffering;

    u->pipe = ngx_pcalloc(r->pool, sizeof(ngx_event_pipe_t));
    if (u->pipe == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    u->pipe->input_filter = ngx_http_proxy_copy_filter;
    u->pipe->input_ctx = r;

    u->input_filter_init = ngx_http_proxy_input_filter_init;
    u->input_filter = ngx_http_proxy_non_buffered_copy_filter;
    u->input_filter_ctx = r;

    u->accel = 1;

    rc = ngx_http_read_client_request_body(r, ngx_http_upstream_init);

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        return rc;
    }

    return NGX_DONE;
}


static ngx_int_t
ngx_http_proxy_eval(ngx_http_request_t *r, ngx_http_proxy_ctx_t *ctx,
    ngx_http_proxy_loc_conf_t *plcf)
{
    u_char               *p;
    size_t                add;
    u_short               port;
    ngx_str_t             proxy;
    ngx_url_t             url;
```

```
    ngx_http_upstream_t  *u;

    if (ngx_http_script_run(r, &proxy, plcf->proxy_lengths->elts, 0,
                            plcf->proxy_values->elts)
        == NULL)
    {
        return NGX_ERROR;
    }

    if (proxy.len > 7
        && ngx_strncasecmp(proxy.data, (u_char *) "http://", 7) == 0)
    {
        add = 7;
        port = 80;

#if (NGX_HTTP_SSL)

    } else if (proxy.len > 8
               && ngx_strncasecmp(proxy.data, (u_char *) "https://", 8) == 0)
    {
        add = 8;
        port = 443;
        r->upstream->ssl = 1;

#endif

    } else {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "invalid URL prefix in \"%V\"", &proxy);
        return NGX_ERROR;
    }

    u = r->upstream;

    u->schema.len = add;
    u->schema.data = proxy.data;

    ngx_memzero(&url, sizeof(ngx_url_t));

    url.url.len = proxy.len - add;
    url.url.data = proxy.data + add;
    url.default_port = port;
    url.uri_part = 1;
    url.no_resolve = 1;

    if (ngx_parse_url(r->pool, &url) != NGX_OK) {
        if (url.err) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "%s in upstream \"%V\"", url.err, &url.url);
        }

        return NGX_ERROR;
    }

    if (url.uri.len) {
        if (url.uri.data[0] == '?') {
            p = ngx_pnalloc(r->pool, url.uri.len + 1);
            if (p == NULL) {
                return NGX_ERROR;
            }

            *p++ = '/';
            ngx_memcpy(p, url.uri.data, url.uri.len);

            url.uri.len++;
            url.uri.data = p - 1;
        }
    }

    ctx->vars.key_start = u->schema;

    ngx_http_proxy_set_vars(&url, &ctx->vars);

    u->resolved = ngx_pcalloc(r->pool, sizeof(ngx_http_upstream_resolved_t));
    if (u->resolved == NULL) {
        return NGX_ERROR;
```

```
    }

    if (url.addrs && url.addrs[0].sockaddr) {
        u->resolved->sockaddr = url.addrs[0].sockaddr;
        u->resolved->socklen = url.addrs[0].socklen;
        u->resolved->naddrs = 1;
        u->resolved->host = url.addrs[0].name;

    } else {
        u->resolved->host = url.host;
        u->resolved->port = (in_port_t) (url.no_port ? port : url.port);
        u->resolved->no_port = url.no_port;
    }

    return NGX_OK;
}


#if (NGX_HTTP_CACHE)

static ngx_int_t
ngx_http_proxy_create_key(ngx_http_request_t *r)
{
    size_t                      len, loc_len;
    u_char                     *p;
    uintptr_t                   escape;
     ngx_str_t                  *key;
     ngx_http_upstream_t        *u;
     ngx_http_proxy_ctx_t       *ctx;
     ngx_http_proxy_loc_conf_t  *plcf;

     u = r->upstream;

     plcf = ngx_http_get_module_loc_conf(r, ngx_http_proxy_module);

     ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);

     key = ngx_array_push(&r->cache->keys);
     if (key == NULL) {
         return NGX_ERROR;
     }

     if (plcf->cache_key.value.data) {

         if (ngx_http_complex_value(r, &plcf->cache_key, key) != NGX_OK) {
             return NGX_ERROR;
         }

         return NGX_OK;
     }

     *key = ctx->vars.key_start;

     key = ngx_array_push(&r->cache->keys);
     if (key == NULL) {
         return NGX_ERROR;
     }

     if (plcf->proxy_lengths && ctx->vars.uri.len) {

         *key = ctx->vars.uri;
         u->uri = ctx->vars.uri;

         return NGX_OK;

     } else if (ctx->vars.uri.len == 0 && r->valid_unparsed_uri && r == r->main)
     {
         *key = r->unparsed_uri;
         u->uri = r->unparsed_uri;

         return NGX_OK;
     }

     loc_len = (r->valid_location && ctx->vars.uri.len) ? plcf->location.len : 0;

     if (r->quoted_uri || r->internal) {
```

```
        escape = 2 * ngx_escape_uri(NULL, r->uri.data + loc_len,
                                    r->uri.len - loc_len, NGX_ESCAPE_URI);
    } else {
        escape = 0;
    }

    len = ctx->vars.uri.len + r->uri.len - loc_len + escape
          + sizeof("?") - 1 + r->args.len;

    p = ngx_pnalloc(r->pool, len);
    if (p == NULL) {
        return NGX_ERROR;
    }

    key->data = p;

    if (r->valid_location) {
        p = ngx_copy(p, ctx->vars.uri.data, ctx->vars.uri.len);
    }

    if (escape) {
        ngx_escape_uri(p, r->uri.data + loc_len,
                       r->uri.len - loc_len, NGX_ESCAPE_URI);
        p += r->uri.len - loc_len + escape;

    } else {
        p = ngx_copy(p, r->uri.data + loc_len, r->uri.len - loc_len);
    }

    if (r->args.len > 0) {
        *p++ = '?';
        p = ngx_copy(p, r->args.data, r->args.len);
    }

    key->len = p - key->data;
    u->uri = *key;

    return NGX_OK;
}

#endif


static ngx_int_t
ngx_http_proxy_create_request(ngx_http_request_t *r)
{
    size_t                        len, uri_len, loc_len, body_len;
    uintptr_t                     escape;
    ngx_buf_t                    *b;
    ngx_str_t                     method;
    ngx_uint_t                    i, unparsed_uri;
    ngx_chain_t                  *cl, *body;
    ngx_list_part_t              *part;
    ngx_table_elt_t              *header;
    ngx_http_upstream_t          *u;
    ngx_http_proxy_ctx_t         *ctx;
    ngx_http_script_code_pt       code;
    ngx_http_proxy_headers_t     *headers;
    ngx_http_script_engine_t      e, le;
    ngx_http_proxy_loc_conf_t    *plcf;
    ngx_http_script_len_code_pt   lcode;

    u = r->upstream;

    plcf = ngx_http_get_module_loc_conf(r, ngx_http_proxy_module);

#if (NGX_HTTP_CACHE)
    headers = u->cacheable ? &plcf->headers_cache : &plcf->headers;
#else
    headers = &plcf->headers;
#endif

    if (u->method.len) {
        /* HEAD was changed to GET to cache response */
        method = u->method;
        method.len++;
```

```
1126
1127        } else if (plcf->method.len) {
1128            method = plcf->method;
1129
1130        } else {
1131            method = r->method_name;
1132            method.len++;
1133        }
1134
1135        ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);
1136
1137        if (method.len == 5
1138            && ngx_strncasecmp(method.data, (u_char *) "HEAD ", 5) == 0)
1139        {
1140            ctx->head = 1;
1141        }
1142
1143        len = method.len + sizeof(ngx_http_proxy_version) - 1 + sizeof(CRLF) - 1;
1144
1145        escape = 0;
1146        loc_len = 0;
1147        unparsed_uri = 0;
1148
1149        if (plcf->proxy_lengths && ctx->vars.uri.len) {
1150            uri_len = ctx->vars.uri.len;
1151
1152        } else if (ctx->vars.uri.len == 0 && r->valid_unparsed_uri && r == r->main)
1153        {
1154            unparsed_uri = 1;
1155            uri_len = r->unparsed_uri.len;
1156
1157        } else {
1158            loc_len = (r->valid_location && ctx->vars.uri.len) ?
1159                          plcf->location.len : 0;
1160
1161            if (r->quoted_uri || r->space_in_uri || r->internal) {
1162                escape = 2 * ngx_escape_uri(NULL, r->uri.data + loc_len,
1163                                            r->uri.len - loc_len, NGX_ESCAPE_URI);
1164            }
1165
1166            uri_len = ctx->vars.uri.len + r->uri.len - loc_len + escape
1167                      + sizeof("?") - 1 + r->args.len;
1168        }
1169
1170        if (uri_len == 0) {
1171            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1172                          "zero length URI to proxy");
1173            return NGX_ERROR;
1174        }
1175
1176        len += uri_len;
1177
1178        ngx_memzero(&le, sizeof(ngx_http_script_engine_t));
1179
1180        ngx_http_script_flush_no_cacheable_variables(r, plcf->body_flushes);
1181        ngx_http_script_flush_no_cacheable_variables(r, headers->flushes);
1182
1183        if (plcf->body_lengths) {
1184            le.ip = plcf->body_lengths->elts;
1185            le.request = r;
1186            le.flushed = 1;
1187            body_len = 0;
1188
1189            while (*(uintptr_t *) le.ip) {
1190                lcode = *(ngx_http_script_len_code_pt *) le.ip;
1191                body_len += lcode(&le);
1192            }
1193
1194            ctx->internal_body_length = body_len;
1195            len += body_len;
1196
1197        } else {
1198            ctx->internal_body_length = r->headers_in.content_length_n;
1199        }
1200
1201        le.ip = headers->lengths->elts;
```

```
    le.request = r;
    le.flushed = 1;

    while (*(uintptr_t *) le.ip) {
        while (*(uintptr_t *) le.ip) {
            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            len += lcode(&le);
        }
        le.ip += sizeof(uintptr_t);
    }


    if (plcf->upstream.pass_request_headers) {
        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            if (ngx_hash_find(&headers->hash, header[i].hash,
                              header[i].lowcase_key, header[i].key.len))
            {
                continue;
            }

            len += header[i].key.len + sizeof(": ") - 1
                + header[i].value.len + sizeof(CRLF) - 1;
        }
    }


    b = ngx_create_temp_buf(r->pool, len);
    if (b == NULL) {
        return NGX_ERROR;
    }

    cl = ngx_alloc_chain_link(r->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    cl->buf = b;


    /* the request line */

    b->last = ngx_copy(b->last, method.data, method.len);

    u->uri.data = b->last;

    if (plcf->proxy_lengths && ctx->vars.uri.len) {
        b->last = ngx_copy(b->last, ctx->vars.uri.data, ctx->vars.uri.len);

    } else if (unparsed_uri) {
        b->last = ngx_copy(b->last, r->unparsed_uri.data, r->unparsed_uri.len);

    } else {
        if (r->valid_location) {
            b->last = ngx_copy(b->last, ctx->vars.uri.data, ctx->vars.uri.len);
        }

        if (escape) {
            ngx_escape_uri(b->last, r->uri.data + loc_len,
                           r->uri.len - loc_len, NGX_ESCAPE_URI);
            b->last += r->uri.len - loc_len + escape;

        } else {
```

```
            b->last = ngx_copy(b->last, r->uri.data + loc_len,
                               r->uri.len - loc_len);
        }

        if (r->args.len > 0) {
            *b->last++ = '?';
            b->last = ngx_copy(b->last, r->args.data, r->args.len);
        }
    }

    u->uri.len = b->last - u->uri.data;

    if (plcf->http_version == NGX_HTTP_VERSION_11) {
        b->last = ngx_cpymem(b->last, ngx_http_proxy_version_11,
                             sizeof(ngx_http_proxy_version_11) - 1);

    } else {
        b->last = ngx_cpymem(b->last, ngx_http_proxy_version,
                             sizeof(ngx_http_proxy_version) - 1);
    }

    ngx_memzero(&e, sizeof(ngx_http_script_engine_t));

    e.ip = headers->values->elts;
    e.pos = b->last;
    e.request = r;
    e.flushed = 1;

    le.ip = headers->lengths->elts;

    while (*(uintptr_t *) le.ip) {
        lcode = *(ngx_http_script_len_code_pt *) le.ip;

        /* skip the header line name length */
        (void) lcode(&le);

        if (*(ngx_http_script_len_code_pt *) le.ip) {

            for (len = 0; *(uintptr_t *) le.ip; len += lcode(&le)) {
                lcode = *(ngx_http_script_len_code_pt *) le.ip;
            }

            e.skip = (len == sizeof(CRLF) - 1) ? 1 : 0;

        } else {
            e.skip = 0;
        }

        le.ip += sizeof(uintptr_t);

        while (*(uintptr_t *) e.ip) {
            code = *(ngx_http_script_code_pt *) e.ip;
            code((ngx_http_script_engine_t *) &e);
        }
        e.ip += sizeof(uintptr_t);
    }

    b->last = e.pos;


    if (plcf->upstream.pass_request_headers) {
        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

```

```
            if (ngx_hash_find(&headers->hash, header[i].hash,
                              header[i].lowcase_key, header[i].key.len))
            {
                continue;
            }

            b->last = ngx_copy(b->last, header[i].key.data, header[i].key.len);

            *b->last++ = ':'; *b->last++ = ' ';

            b->last = ngx_copy(b->last, header[i].value.data,
                               header[i].value.len);

            *b->last++ = CR; *b->last++ = LF;

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http proxy header: \"%V: %V\"",
                           &header[i].key, &header[i].value);
        }
    }


    /* add "\r\n" at the header end */
    *b->last++ = CR; *b->last++ = LF;

    if (plcf->body_values) {
        e.ip = plcf->body_values->elts;
        e.pos = b->last;

        while (*(uintptr_t *) e.ip) {
            code = *(ngx_http_script_code_pt *) e.ip;
            code((ngx_http_script_engine_t *) &e);
        }

        b->last = e.pos;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http proxy header:%N\"%*s\"",
                   (size_t) (b->last - b->pos), b->pos);

    if (plcf->body_values == NULL && plcf->upstream.pass_request_body) {

        body = u->request_bufs;
        u->request_bufs = cl;

        while (body) {
            b = ngx_alloc_buf(r->pool);
            if (b == NULL) {
                return NGX_ERROR;
            }

            ngx_memcpy(b, body->buf, sizeof(ngx_buf_t));

            cl->next = ngx_alloc_chain_link(r->pool);
            if (cl->next == NULL) {
                return NGX_ERROR;
            }

            cl = cl->next;
            cl->buf = b;

            body = body->next;
        }

    } else {
        u->request_bufs = cl;
    }

    b->flush = 1;
    cl->next = NULL;

    return NGX_OK;
}
```

```
1430 static ngx_int_t
1431 ngx_http_proxy_reinit_request(ngx_http_request_t *r)
1432 {
1433     ngx_http_proxy_ctx_t  *ctx;
1434
1435     ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);
1436
1437     if (ctx == NULL) {
1438         return NGX_OK;
1439     }
1440
1441     ctx->status.code = 0;
1442     ctx->status.count = 0;
1443     ctx->status.start = NULL;
1444     ctx->status.end = NULL;
1445     ctx->chunked.state = 0;
1446
1447     r->upstream->process_header = ngx_http_proxy_process_status_line;
1448     r->upstream->pipe->input_filter = ngx_http_proxy_copy_filter;
1449     r->upstream->input_filter = ngx_http_proxy_non_buffered_copy_filter;
1450     r->state = 0;
1451
1452     return NGX_OK;
1453 }
1454
1455
1456 static ngx_int_t
1457 ngx_http_proxy_process_status_line(ngx_http_request_t *r)
1458 {
1459     size_t                 len;
1460     ngx_int_t              rc;
1461     ngx_http_upstream_t   *u;
1462     ngx_http_proxy_ctx_t  *ctx;
1463
1464     ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);
1465
1466     if (ctx == NULL) {
1467         return NGX_ERROR;
1468     }
1469
1470     u = r->upstream;
1471
1472     rc = ngx_http_parse_status_line(r, &u->buffer, &ctx->status);
1473
1474     if (rc == NGX_AGAIN) {
1475         return rc;
1476     }
1477
1478     if (rc == NGX_ERROR) {
1479
1480 #if (NGX_HTTP_CACHE)
1481
1482         if (r->cache) {
1483             r->http_version = NGX_HTTP_VERSION_9;
1484             return NGX_OK;
1485         }
1486
1487 #endif
1488
1489         ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1490                       "upstream sent no valid HTTP/1.0 header");
1491
1492 #if 0
1493         if (u->accel) {
1494             return NGX_HTTP_UPSTREAM_INVALID_HEADER;
1495         }
1496 #endif
1497
1498         r->http_version = NGX_HTTP_VERSION_9;
1499         u->state->status = NGX_HTTP_OK;
1500         u->headers_in.connection_close = 1;
1501
1502         return NGX_OK;
1503     }
1504
1505     if (u->state && u->state->status == 0) {
```

```
1506        u->state->status = ctx->status.code;
1507    }
1508
1509    u->headers_in.status_n = ctx->status.code;
1510
1511    len = ctx->status.end - ctx->status.start;
1512    u->headers_in.status_line.len = len;
1513
1514    u->headers_in.status_line.data = ngx_pnalloc(r->pool, len);
1515    if (u->headers_in.status_line.data == NULL) {
1516        return NGX_ERROR;
1517    }
1518
1519    ngx_memcpy(u->headers_in.status_line.data, ctx->status.start, len);
1520
1521    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1522                   "http proxy status %ui \"%V\"",
1523                   u->headers_in.status_n, &u->headers_in.status_line);
1524
1525    if (ctx->status.http_version < NGX_HTTP_VERSION_11) {
1526        u->headers_in.connection_close = 1;
1527    }
1528
1529    u->process_header = ngx_http_proxy_process_header;
1530
1531    return ngx_http_proxy_process_header(r);
1532}
1533
1534
1535static ngx_int_t
1536ngx_http_proxy_process_header(ngx_http_request_t *r)
1537{
1538    ngx_int_t                      rc;
1539    ngx_table_elt_t               *h;
1540    ngx_http_upstream_t           *u;
1541    ngx_http_proxy_ctx_t          *ctx;
1542    ngx_http_upstream_header_t    *hh;
1543    ngx_http_upstream_main_conf_t  *umcf;
1544
1545    umcf = ngx_http_get_module_main_conf(r, ngx_http_upstream_module);
1546
1547    for ( ;; ) {
1548
1549        rc = ngx_http_parse_header_line(r, &r->upstream->buffer, 1);
1550
1551        if (rc == NGX_OK) {
1552
1553            /* a header line has been parsed successfully */
1554
1555            h = ngx_list_push(&r->upstream->headers_in.headers);
1556            if (h == NULL) {
1557                return NGX_ERROR;
1558            }
1559
1560            h->hash = r->header_hash;
1561
1562            h->key.len = r->header_name_end - r->header_name_start;
1563            h->value.len = r->header_end - r->header_start;
1564
1565            h->key.data = ngx_pnalloc(r->pool,
1566                                      h->key.len + 1 + h->value.len + 1 + h->key.len);
1567            if (h->key.data == NULL) {
1568                return NGX_ERROR;
1569            }
1570
1571            h->value.data = h->key.data + h->key.len + 1;
1572            h->lowcase_key = h->key.data + h->key.len + 1 + h->value.len + 1;
1573
1574            ngx_memcpy(h->key.data, r->header_name_start, h->key.len);
1575            h->key.data[h->key.len] = '\0';
1576            ngx_memcpy(h->value.data, r->header_start, h->value.len);
1577            h->value.data[h->value.len] = '\0';
1578
1579            if (h->key.len == r->lowcase_index) {
1580                ngx_memcpy(h->lowcase_key, r->lowcase_header, h->key.len);
1581
```

```
            } else {
                ngx_strlow(h->lowcase_key, h->key.data, h->key.len);
            }

            hh = ngx_hash_find(&umcf->headers_in_hash, h->hash,
                               h->lowcase_key, h->key.len);

            if (hh && hh->handler(r, h, hh->offset) != NGX_OK) {
                return NGX_ERROR;
            }

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http proxy header: \"%V: %V\"",
                           &h->key, &h->value);

            continue;
        }

        if (rc == NGX_HTTP_PARSE_HEADER_DONE) {

            /* a whole header has been parsed successfully */

            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http proxy header done");

            /*
             * if no "Server" and "Date" in header line,
             * then add the special empty headers
             */

            if (r->upstream->headers_in.server == NULL) {
                h = ngx_list_push(&r->upstream->headers_in.headers);
                if (h == NULL) {
                    return NGX_ERROR;
                }

                h->hash = ngx_hash(ngx_hash(ngx_hash(ngx_hash(
                                  ngx_hash('s', 'e'), 'r'), 'v'), 'e'), 'r');

                ngx_str_set(&h->key, "Server");
                ngx_str_null(&h->value);
                h->lowcase_key = (u_char *) "server";
            }

            if (r->upstream->headers_in.date == NULL) {
                h = ngx_list_push(&r->upstream->headers_in.headers);
                if (h == NULL) {
                    return NGX_ERROR;
                }

                h->hash = ngx_hash(ngx_hash(ngx_hash('d', 'a'), 't'), 'e');

                ngx_str_set(&h->key, "Date");
                ngx_str_null(&h->value);
                h->lowcase_key = (u_char *) "date";
            }

            /* clear content length if response is chunked */

            u = r->upstream;

            if (u->headers_in.chunked) {
                u->headers_in.content_length_n = -1;
            }

            /*
             * set u->keepalive if response has no body; this allows to keep
             * connections alive in case of r->header_only or X-Accel-Redirect
             */

            ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);

            if (u->headers_in.status_n == NGX_HTTP_NO_CONTENT
                || u->headers_in.status_n == NGX_HTTP_NOT_MODIFIED
                || ctx->head
                || (!u->headers_in.chunked
```

```
                    && u->headers_in.content_length_n == 0))
            {
                u->keepalive = !u->headers_in.connection_close;
            }

            if (u->headers_in.status_n == NGX_HTTP_SWITCHING_PROTOCOLS) {
                u->keepalive = 0;

                if (r->headers_in.upgrade) {
                    u->upgrade = 1;
                }
            }

            return NGX_OK;
        }

        if (rc == NGX_AGAIN) {
            return NGX_AGAIN;
        }

        /* there was error while a header line parsing */

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "upstream sent invalid header");

        return NGX_HTTP_UPSTREAM_INVALID_HEADER;
    }
}


static ngx_int_t
ngx_http_proxy_input_filter_init(void *data)
{
    ngx_http_request_t    *r = data;
    ngx_http_upstream_t   *u;
    ngx_http_proxy_ctx_t  *ctx;

    u = r->upstream;
    ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);

    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http proxy filter init s:%d h:%d c:%d l:%O",
                   u->headers_in.status_n, ctx->head, u->headers_in.chunked,
                   u->headers_in.content_length_n);

    /* as per RFC2616, 4.4 Message Length */

    if (u->headers_in.status_n == NGX_HTTP_NO_CONTENT
        || u->headers_in.status_n == NGX_HTTP_NOT_MODIFIED
        || ctx->head)
    {
        /* 1xx, 204, and 304 and replies to HEAD requests */
        /* no 1xx since we don't send Expect and Upgrade */

        u->pipe->length = 0;
        u->length = 0;
        u->keepalive = !u->headers_in.connection_close;

    } else if (u->headers_in.chunked) {
        /* chunked */

        u->pipe->input_filter = ngx_http_proxy_chunked_filter;
        u->pipe->length = 3; /* "0" LF LF */

        u->input_filter = ngx_http_proxy_non_buffered_chunked_filter;
        u->length = 1;

    } else if (u->headers_in.content_length_n == 0) {
        /* empty body: special case as filter won't be called */

        u->pipe->length = 0;
        u->length = 0;
```

```
1734         u->keepalive = !u->headers_in.connection_close;
1735
1736     } else {
1737         /* content length or connection close */
1738
1739         u->pipe->length = u->headers_in.content_length_n;
1740         u->length = u->headers_in.content_length_n;
1741     }
1742
1743     return NGX_OK;
1744 }
1745
1746
1747 static ngx_int_t
1748 ngx_http_proxy_copy_filter(ngx_event_pipe_t *p, ngx_buf_t *buf)
1749 {
1750     ngx_buf_t           *b;
1751     ngx_chain_t         *cl;
1752     ngx_http_request_t  *r;
1753
1754     if (buf->pos == buf->last) {
1755         return NGX_OK;
1756     }
1757
1758     cl = ngx_chain_get_free_buf(p->pool, &p->free);
1759     if (cl == NULL) {
1760         return NGX_ERROR;
1761     }
1762
1763     b = cl->buf;
1764
1765     ngx_memcpy(b, buf, sizeof(ngx_buf_t));
1766     b->shadow = buf;
1767     b->tag = p->tag;
1768     b->last_shadow = 1;
1769     b->recycled = 1;
1770     buf->shadow = b;
1771
1772     ngx_log_debug1(NGX_LOG_DEBUG_EVENT, p->log, 0, "input buf #%d", b->num);
1773
1774     if (p->in) {
1775         *p->last_in = cl;
1776     } else {
1777         p->in = cl;
1778     }
1779     p->last_in = &cl->next;
1780
1781     if (p->length == -1) {
1782         return NGX_OK;
1783     }
1784
1785     p->length -= b->last - b->pos;
1786
1787     if (p->length == 0) {
1788         r = p->input_ctx;
1789         p->upstream_done = 1;
1790         r->upstream->keepalive = !r->upstream->headers_in.connection_close;
1791
1792     } else if (p->length < 0) {
1793         r = p->input_ctx;
1794         p->upstream_done = 1;
1795
1796         ngx_log_error(NGX_LOG_WARN, r->connection->log, 0,
1797                       "upstream sent more data than specified in "
1798                       "\"Content-Length\" header");
1799     }
1800
1801     return NGX_OK;
1802 }
1803
1804
1805 static ngx_int_t
1806 ngx_http_proxy_chunked_filter(ngx_event_pipe_t *p, ngx_buf_t *buf)
1807 {
1808     ngx_int_t              rc;
1809     ngx_buf_t             *b, **prev;
```

```
    ngx_chain_t           *cl;
    ngx_http_request_t    *r;
    ngx_http_proxy_ctx_t  *ctx;

    if (buf->pos == buf->last) {
        return NGX_OK;
    }

    r = p->input_ctx;
    ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);

    if (ctx == NULL) {
        return NGX_ERROR;
    }

    b = NULL;
    prev = &buf->shadow;

    for ( ;; ) {

        rc = ngx_http_parse_chunked(r, buf, &ctx->chunked);

        if (rc == NGX_OK) {

            /* a chunk has been parsed successfully */

            cl = ngx_chain_get_free_buf(p->pool, &p->free);
            if (cl == NULL) {
                return NGX_ERROR;
            }

            b = cl->buf;

            ngx_memzero(b, sizeof(ngx_buf_t));

            b->pos = buf->pos;
            b->start = buf->start;
            b->end = buf->end;
            b->tag = p->tag;
            b->temporary = 1;
            b->recycled = 1;

            *prev = b;
            prev = &b->shadow;

            if (p->in) {
                *p->last_in = cl;
            } else {
                p->in = cl;
            }
            p->last_in = &cl->next;

            /* STUB */ b->num = buf->num;

            ngx_log_debug2(NGX_LOG_DEBUG_EVENT, p->log, 0,
                           "input buf #%d %p", b->num, b->pos);

            if (buf->last - buf->pos >= ctx->chunked.size) {

                buf->pos += (size_t) ctx->chunked.size;
                b->last = buf->pos;
                ctx->chunked.size = 0;

                continue;
            }

            ctx->chunked.size -= buf->last - buf->pos;
            buf->pos = buf->last;
            b->last = buf->last;

            continue;
        }

        if (rc == NGX_DONE) {

            /* a whole response has been parsed successfully */
```

```
            p->upstream_done = 1;
            r->upstream->keepalive = !r->upstream->headers_in.connection_close;

            break;
        }

        if (rc == NGX_AGAIN) {

            /* set p->length, minimal amount of data we want to see */

            p->length = ctx->chunked.length;

            break;
        }

        /* invalid response */

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "upstream sent invalid chunked response");

        return NGX_ERROR;
    }

    ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http proxy chunked state %d, length %d",
                   ctx->chunked.state, p->length);

    if (b) {
        b->shadow = buf;
        b->last_shadow = 1;

        ngx_log_debug2(NGX_LOG_DEBUG_EVENT, p->log, 0,
                       "input buf %p %z", b->pos, b->last - b->pos);

        return NGX_OK;
    }

    /* there is no data record in the buf, add it to free chain */

    if (ngx_event_pipe_add_free_buf(p, buf) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_proxy_non_buffered_copy_filter(void *data, ssize_t bytes)
{
    ngx_http_request_t   *r = data;

    ngx_buf_t            *b;
    ngx_chain_t          *cl, **ll;
    ngx_http_upstream_t  *u;

    u = r->upstream;

    for (cl = u->out_bufs, ll = &u->out_bufs; cl; cl = cl->next) {
        ll = &cl->next;
    }

    cl = ngx_chain_get_free_buf(r->pool, &u->free_bufs);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    *ll = cl;

    cl->buf->flush = 1;
    cl->buf->memory = 1;

    b = &u->buffer;

    cl->buf->pos = b->last;
```

```
    b->last += bytes;
    cl->buf->last = b->last;
    cl->buf->tag = u->output.tag;

    if (u->length == -1) {
        return NGX_OK;
    }

    u->length -= bytes;

    if (u->length == 0) {
        u->keepalive = !u->headers_in.connection_close;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_proxy_non_buffered_chunked_filter(void *data, ssize_t bytes)
{
    ngx_http_request_t   *r = data;

    ngx_int_t             rc;
    ngx_buf_t            *b, *buf;
    ngx_chain_t          *cl, **ll;
    ngx_http_upstream_t  *u;
    ngx_http_proxy_ctx_t  *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);

    if (ctx == NULL) {
        return NGX_ERROR;
    }

    u = r->upstream;
    buf = &u->buffer;

    buf->pos = buf->last;
    buf->last += bytes;

    for (cl = u->out_bufs, ll = &u->out_bufs; cl; cl = cl->next) {
        ll = &cl->next;
    }

    for ( ;; ) {

        rc = ngx_http_parse_chunked(r, buf, &ctx->chunked);

        if (rc == NGX_OK) {

            /* a chunk has been parsed successfully */

            cl = ngx_chain_get_free_buf(r->pool, &u->free_bufs);
            if (cl == NULL) {
                return NGX_ERROR;
            }

            *ll = cl;
            ll = &cl->next;

            b = cl->buf;

            b->flush = 1;
            b->memory = 1;

            b->pos = buf->pos;
            b->tag = u->output.tag;

            if (buf->last - buf->pos >= ctx->chunked.size) {
                buf->pos += (size_t) ctx->chunked.size;
                b->last = buf->pos;
                ctx->chunked.size = 0;

            } else {
                ctx->chunked.size -= buf->last - buf->pos;
```

```
2038                buf->pos = buf->last;
2039                b->last = buf->last;
2040            }
2041
2042            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2043                           "http proxy out buf %p %z",
2044                           b->pos, b->last - b->pos);
2045
2046            continue;
2047        }
2048
2049        if (rc == NGX_DONE) {
2050
2051            /* a whole response has been parsed successfully */
2052
2053            u->keepalive = !u->headers_in.connection_close;
2054            u->length = 0;
2055
2056            break;
2057        }
2058
2059        if (rc == NGX_AGAIN) {
2060            break;
2061        }
2062
2063        /* invalid response */
2064
2065        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
2066                      "upstream sent invalid chunked response");
2067
2068        return NGX_ERROR;
2069    }
2070
2071    /* provide continuous buffer for subrequests in memory */
2072
2073    if (r->subrequest_in_memory) {
2074
2075        cl = u->out_bufs;
2076
2077        if (cl) {
2078            buf->pos = cl->buf->pos;
2079        }
2080
2081        buf->last = buf->pos;
2082
2083        for (cl = u->out_bufs; cl; cl = cl->next) {
2084            ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2085                           "http proxy in memory %p-%p %uz",
2086                           cl->buf->pos, cl->buf->last, ngx_buf_size(cl->buf));
2087
2088            if (buf->last == cl->buf->pos) {
2089                buf->last = cl->buf->last;
2090                continue;
2091            }
2092
2093            buf->last = ngx_movemem(buf->last, cl->buf->pos,
2094                                    cl->buf->last - cl->buf->pos);
2095
2096            cl->buf->pos = buf->last - (cl->buf->last - cl->buf->pos);
2097            cl->buf->last = buf->last;
2098        }
2099    }
2100
2101    return NGX_OK;
2102 }
2103
2104
2105 static void
2106 ngx_http_proxy_abort_request(ngx_http_request_t *r)
2107 {
2108     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2109                    "abort http proxy request");
2110
2111     return;
2112 }
2113
```

```
2114
2115 static void
2116 ngx_http_proxy_finalize_request(ngx_http_request_t *r, ngx_int_t rc)
2117 {
2118     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
2119                    "finalize http proxy request");
2120
2121     return;
2122 }
2123
2124
2125 static ngx_int_t
2126 ngx_http_proxy_host_variable(ngx_http_request_t *r,
2127     ngx_http_variable_value_t *v, uintptr_t data)
2128 {
2129     ngx_http_proxy_ctx_t  *ctx;
2130
2131     ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);
2132
2133     if (ctx == NULL) {
2134         v->not_found = 1;
2135         return NGX_OK;
2136     }
2137
2138     v->len = ctx->vars.host_header.len;
2139     v->valid = 1;
2140     v->no_cacheable = 0;
2141     v->not_found = 0;
2142     v->data = ctx->vars.host_header.data;
2143
2144     return NGX_OK;
2145 }
2146
2147
2148 static ngx_int_t
2149 ngx_http_proxy_port_variable(ngx_http_request_t *r,
2150     ngx_http_variable_value_t *v, uintptr_t data)
2151 {
2152     ngx_http_proxy_ctx_t  *ctx;
2153
2154     ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);
2155
2156     if (ctx == NULL) {
2157         v->not_found = 1;
2158         return NGX_OK;
2159     }
2160
2161     v->len = ctx->vars.port.len;
2162     v->valid = 1;
2163     v->no_cacheable = 0;
2164     v->not_found = 0;
2165     v->data = ctx->vars.port.data;
2166
2167     return NGX_OK;
2168 }
2169
2170
2171 static ngx_int_t
2172 ngx_http_proxy_add_x_forwarded_for_variable(ngx_http_request_t *r,
2173     ngx_http_variable_value_t *v, uintptr_t data)
2174 {
2175     size_t             len;
2176     u_char            *p;
2177     ngx_uint_t         i, n;
2178     ngx_table_elt_t  **h;
2179
2180     v->valid = 1;
2181     v->no_cacheable = 0;
2182     v->not_found = 0;
2183
2184     n = r->headers_in.x_forwarded_for.nelts;
2185     h = r->headers_in.x_forwarded_for.elts;
2186
2187     len = 0;
2188
2189     for (i = 0; i < n; i++) {
```

```
2190            len += h[i]->value.len + sizeof(", ") - 1;
2191        }
2192
2193        if (len == 0) {
2194            v->len = r->connection->addr_text.len;
2195            v->data = r->connection->addr_text.data;
2196            return NGX_OK;
2197        }
2198
2199        len += r->connection->addr_text.len;
2200
2201        p = ngx_pnalloc(r->pool, len);
2202        if (p == NULL) {
2203            return NGX_ERROR;
2204        }
2205
2206        v->len = len;
2207        v->data = p;
2208
2209        for (i = 0; i < n; i++) {
2210            p = ngx_copy(p, h[i]->value.data, h[i]->value.len);
2211            *p++ = ','; *p++ = ' ';
2212        }
2213
2214        ngx_memcpy(p, r->connection->addr_text.data, r->connection->addr_text.len);
2215
2216        return NGX_OK;
2217    }
2218
2219
2220    static ngx_int_t
2221    ngx_http_proxy_internal_body_length_variable(ngx_http_request_t *r,
2222        ngx_http_variable_value_t *v, uintptr_t data)
2223    {
2224        ngx_http_proxy_ctx_t  *ctx;
2225
2226        ctx = ngx_http_get_module_ctx(r, ngx_http_proxy_module);
2227
2228        if (ctx == NULL || ctx->internal_body_length < 0) {
2229            v->not_found = 1;
2230            return NGX_OK;
2231        }
2232
2233        v->valid = 1;
2234        v->no_cacheable = 0;
2235        v->not_found = 0;
2236
2237        v->data = ngx_pnalloc(r->pool, NGX_OFF_T_LEN);
2238
2239        if (v->data == NULL) {
2240            return NGX_ERROR;
2241        }
2242
2243        v->len = ngx_sprintf(v->data, "%O", ctx->internal_body_length) - v->data;
2244
2245        return NGX_OK;
2246    }
2247
2248
2249    static ngx_int_t
2250    ngx_http_proxy_rewrite_redirect(ngx_http_request_t *r, ngx_table_elt_t *h,
2251        size_t prefix)
2252    {
2253        size_t                     len;
2254        ngx_int_t                  rc;
2255        ngx_uint_t                 i;
2256        ngx_http_proxy_rewrite_t   *pr;
2257        ngx_http_proxy_loc_conf_t  *plcf;
2258
2259        plcf = ngx_http_get_module_loc_conf(r, ngx_http_proxy_module);
2260
2261        pr = plcf->redirects->elts;
2262
2263        if (pr == NULL) {
2264            return NGX_DECLINED;
2265        }
```

```
2266
2267        len = h->value.len - prefix;
2268
2269        for (i = 0; i < plcf->redirects->nelts; i++) {
2270            rc = pr[i].handler(r, h, prefix, len, &pr[i]);
2271
2272            if (rc != NGX_DECLINED) {
2273                return rc;
2274            }
2275        }
2276
2277        return NGX_DECLINED;
2278    }
2279
2280
2281    static ngx_int_t
2282    ngx_http_proxy_rewrite_cookie(ngx_http_request_t *r, ngx_table_elt_t *h)
2283    {
2284        size_t                     prefix;
2285        u_char                    *p;
2286        ngx_int_t                  rc, rv;
2287        ngx_http_proxy_loc_conf_t  *plcf;
2288
2289        p = (u_char *) ngx_strchr(h->value.data, ';');
2290        if (p == NULL) {
2291            return NGX_DECLINED;
2292        }
2293
2294        prefix = p + 1 - h->value.data;
2295
2296        rv = NGX_DECLINED;
2297
2298        plcf = ngx_http_get_module_loc_conf(r, ngx_http_proxy_module);
2299
2300        if (plcf->cookie_domains) {
2301            p = ngx_strcasestrn(h->value.data + prefix, "domain=", 7 - 1);
2302
2303            if (p) {
2304                rc = ngx_http_proxy_rewrite_cookie_value(r, h, p + 7,
2305                                                         plcf->cookie_domains);
2306                if (rc == NGX_ERROR) {
2307                    return NGX_ERROR;
2308                }
2309
2310                if (rc != NGX_DECLINED) {
2311                    rv = rc;
2312                }
2313            }
2314        }
2315
2316        if (plcf->cookie_paths) {
2317            p = ngx_strcasestrn(h->value.data + prefix, "path=", 5 - 1);
2318
2319            if (p) {
2320                rc = ngx_http_proxy_rewrite_cookie_value(r, h, p + 5,
2321                                                         plcf->cookie_paths);
2322                if (rc == NGX_ERROR) {
2323                    return NGX_ERROR;
2324                }
2325
2326                if (rc != NGX_DECLINED) {
2327                    rv = rc;
2328                }
2329            }
2330        }
2331
2332        return rv;
2333    }
2334
2335
2336    static ngx_int_t
2337    ngx_http_proxy_rewrite_cookie_value(ngx_http_request_t *r, ngx_table_elt_t *h,
2338        u_char *value, ngx_array_t *rewrites)
2339    {
2340        size_t                     len, prefix;
2341        u_char                    *p;
```

```
    ngx_int_t                  rc;
    ngx_uint_t                 i;
    ngx_http_proxy_rewrite_t  *pr;

    prefix = value - h->value.data;

    p = (u_char *) ngx_strchr(value, ';');

    len = p ? (size_t) (p - value) : (h->value.len - prefix);

    pr = rewrites->elts;

    for (i = 0; i < rewrites->nelts; i++) {
        rc = pr[i].handler(r, h, prefix, len, &pr[i]);

        if (rc != NGX_DECLINED) {
            return rc;
        }
    }

    return NGX_DECLINED;
}


static ngx_int_t
ngx_http_proxy_rewrite_complex_handler(ngx_http_request_t *r,
    ngx_table_elt_t *h, size_t prefix, size_t len, ngx_http_proxy_rewrite_t *pr)
{
    ngx_str_t  pattern, replacement;

    if (ngx_http_complex_value(r, &pr->pattern.complex, &pattern) != NGX_OK) {
        return NGX_ERROR;
    }

    if (pattern.len > len
        || ngx_rstrncmp(h->value.data + prefix, pattern.data,
                        pattern.len) != 0)
    {
        return NGX_DECLINED;
    }

    if (ngx_http_complex_value(r, &pr->replacement, &replacement) != NGX_OK) {
        return NGX_ERROR;
    }

    return ngx_http_proxy_rewrite(r, h, prefix, pattern.len, &replacement);
}


#if (NGX_PCRE)

static ngx_int_t
ngx_http_proxy_rewrite_regex_handler(ngx_http_request_t *r, ngx_table_elt_t *h,
    size_t prefix, size_t len, ngx_http_proxy_rewrite_t *pr)
{
    ngx_str_t  pattern, replacement;

    pattern.len = len;
    pattern.data = h->value.data + prefix;

    if (ngx_http_regex_exec(r, pr->pattern.regex, &pattern) != NGX_OK) {
        return NGX_DECLINED;
    }

    if (ngx_http_complex_value(r, &pr->replacement, &replacement) != NGX_OK) {
        return NGX_ERROR;
    }

    if (prefix == 0 && h->value.len == len) {
        h->value = replacement;
        return NGX_OK;
    }

    return ngx_http_proxy_rewrite(r, h, prefix, len, &replacement);
}

```

```
#endif


static ngx_int_t
ngx_http_proxy_rewrite_domain_handler(ngx_http_request_t *r,
    ngx_table_elt_t *h, size_t prefix, size_t len, ngx_http_proxy_rewrite_t *pr)
{
    u_char     *p;
    ngx_str_t   pattern, replacement;

    if (ngx_http_complex_value(r, &pr->pattern.complex, &pattern) != NGX_OK) {
        return NGX_ERROR;
    }

    p = h->value.data + prefix;

    if (p[0] == '.') {
        p++;
        prefix++;
        len--;
    }

    if (pattern.len != len || ngx_rstrncasecmp(pattern.data, p, len) != 0) {
        return NGX_DECLINED;
    }

    if (ngx_http_complex_value(r, &pr->replacement, &replacement) != NGX_OK) {
        return NGX_ERROR;
    }

    return ngx_http_proxy_rewrite(r, h, prefix, len, &replacement);
}


static ngx_int_t
ngx_http_proxy_rewrite(ngx_http_request_t *r, ngx_table_elt_t *h, size_t prefix,
    size_t len, ngx_str_t *replacement)
{
    u_char  *p, *data;
    size_t   new_len;

    new_len = replacement->len + h->value.len - len;

    if (replacement->len > len) {

        data = ngx_pnalloc(r->pool, new_len + 1);
        if (data == NULL) {
            return NGX_ERROR;
        }

        p = ngx_copy(data, h->value.data, prefix);
        p = ngx_copy(p, replacement->data, replacement->len);

        ngx_memcpy(p, h->value.data + prefix + len,
                   h->value.len - len - prefix + 1);

        h->value.data = data;

    } else {
        p = ngx_copy(h->value.data + prefix, replacement->data,
                     replacement->len);

        ngx_memmove(p, h->value.data + prefix + len,
                    h->value.len - len - prefix + 1);
    }

    h->value.len = new_len;

    return NGX_OK;
}


static ngx_int_t
ngx_http_proxy_add_variables(ngx_conf_t *cf)
{
    ngx_http_variable_t  *var, *v;
```

```
2494
2495        for (v = ngx_http_proxy_vars; v->name.len; v++) {
2496            var = ngx_http_add_variable(cf, &v->name, v->flags);
2497            if (var == NULL) {
2498                return NGX_ERROR;
2499            }
2500
2501            var->get_handler = v->get_handler;
2502            var->data = v->data;
2503        }
2504
2505        return NGX_OK;
2506    }
2507
2508
2509    static void *
2510    ngx_http_proxy_create_main_conf(ngx_conf_t *cf)
2511    {
2512        ngx_http_proxy_main_conf_t  *conf;
2513
2514        conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_proxy_main_conf_t));
2515        if (conf == NULL) {
2516            return NULL;
2517        }
2518
2519    #if (NGX_HTTP_CACHE)
2520        if (ngx_array_init(&conf->caches, cf->pool, 4,
2521                           sizeof(ngx_http_file_cache_t *))
2522            != NGX_OK)
2523        {
2524            return NULL;
2525        }
2526    #endif
2527
2528        return conf;
2529    }
2530
2531
2532    static void *
2533    ngx_http_proxy_create_loc_conf(ngx_conf_t *cf)
2534    {
2535        ngx_http_proxy_loc_conf_t  *conf;
2536
2537        conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_proxy_loc_conf_t));
2538        if (conf == NULL) {
2539            return NULL;
2540        }
2541
2542        /*
2543         * set by ngx_pcalloc():
2544         *
2545         *     conf->upstream.bufs.num = 0;
2546         *     conf->upstream.ignore_headers = 0;
2547         *     conf->upstream.next_upstream = 0;
2548         *     conf->upstream.cache_zone = NULL;
2549         *     conf->upstream.cache_use_stale = 0;
2550         *     conf->upstream.cache_methods = 0;
2551         *     conf->upstream.temp_path = NULL;
2552         *     conf->upstream.hide_headers_hash = { NULL, 0 };
2553         *     conf->upstream.uri = { 0, NULL };
2554         *     conf->upstream.location = NULL;
2555         *     conf->upstream.store_lengths = NULL;
2556         *     conf->upstream.store_values = NULL;
2557         *     conf->upstream.ssl_name = NULL;
2558         *
2559         *     conf->method = { 0, NULL };
2560         *     conf->headers_source = NULL;
2561         *     conf->headers.lengths = NULL;
2562         *     conf->headers.values = NULL;
2563         *     conf->headers.hash = { NULL, 0 };
2564         *     conf->headers_cache.lengths = NULL;
2565         *     conf->headers_cache.values = NULL;
2566         *     conf->headers_cache.hash = { NULL, 0 };
2567         *     conf->body_lengths = NULL;
2568         *     conf->body_values = NULL;
2569         *     conf->body_source = { 0, NULL };
```

```
2570        *     conf->redirects = NULL;
2571        *     conf->ssl = 0;
2572        *     conf->ssl_protocols = 0;
2573        *     conf->ssl_ciphers = { 0, NULL };
2574        *     conf->ssl_trusted_certificate = { 0, NULL };
2575        *     conf->ssl_crl = { 0, NULL };
2576        *     conf->ssl_certificate = { 0, NULL };
2577        *     conf->ssl_certificate_key = { 0, NULL };
2578        */
2579
2580       conf->upstream.store = NGX_CONF_UNSET;
2581       conf->upstream.store_access = NGX_CONF_UNSET_UINT;
2582       conf->upstream.next_upstream_tries = NGX_CONF_UNSET_UINT;
2583       conf->upstream.buffering = NGX_CONF_UNSET;
2584       conf->upstream.ignore_client_abort = NGX_CONF_UNSET;
2585       conf->upstream.force_ranges = NGX_CONF_UNSET;
2586
2587       conf->upstream.local = NGX_CONF_UNSET_PTR;
2588
2589       conf->upstream.connect_timeout = NGX_CONF_UNSET_MSEC;
2590       conf->upstream.send_timeout = NGX_CONF_UNSET_MSEC;
2591       conf->upstream.read_timeout = NGX_CONF_UNSET_MSEC;
2592       conf->upstream.next_upstream_timeout = NGX_CONF_UNSET_MSEC;
2593
2594       conf->upstream.send_lowat = NGX_CONF_UNSET_SIZE;
2595       conf->upstream.buffer_size = NGX_CONF_UNSET_SIZE;
2596       conf->upstream.limit_rate = NGX_CONF_UNSET_SIZE;
2597
2598       conf->upstream.busy_buffers_size_conf = NGX_CONF_UNSET_SIZE;
2599       conf->upstream.max_temp_file_size_conf = NGX_CONF_UNSET_SIZE;
2600       conf->upstream.temp_file_write_size_conf = NGX_CONF_UNSET_SIZE;
2601
2602       conf->upstream.pass_request_headers = NGX_CONF_UNSET;
2603       conf->upstream.pass_request_body = NGX_CONF_UNSET;
2604
2605   #if (NGX_HTTP_CACHE)
2606       conf->upstream.cache = NGX_CONF_UNSET;
2607       conf->upstream.cache_min_uses = NGX_CONF_UNSET_UINT;
2608       conf->upstream.cache_bypass = NGX_CONF_UNSET_PTR;
2609       conf->upstream.no_cache = NGX_CONF_UNSET_PTR;
2610       conf->upstream.cache_valid = NGX_CONF_UNSET_PTR;
2611       conf->upstream.cache_lock = NGX_CONF_UNSET;
2612       conf->upstream.cache_lock_timeout = NGX_CONF_UNSET_MSEC;
2613       conf->upstream.cache_lock_age = NGX_CONF_UNSET_MSEC;
2614       conf->upstream.cache_revalidate = NGX_CONF_UNSET;
2615   #endif
2616
2617       conf->upstream.hide_headers = NGX_CONF_UNSET_PTR;
2618       conf->upstream.pass_headers = NGX_CONF_UNSET_PTR;
2619
2620       conf->upstream.intercept_errors = NGX_CONF_UNSET;
2621
2622   #if (NGX_HTTP_SSL)
2623       conf->upstream.ssl_session_reuse = NGX_CONF_UNSET;
2624       conf->upstream.ssl_server_name = NGX_CONF_UNSET;
2625       conf->upstream.ssl_verify = NGX_CONF_UNSET;
2626       conf->ssl_verify_depth = NGX_CONF_UNSET_UINT;
2627       conf->ssl_passwords = NGX_CONF_UNSET_PTR;
2628   #endif
2629
2630       /* "proxy_cyclic_temp_file" is disabled */
2631       conf->upstream.cyclic_temp_file = 0;
2632
2633       conf->redirect = NGX_CONF_UNSET;
2634       conf->upstream.change_buffering = 1;
2635
2636       conf->cookie_domains = NGX_CONF_UNSET_PTR;
2637       conf->cookie_paths = NGX_CONF_UNSET_PTR;
2638
2639       conf->http_version = NGX_CONF_UNSET_UINT;
2640
2641       conf->headers_hash_max_size = NGX_CONF_UNSET_UINT;
2642       conf->headers_hash_bucket_size = NGX_CONF_UNSET_UINT;
2643
2644       ngx_str_set(&conf->upstream.module, "proxy");
2645
```

```
    return conf;
}


static char *
ngx_http_proxy_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_proxy_loc_conf_t *prev = parent;
    ngx_http_proxy_loc_conf_t *conf = child;

    u_char                     *p;
    size_t                      size;
    ngx_int_t                   rc;
    ngx_hash_init_t             hash;
    ngx_http_core_loc_conf_t   *clcf;
    ngx_http_proxy_rewrite_t   *pr;
    ngx_http_script_compile_t   sc;

#if (NGX_HTTP_CACHE)

    if (conf->upstream.store > 0) {
        conf->upstream.cache = 0;
    }

    if (conf->upstream.cache > 0) {
        conf->upstream.store = 0;
    }

#endif

    if (conf->upstream.store == NGX_CONF_UNSET) {
        ngx_conf_merge_value(conf->upstream.store,
                             prev->upstream.store, 0);

        conf->upstream.store_lengths = prev->upstream.store_lengths;
        conf->upstream.store_values = prev->upstream.store_values;
    }

    ngx_conf_merge_uint_value(conf->upstream.store_access,
                              prev->upstream.store_access, 0600);

    ngx_conf_merge_uint_value(conf->upstream.next_upstream_tries,
                              prev->upstream.next_upstream_tries, 0);

    ngx_conf_merge_value(conf->upstream.buffering,
                              prev->upstream.buffering, 1);

    ngx_conf_merge_value(conf->upstream.ignore_client_abort,
                              prev->upstream.ignore_client_abort, 0);

    ngx_conf_merge_value(conf->upstream.force_ranges,
                              prev->upstream.force_ranges, 0);

    ngx_conf_merge_ptr_value(conf->upstream.local,
                              prev->upstream.local, NULL);

    ngx_conf_merge_msec_value(conf->upstream.connect_timeout,
                              prev->upstream.connect_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.send_timeout,
                              prev->upstream.send_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.read_timeout,
                              prev->upstream.read_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.next_upstream_timeout,
                              prev->upstream.next_upstream_timeout, 0);

    ngx_conf_merge_size_value(conf->upstream.send_lowat,
                              prev->upstream.send_lowat, 0);

    ngx_conf_merge_size_value(conf->upstream.buffer_size,
                              prev->upstream.buffer_size,
                              (size_t) ngx_pagesize);

    ngx_conf_merge_size_value(conf->upstream.limit_rate,
```

```
                              prev->upstream.limit_rate, 0);

    ngx_conf_merge_bufs_value(conf->upstream.bufs, prev->upstream.bufs,
                              8, ngx_pagesize);

    if (conf->upstream.bufs.num < 2) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "there must be at least 2 \"proxy_buffers\"");
        return NGX_CONF_ERROR;
    }


    size = conf->upstream.buffer_size;
    if (size < conf->upstream.bufs.size) {
        size = conf->upstream.bufs.size;
    }


    ngx_conf_merge_size_value(conf->upstream.busy_buffers_size_conf,
                              prev->upstream.busy_buffers_size_conf,
                              NGX_CONF_UNSET_SIZE);

    if (conf->upstream.busy_buffers_size_conf == NGX_CONF_UNSET_SIZE) {
        conf->upstream.busy_buffers_size = 2 * size;
    } else {
        conf->upstream.busy_buffers_size =
                                        conf->upstream.busy_buffers_size_conf;
    }

    if (conf->upstream.busy_buffers_size < size) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
             "\"proxy_busy_buffers_size\" must be equal to or greater than "
             "the maximum of the value of \"proxy_buffer_size\" and "
             "one of the \"proxy_buffers\"");

        return NGX_CONF_ERROR;
    }

    if (conf->upstream.busy_buffers_size
        > (conf->upstream.bufs.num - 1) * conf->upstream.bufs.size)
    {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
             "\"proxy_busy_buffers_size\" must be less than "
             "the size of all \"proxy_buffers\" minus one buffer");

        return NGX_CONF_ERROR;
    }


    ngx_conf_merge_size_value(conf->upstream.temp_file_write_size_conf,
                              prev->upstream.temp_file_write_size_conf,
                              NGX_CONF_UNSET_SIZE);

    if (conf->upstream.temp_file_write_size_conf == NGX_CONF_UNSET_SIZE) {
        conf->upstream.temp_file_write_size = 2 * size;
    } else {
        conf->upstream.temp_file_write_size =
                                      conf->upstream.temp_file_write_size_conf;
    }

    if (conf->upstream.temp_file_write_size < size) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
             "\"proxy_temp_file_write_size\" must be equal to or greater "
             "than the maximum of the value of \"proxy_buffer_size\" and "
             "one of the \"proxy_buffers\"");

        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_size_value(conf->upstream.max_temp_file_size_conf,
                              prev->upstream.max_temp_file_size_conf,
                              NGX_CONF_UNSET_SIZE);

    if (conf->upstream.max_temp_file_size_conf == NGX_CONF_UNSET_SIZE) {
        conf->upstream.max_temp_file_size = 1024 * 1024 * 1024;
    } else {
```

```
2798        conf->upstream.max_temp_file_size =
2799                                     conf->upstream.max_temp_file_size_conf;
2800    }
2801
2802    if (conf->upstream.max_temp_file_size != 0
2803        && conf->upstream.max_temp_file_size < size)
2804    {
2805        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
2806             "\"proxy_max_temp_file_size\" must be equal to zero to disable "
2807             "temporary files usage or must be equal to or greater than "
2808             "the maximum of the value of \"proxy_buffer_size\" and "
2809             "one of the \"proxy_buffers\"");
2810
2811        return NGX_CONF_ERROR;
2812    }
2813
2814
2815    ngx_conf_merge_bitmask_value(conf->upstream.ignore_headers,
2816                              prev->upstream.ignore_headers,
2817                              NGX_CONF_BITMASK_SET);
2818
2819
2820    ngx_conf_merge_bitmask_value(conf->upstream.next_upstream,
2821                              prev->upstream.next_upstream,
2822                              (NGX_CONF_BITMASK_SET
2823                               |NGX_HTTP_UPSTREAM_FT_ERROR
2824                               |NGX_HTTP_UPSTREAM_FT_TIMEOUT));
2825
2826    if (conf->upstream.next_upstream & NGX_HTTP_UPSTREAM_FT_OFF) {
2827        conf->upstream.next_upstream = NGX_CONF_BITMASK_SET
2828                                       |NGX_HTTP_UPSTREAM_FT_OFF;
2829    }
2830
2831    if (ngx_conf_merge_path_value(cf, &conf->upstream.temp_path,
2832                              prev->upstream.temp_path,
2833                              &ngx_http_proxy_temp_path)
2834        != NGX_OK)
2835    {
2836        return NGX_CONF_ERROR;
2837    }
2838
2839
2840#if (NGX_HTTP_CACHE)
2841
2842    if (conf->upstream.cache == NGX_CONF_UNSET) {
2843        ngx_conf_merge_value(conf->upstream.cache,
2844                              prev->upstream.cache, 0);
2845
2846        conf->upstream.cache_zone = prev->upstream.cache_zone;
2847        conf->upstream.cache_value = prev->upstream.cache_value;
2848    }
2849
2850    if (conf->upstream.cache_zone && conf->upstream.cache_zone->data == NULL) {
2851        ngx_shm_zone_t  *shm_zone;
2852
2853        shm_zone = conf->upstream.cache_zone;
2854
2855        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
2856                           "\"proxy_cache\" zone \"%V\" is unknown",
2857                           &shm_zone->shm.name);
2858
2859        return NGX_CONF_ERROR;
2860    }
2861
2862    ngx_conf_merge_uint_value(conf->upstream.cache_min_uses,
2863                              prev->upstream.cache_min_uses, 1);
2864
2865    ngx_conf_merge_bitmask_value(conf->upstream.cache_use_stale,
2866                              prev->upstream.cache_use_stale,
2867                              (NGX_CONF_BITMASK_SET
2868                               |NGX_HTTP_UPSTREAM_FT_OFF));
2869
2870    if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_OFF) {
2871        conf->upstream.cache_use_stale = NGX_CONF_BITMASK_SET
2872                                         |NGX_HTTP_UPSTREAM_FT_OFF;
2873    }
```

```
2874
2875        if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_ERROR) {
2876            conf->upstream.cache_use_stale |= NGX_HTTP_UPSTREAM_FT_NOLIVE;
2877        }
2878
2879        if (conf->upstream.cache_methods == 0) {
2880            conf->upstream.cache_methods = prev->upstream.cache_methods;
2881        }
2882
2883        conf->upstream.cache_methods |= NGX_HTTP_GET|NGX_HTTP_HEAD;
2884
2885        ngx_conf_merge_ptr_value(conf->upstream.cache_bypass,
2886                                 prev->upstream.cache_bypass, NULL);
2887
2888        ngx_conf_merge_ptr_value(conf->upstream.no_cache,
2889                                 prev->upstream.no_cache, NULL);
2890
2891        ngx_conf_merge_ptr_value(conf->upstream.cache_valid,
2892                                 prev->upstream.cache_valid, NULL);
2893
2894        if (conf->cache_key.value.data == NULL) {
2895            conf->cache_key = prev->cache_key;
2896        }
2897
2898        ngx_conf_merge_value(conf->upstream.cache_lock,
2899                             prev->upstream.cache_lock, 0);
2900
2901        ngx_conf_merge_msec_value(conf->upstream.cache_lock_timeout,
2902                             prev->upstream.cache_lock_timeout, 5000);
2903
2904        ngx_conf_merge_msec_value(conf->upstream.cache_lock_age,
2905                             prev->upstream.cache_lock_age, 5000);
2906
2907        ngx_conf_merge_value(conf->upstream.cache_revalidate,
2908                             prev->upstream.cache_revalidate, 0);
2909
2910    #endif
2911
2912        ngx_conf_merge_str_value(conf->method, prev->method, "");
2913
2914        if (conf->method.len
2915            && conf->method.data[conf->method.len - 1] != ' ')
2916        {
2917            conf->method.data[conf->method.len] = ' ';
2918            conf->method.len++;
2919        }
2920
2921        ngx_conf_merge_value(conf->upstream.pass_request_headers,
2922                             prev->upstream.pass_request_headers, 1);
2923        ngx_conf_merge_value(conf->upstream.pass_request_body,
2924                             prev->upstream.pass_request_body, 1);
2925
2926        ngx_conf_merge_value(conf->upstream.intercept_errors,
2927                             prev->upstream.intercept_errors, 0);
2928
2929    #if (NGX_HTTP_SSL)
2930
2931        ngx_conf_merge_value(conf->upstream.ssl_session_reuse,
2932                             prev->upstream.ssl_session_reuse, 1);
2933
2934        ngx_conf_merge_bitmask_value(conf->ssl_protocols, prev->ssl_protocols,
2935                                     (NGX_CONF_BITMASK_SET|NGX_SSL_SSLv3
2936                                      |NGX_SSL_TLSv1|NGX_SSL_TLSv1_1
2937                                      |NGX_SSL_TLSv1_2));
2938
2939        ngx_conf_merge_str_value(conf->ssl_ciphers, prev->ssl_ciphers,
2940                                 "DEFAULT");
2941
2942        if (conf->upstream.ssl_name == NULL) {
2943            conf->upstream.ssl_name = prev->upstream.ssl_name;
2944        }
2945
2946        ngx_conf_merge_value(conf->upstream.ssl_server_name,
2947                             prev->upstream.ssl_server_name, 0);
2948        ngx_conf_merge_value(conf->upstream.ssl_verify,
2949                             prev->upstream.ssl_verify, 0);
```

```
    ngx_conf_merge_uint_value(conf->ssl_verify_depth,
                              prev->ssl_verify_depth, 1);
    ngx_conf_merge_str_value(conf->ssl_trusted_certificate,
                              prev->ssl_trusted_certificate, "");
    ngx_conf_merge_str_value(conf->ssl_crl, prev->ssl_crl, "");

    ngx_conf_merge_str_value(conf->ssl_certificate,
                              prev->ssl_certificate, "");
    ngx_conf_merge_str_value(conf->ssl_certificate_key,
                              prev->ssl_certificate_key, "");
    ngx_conf_merge_ptr_value(conf->ssl_passwords, prev->ssl_passwords, NULL);

    if (conf->ssl && ngx_http_proxy_set_ssl(cf, conf) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

#endif

    ngx_conf_merge_value(conf->redirect, prev->redirect, 1);

    if (conf->redirect) {

        if (conf->redirects == NULL) {
            conf->redirects = prev->redirects;
        }

        if (conf->redirects == NULL && conf->url.data) {

            conf->redirects = ngx_array_create(cf->pool, 1,
                                               sizeof(ngx_http_proxy_rewrite_t));
            if (conf->redirects == NULL) {
                return NGX_CONF_ERROR;
            }

            pr = ngx_array_push(conf->redirects);
            if (pr == NULL) {
                return NGX_CONF_ERROR;
            }

            ngx_memzero(&pr->pattern.complex,
                        sizeof(ngx_http_complex_value_t));

            ngx_memzero(&pr->replacement, sizeof(ngx_http_complex_value_t));

            pr->handler = ngx_http_proxy_rewrite_complex_handler;

            if (conf->vars.uri.len) {
                pr->pattern.complex.value = conf->url;
                pr->replacement.value = conf->location;

            } else {
                pr->pattern.complex.value.len = conf->url.len
                                                + sizeof("/") - 1;

                p = ngx_pnalloc(cf->pool, pr->pattern.complex.value.len);
                if (p == NULL) {
                    return NGX_CONF_ERROR;
                }

                pr->pattern.complex.value.data = p;

                p = ngx_cpymem(p, conf->url.data, conf->url.len);
                *p = '/';

                ngx_str_set(&pr->replacement.value, "/");
            }
        }
    }

    ngx_conf_merge_ptr_value(conf->cookie_domains, prev->cookie_domains, NULL);

    ngx_conf_merge_ptr_value(conf->cookie_paths, prev->cookie_paths, NULL);

    ngx_conf_merge_uint_value(conf->http_version, prev->http_version,
                              NGX_HTTP_VERSION_10);

```

```
3026     ngx_conf_merge_uint_value(conf->headers_hash_max_size,
3027                               prev->headers_hash_max_size, 512);
3028
3029     ngx_conf_merge_uint_value(conf->headers_hash_bucket_size,
3030                               prev->headers_hash_bucket_size, 64);
3031
3032     conf->headers_hash_bucket_size = ngx_align(conf->headers_hash_bucket_size,
3033                                                ngx_cacheline_size);
3034
3035     hash.max_size = conf->headers_hash_max_size;
3036     hash.bucket_size = conf->headers_hash_bucket_size;
3037     hash.name = "proxy_headers_hash";
3038
3039     if (ngx_http_upstream_hide_headers_hash(cf, &conf->upstream,
3040             &prev->upstream, ngx_http_proxy_hide_headers, &hash)
3041         != NGX_OK)
3042     {
3043         return NGX_CONF_ERROR;
3044     }
3045
3046     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
3047
3048     if (clcf->noname
3049         && conf->upstream.upstream == NULL && conf->proxy_lengths == NULL)
3050     {
3051         conf->upstream.upstream = prev->upstream.upstream;
3052         conf->location = prev->location;
3053         conf->vars = prev->vars;
3054
3055         conf->proxy_lengths = prev->proxy_lengths;
3056         conf->proxy_values = prev->proxy_values;
3057
3058 #if (NGX_HTTP_SSL)
3059         conf->upstream.ssl = prev->upstream.ssl;
3060 #endif
3061     }
3062
3063     if (clcf->lmt_excpt && clcf->handler == NULL
3064         && (conf->upstream.upstream || conf->proxy_lengths))
3065     {
3066         clcf->handler = ngx_http_proxy_handler;
3067     }
3068
3069     if (conf->body_source.data == NULL) {
3070         conf->body_flushes = prev->body_flushes;
3071         conf->body_source = prev->body_source;
3072         conf->body_lengths = prev->body_lengths;
3073         conf->body_values = prev->body_values;
3074     }
3075
3076     if (conf->body_source.data && conf->body_lengths == NULL) {
3077
3078         ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
3079
3080         sc.cf = cf;
3081         sc.source = &conf->body_source;
3082         sc.flushes = &conf->body_flushes;
3083         sc.lengths = &conf->body_lengths;
3084         sc.values = &conf->body_values;
3085         sc.complete_lengths = 1;
3086         sc.complete_values = 1;
3087
3088         if (ngx_http_script_compile(&sc) != NGX_OK) {
3089             return NGX_CONF_ERROR;
3090         }
3091     }
3092
3093     if (conf->headers_source == NULL) {
3094         conf->headers = prev->headers;
3095 #if (NGX_HTTP_CACHE)
3096         conf->headers_cache = prev->headers_cache;
3097 #endif
3098         conf->headers_source = prev->headers_source;
3099     }
3100
3101     rc = ngx_http_proxy_init_headers(cf, conf, &conf->headers,
```

```
                                       ngx_http_proxy_headers);
    if (rc != NGX_OK) {
        return NGX_CONF_ERROR;
    }

#if (NGX_HTTP_CACHE)

    if (conf->upstream.cache) {
        rc = ngx_http_proxy_init_headers(cf, conf, &conf->headers_cache,
                                         ngx_http_proxy_cache_headers);
        if (rc != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

#endif

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_proxy_init_headers(ngx_conf_t *cf, ngx_http_proxy_loc_conf_t *conf,
    ngx_http_proxy_headers_t *headers, ngx_keyval_t *default_headers)
{
    u_char                     *p;
    size_t                      size;
    uintptr_t                  *code;
    ngx_uint_t                  i;
    ngx_array_t                 headers_names, headers_merged;
    ngx_keyval_t               *src, *s, *h;
    ngx_hash_key_t             *hk;
    ngx_hash_init_t             hash;
    ngx_http_script_compile_t   sc;
    ngx_http_script_copy_code_t  *copy;

    if (headers->hash.buckets) {
        return NGX_OK;
    }

    if (ngx_array_init(&headers_names, cf->temp_pool, 4, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (ngx_array_init(&headers_merged, cf->temp_pool, 4, sizeof(ngx_keyval_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (conf->headers_source == NULL) {
        conf->headers_source = ngx_array_create(cf->pool, 4,
                                                sizeof(ngx_keyval_t));
        if (conf->headers_source == NULL) {
            return NGX_ERROR;
        }
    }

    headers->lengths = ngx_array_create(cf->pool, 64, 1);
    if (headers->lengths == NULL) {
        return NGX_ERROR;
    }

    headers->values = ngx_array_create(cf->pool, 512, 1);
    if (headers->values == NULL) {
        return NGX_ERROR;
    }

    src = conf->headers_source->elts;
    for (i = 0; i < conf->headers_source->nelts; i++) {

        s = ngx_array_push(&headers_merged);
        if (s == NULL) {
            return NGX_ERROR;
```

```
3178            }
3179
3180            *s = src[i];
3181        }
3182
3183        h = default_headers;
3184
3185        while (h->key.len) {
3186
3187            src = headers_merged.elts;
3188            for (i = 0; i < headers_merged.nelts; i++) {
3189                if (ngx_strcasecmp(h->key.data, src[i].key.data) == 0) {
3190                    goto next;
3191                }
3192            }
3193
3194            s = ngx_array_push(&headers_merged);
3195            if (s == NULL) {
3196                return NGX_ERROR;
3197            }
3198
3199            *s = *h;
3200
3201        next:
3202
3203            h++;
3204        }
3205
3206
3207        src = headers_merged.elts;
3208        for (i = 0; i < headers_merged.nelts; i++) {
3209
3210            hk = ngx_array_push(&headers_names);
3211            if (hk == NULL) {
3212                return NGX_ERROR;
3213            }
3214
3215            hk->key = src[i].key;
3216            hk->key_hash = ngx_hash_key_lc(src[i].key.data, src[i].key.len);
3217            hk->value = (void *) 1;
3218
3219            if (src[i].value.len == 0) {
3220                continue;
3221            }
3222
3223            if (ngx_http_script_variables_count(&src[i].value) == 0) {
3224                copy = ngx_array_push_n(headers->lengths,
3225                                        sizeof(ngx_http_script_copy_code_t));
3226                if (copy == NULL) {
3227                    return NGX_ERROR;
3228                }
3229
3230                copy->code = (ngx_http_script_code_pt)
3231                                                    ngx_http_script_copy_len_code;
3232                copy->len = src[i].key.len + sizeof(": ") - 1
3233                            + src[i].value.len + sizeof(CRLF) - 1;
3234
3235
3236                size = (sizeof(ngx_http_script_copy_code_t)
3237                           + src[i].key.len + sizeof(": ") - 1
3238                           + src[i].value.len + sizeof(CRLF) - 1
3239                           + sizeof(uintptr_t) - 1)
3240                       & ~(sizeof(uintptr_t) - 1);
3241
3242                copy = ngx_array_push_n(headers->values, size);
3243                if (copy == NULL) {
3244                    return NGX_ERROR;
3245                }
3246
3247                copy->code = ngx_http_script_copy_code;
3248                copy->len = src[i].key.len + sizeof(": ") - 1
3249                            + src[i].value.len + sizeof(CRLF) - 1;
3250
3251                p = (u_char *) copy + sizeof(ngx_http_script_copy_code_t);
3252
3253                p = ngx_cpymem(p, src[i].key.data, src[i].key.len);
```

```
            *p++ = ':'; *p++ = ' ';
            p = ngx_cpymem(p, src[i].value.data, src[i].value.len);
            *p++ = CR; *p = LF;

        } else {
            copy = ngx_array_push_n(headers->lengths,
                                    sizeof(ngx_http_script_copy_code_t));
            if (copy == NULL) {
                return NGX_ERROR;
            }

            copy->code = (ngx_http_script_code_pt)
                                                ngx_http_script_copy_len_code;
            copy->len = src[i].key.len + sizeof(": ") - 1;


            size = (sizeof(ngx_http_script_copy_code_t)
                    + src[i].key.len + sizeof(": ") - 1 + sizeof(uintptr_t) - 1)
                    & ~(sizeof(uintptr_t) - 1);

            copy = ngx_array_push_n(headers->values, size);
            if (copy == NULL) {
                return NGX_ERROR;
            }

            copy->code = ngx_http_script_copy_code;
            copy->len = src[i].key.len + sizeof(": ") - 1;

            p = (u_char *) copy + sizeof(ngx_http_script_copy_code_t);
            p = ngx_cpymem(p, src[i].key.data, src[i].key.len);
            *p++ = ':'; *p = ' ';


            ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

            sc.cf = cf;
            sc.source = &src[i].value;
            sc.flushes = &headers->flushes;
            sc.lengths = &headers->lengths;
            sc.values = &headers->values;

            if (ngx_http_script_compile(&sc) != NGX_OK) {
                return NGX_ERROR;
            }


            copy = ngx_array_push_n(headers->lengths,
                                    sizeof(ngx_http_script_copy_code_t));
            if (copy == NULL) {
                return NGX_ERROR;
            }

            copy->code = (ngx_http_script_code_pt)
                                                ngx_http_script_copy_len_code;
            copy->len = sizeof(CRLF) - 1;


            size = (sizeof(ngx_http_script_copy_code_t)
                    + sizeof(CRLF) - 1 + sizeof(uintptr_t) - 1)
                    & ~(sizeof(uintptr_t) - 1);

            copy = ngx_array_push_n(headers->values, size);
            if (copy == NULL) {
                return NGX_ERROR;
            }

            copy->code = ngx_http_script_copy_code;
            copy->len = sizeof(CRLF) - 1;

            p = (u_char *) copy + sizeof(ngx_http_script_copy_code_t);
            *p++ = CR; *p = LF;
        }

        code = ngx_array_push_n(headers->lengths, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
```

```
3330         }
3331
3332         *code = (uintptr_t) NULL;
3333
3334         code = ngx_array_push_n(headers->values, sizeof(uintptr_t));
3335         if (code == NULL) {
3336             return NGX_ERROR;
3337         }
3338
3339         *code = (uintptr_t) NULL;
3340     }
3341
3342     code = ngx_array_push_n(headers->lengths, sizeof(uintptr_t));
3343     if (code == NULL) {
3344         return NGX_ERROR;
3345     }
3346
3347     *code = (uintptr_t) NULL;
3348
3349
3350     hash.hash = &headers->hash;
3351     hash.key = ngx_hash_key_lc;
3352     hash.max_size = conf->headers_hash_max_size;
3353     hash.bucket_size = conf->headers_hash_bucket_size;
3354     hash.name = "proxy_headers_hash";
3355     hash.pool = cf->pool;
3356     hash.temp_pool = NULL;
3357
3358     return ngx_hash_init(&hash, headers_names.elts, headers_names.nelts);
3359 }
3360
3361
3362 static char *
3363 ngx_http_proxy_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3364 {
3365     ngx_http_proxy_loc_conf_t *plcf = conf;
3366
3367     size_t                     add;
3368     u_short                    port;
3369     ngx_str_t                 *value, *url;
3370     ngx_url_t                  u;
3371     ngx_uint_t                 n;
3372     ngx_http_core_loc_conf_t  *clcf;
3373     ngx_http_script_compile_t  sc;
3374
3375     if (plcf->upstream.upstream || plcf->proxy_lengths) {
3376         return "is duplicate";
3377     }
3378
3379     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
3380
3381     clcf->handler = ngx_http_proxy_handler;
3382
3383     if (clcf->name.data[clcf->name.len - 1] == '/') {
3384         clcf->auto_redirect = 1;
3385     }
3386
3387     value = cf->args->elts;
3388
3389     url = &value[1];
3390
3391     n = ngx_http_script_variables_count(url);
3392
3393     if (n) {
3394
3395         ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
3396
3397         sc.cf = cf;
3398         sc.source = url;
3399         sc.lengths = &plcf->proxy_lengths;
3400         sc.values = &plcf->proxy_values;
3401         sc.variables = n;
3402         sc.complete_lengths = 1;
3403         sc.complete_values = 1;
3404
3405         if (ngx_http_script_compile(&sc) != NGX_OK) {
```

```
3406            return NGX_CONF_ERROR;
3407        }
3408
3409 #if (NGX_HTTP_SSL)
3410        plcf->ssl = 1;
3411 #endif
3412
3413        return NGX_CONF_OK;
3414    }
3415
3416    if (ngx_strncasecmp(url->data, (u_char *) "http://", 7) == 0) {
3417        add = 7;
3418        port = 80;
3419
3420    } else if (ngx_strncasecmp(url->data, (u_char *) "https://", 8) == 0) {
3421
3422 #if (NGX_HTTP_SSL)
3423        plcf->ssl = 1;
3424
3425        add = 8;
3426        port = 443;
3427 #else
3428        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3429                           "https protocol requires SSL support");
3430        return NGX_CONF_ERROR;
3431 #endif
3432
3433    } else {
3434        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid URL prefix");
3435        return NGX_CONF_ERROR;
3436    }
3437
3438    ngx_memzero(&u, sizeof(ngx_url_t));
3439
3440    u.url.len = url->len - add;
3441    u.url.data = url->data + add;
3442    u.default_port = port;
3443    u.uri_part = 1;
3444    u.no_resolve = 1;
3445
3446    plcf->upstream.upstream = ngx_http_upstream_add(cf, &u, 0);
3447    if (plcf->upstream.upstream == NULL) {
3448        return NGX_CONF_ERROR;
3449    }
3450
3451    plcf->vars.schema.len = add;
3452    plcf->vars.schema.data = url->data;
3453    plcf->vars.key_start = plcf->vars.schema;
3454
3455    ngx_http_proxy_set_vars(&u, &plcf->vars);
3456
3457    plcf->location = clcf->name;
3458
3459    if (clcf->named
3460 #if (NGX_PCRE)
3461        || clcf->regex
3462 #endif
3463        || clcf->noname)
3464    {
3465        if (plcf->vars.uri.len) {
3466            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3467                               "\"proxy_pass\" cannot have URI part in "
3468                               "location given by regular expression, "
3469                               "or inside named location, "
3470                               "or inside \"if\" statement, "
3471                               "or inside \"limit_except\" block");
3472            return NGX_CONF_ERROR;
3473        }
3474
3475        plcf->location.len = 0;
3476    }
3477
3478    plcf->url = *url;
3479
3480    return NGX_CONF_OK;
3481 }
```

```
3482
3483
3484 static char *
3485 ngx_http_proxy_redirect(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3486 {
3487     ngx_http_proxy_loc_conf_t *plcf = conf;
3488
3489     u_char                           *p;
3490     ngx_str_t                        *value;
3491     ngx_http_proxy_rewrite_t         *pr;
3492     ngx_http_compile_complex_value_t  ccv;
3493
3494     if (plcf->redirect == 0) {
3495         return NGX_CONF_OK;
3496     }
3497
3498     plcf->redirect = 1;
3499
3500     value = cf->args->elts;
3501
3502     if (cf->args->nelts == 2) {
3503         if (ngx_strcmp(value[1].data, "off") == 0) {
3504             plcf->redirect = 0;
3505             plcf->redirects = NULL;
3506             return NGX_CONF_OK;
3507         }
3508
3509         if (ngx_strcmp(value[1].data, "false") == 0) {
3510             ngx_conf_log_error(NGX_LOG_ERR, cf, 0,
3511                                "invalid parameter \"false\", use \"off\" instead");
3512             plcf->redirect = 0;
3513             plcf->redirects = NULL;
3514             return NGX_CONF_OK;
3515         }
3516
3517         if (ngx_strcmp(value[1].data, "default") != 0) {
3518             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3519                                "invalid parameter \"%V\"", &value[1]);
3520             return NGX_CONF_ERROR;
3521         }
3522     }
3523
3524     if (plcf->redirects == NULL) {
3525         plcf->redirects = ngx_array_create(cf->pool, 1,
3526                                            sizeof(ngx_http_proxy_rewrite_t));
3527         if (plcf->redirects == NULL) {
3528             return NGX_CONF_ERROR;
3529         }
3530     }
3531
3532     pr = ngx_array_push(plcf->redirects);
3533     if (pr == NULL) {
3534         return NGX_CONF_ERROR;
3535     }
3536
3537     if (ngx_strcmp(value[1].data, "default") == 0) {
3538         if (plcf->proxy_lengths) {
3539             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3540                                "\"proxy_redirect default\" cannot be used "
3541                                "with \"proxy_pass\" directive with variables");
3542             return NGX_CONF_ERROR;
3543         }
3544
3545         if (plcf->url.data == NULL) {
3546             ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3547                                "\"proxy_redirect default\" should be placed "
3548                                "after the \"proxy_pass\" directive");
3549             return NGX_CONF_ERROR;
3550         }
3551
3552         pr->handler = ngx_http_proxy_rewrite_complex_handler;
3553
3554         ngx_memzero(&pr->pattern.complex, sizeof(ngx_http_complex_value_t));
3555
3556         ngx_memzero(&pr->replacement, sizeof(ngx_http_complex_value_t));
3557
```

```
3558            if (plcf->vars.uri.len) {
3559                pr->pattern.complex.value = plcf->url;
3560                pr->replacement.value = plcf->location;
3561
3562            } else {
3563                pr->pattern.complex.value.len = plcf->url.len + sizeof("/") - 1;
3564
3565                p = ngx_pnalloc(cf->pool, pr->pattern.complex.value.len);
3566                if (p == NULL) {
3567                    return NGX_CONF_ERROR;
3568                }
3569
3570                pr->pattern.complex.value.data = p;
3571
3572                p = ngx_cpymem(p, plcf->url.data, plcf->url.len);
3573                *p = '/';
3574
3575                ngx_str_set(&pr->replacement.value, "/");
3576            }
3577
3578            return NGX_CONF_OK;
3579        }
3580
3581
3582        if (value[1].data[0] == '~') {
3583            value[1].len--;
3584            value[1].data++;
3585
3586            if (value[1].data[0] == '*') {
3587                value[1].len--;
3588                value[1].data++;
3589
3590                if (ngx_http_proxy_rewrite_regex(cf, pr, &value[1], 1) != NGX_OK) {
3591                    return NGX_CONF_ERROR;
3592                }
3593
3594            } else {
3595                if (ngx_http_proxy_rewrite_regex(cf, pr, &value[1], 0) != NGX_OK) {
3596                    return NGX_CONF_ERROR;
3597                }
3598            }
3599
3600        } else {
3601
3602            ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3603
3604            ccv.cf = cf;
3605            ccv.value = &value[1];
3606            ccv.complex_value = &pr->pattern.complex;
3607
3608            if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3609                return NGX_CONF_ERROR;
3610            }
3611
3612            pr->handler = ngx_http_proxy_rewrite_complex_handler;
3613        }
3614
3615
3616        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3617
3618        ccv.cf = cf;
3619        ccv.value = &value[2];
3620        ccv.complex_value = &pr->replacement;
3621
3622        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3623            return NGX_CONF_ERROR;
3624        }
3625
3626        return NGX_CONF_OK;
3627    }
3628
3629
3630    static char *
3631    ngx_http_proxy_cookie_domain(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3632    {
3633        ngx_http_proxy_loc_conf_t *plcf = conf;
```

```
3634
3635      ngx_str_t                         *value;
3636      ngx_http_proxy_rewrite_t          *pr;
3637      ngx_http_compile_complex_value_t   ccv;
3638
3639      if (plcf->cookie_domains == NULL) {
3640          return NGX_CONF_OK;
3641      }
3642
3643      value = cf->args->elts;
3644
3645      if (cf->args->nelts == 2) {
3646
3647          if (ngx_strcmp(value[1].data, "off") == 0) {
3648              plcf->cookie_domains = NULL;
3649              return NGX_CONF_OK;
3650          }
3651
3652          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3653                             "invalid parameter \"%V\"", &value[1]);
3654          return NGX_CONF_ERROR;
3655      }
3656
3657      if (plcf->cookie_domains == NGX_CONF_UNSET_PTR) {
3658          plcf->cookie_domains = ngx_array_create(cf->pool, 1,
3659                                                  sizeof(ngx_http_proxy_rewrite_t));
3660          if (plcf->cookie_domains == NULL) {
3661              return NGX_CONF_ERROR;
3662          }
3663      }
3664
3665      pr = ngx_array_push(plcf->cookie_domains);
3666      if (pr == NULL) {
3667          return NGX_CONF_ERROR;
3668      }
3669
3670      if (value[1].data[0] == '~') {
3671          value[1].len--;
3672          value[1].data++;
3673
3674          if (ngx_http_proxy_rewrite_regex(cf, pr, &value[1], 1) != NGX_OK) {
3675              return NGX_CONF_ERROR;
3676          }
3677
3678      } else {
3679
3680          if (value[1].data[0] == '.') {
3681              value[1].len--;
3682              value[1].data++;
3683          }
3684
3685          ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3686
3687          ccv.cf = cf;
3688          ccv.value = &value[1];
3689          ccv.complex_value = &pr->pattern.complex;
3690
3691          if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3692              return NGX_CONF_ERROR;
3693          }
3694
3695          pr->handler = ngx_http_proxy_rewrite_domain_handler;
3696
3697          if (value[2].data[0] == '.') {
3698              value[2].len--;
3699              value[2].data++;
3700          }
3701      }
3702
3703      ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3704
3705      ccv.cf = cf;
3706      ccv.value = &value[2];
3707      ccv.complex_value = &pr->replacement;
3708
3709      if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
```

```
3710         return NGX_CONF_ERROR;
3711     }
3712
3713     return NGX_CONF_OK;
3714 }
3715
3716
3717 static char *
3718 ngx_http_proxy_cookie_path(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3719 {
3720     ngx_http_proxy_loc_conf_t *plcf = conf;
3721
3722     ngx_str_t                         *value;
3723     ngx_http_proxy_rewrite_t          *pr;
3724     ngx_http_compile_complex_value_t   ccv;
3725
3726     if (plcf->cookie_paths == NULL) {
3727         return NGX_CONF_OK;
3728     }
3729
3730     value = cf->args->elts;
3731
3732     if (cf->args->nelts == 2) {
3733
3734         if (ngx_strcmp(value[1].data, "off") == 0) {
3735             plcf->cookie_paths = NULL;
3736             return NGX_CONF_OK;
3737         }
3738
3739         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3740                            "invalid parameter \"%V\"", &value[1]);
3741         return NGX_CONF_ERROR;
3742     }
3743
3744     if (plcf->cookie_paths == NGX_CONF_UNSET_PTR) {
3745         plcf->cookie_paths = ngx_array_create(cf->pool, 1,
3746                                               sizeof(ngx_http_proxy_rewrite_t));
3747         if (plcf->cookie_paths == NULL) {
3748             return NGX_CONF_ERROR;
3749         }
3750     }
3751
3752     pr = ngx_array_push(plcf->cookie_paths);
3753     if (pr == NULL) {
3754         return NGX_CONF_ERROR;
3755     }
3756
3757     if (value[1].data[0] == '~') {
3758         value[1].len--;
3759         value[1].data++;
3760
3761         if (value[1].data[0] == '*') {
3762             value[1].len--;
3763             value[1].data++;
3764
3765             if (ngx_http_proxy_rewrite_regex(cf, pr, &value[1], 1) != NGX_OK) {
3766                 return NGX_CONF_ERROR;
3767             }
3768
3769         } else {
3770             if (ngx_http_proxy_rewrite_regex(cf, pr, &value[1], 0) != NGX_OK) {
3771                 return NGX_CONF_ERROR;
3772             }
3773         }
3774
3775     } else {
3776
3777         ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3778
3779         ccv.cf = cf;
3780         ccv.value = &value[1];
3781         ccv.complex_value = &pr->pattern.complex;
3782
3783         if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3784             return NGX_CONF_ERROR;
3785         }
```

```
3786
3787            pr->handler = ngx_http_proxy_rewrite_complex_handler;
3788        }
3789
3790        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3791
3792        ccv.cf = cf;
3793        ccv.value = &value[2];
3794        ccv.complex_value = &pr->replacement;
3795
3796        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3797            return NGX_CONF_ERROR;
3798        }
3799
3800        return NGX_CONF_OK;
3801    }
3802
3803
3804    static ngx_int_t
3805    ngx_http_proxy_rewrite_regex(ngx_conf_t *cf, ngx_http_proxy_rewrite_t *pr,
3806        ngx_str_t *regex, ngx_uint_t caseless)
3807    {
3808    #if (NGX_PCRE)
3809        u_char               errstr[NGX_MAX_CONF_ERRSTR];
3810        ngx_regex_compile_t  rc;
3811
3812        ngx_memzero(&rc, sizeof(ngx_regex_compile_t));
3813
3814        rc.pattern = *regex;
3815        rc.err.len = NGX_MAX_CONF_ERRSTR;
3816        rc.err.data = errstr;
3817
3818        if (caseless) {
3819            rc.options = NGX_REGEX_CASELESS;
3820        }
3821
3822        pr->pattern.regex = ngx_http_regex_compile(cf, &rc);
3823        if (pr->pattern.regex == NULL) {
3824            return NGX_ERROR;
3825        }
3826
3827        pr->handler = ngx_http_proxy_rewrite_regex_handler;
3828
3829        return NGX_OK;
3830
3831    #else
3832
3833        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
3834                           "using regex \"%V\" requires PCRE library", regex);
3835        return NGX_ERROR;
3836
3837    #endif
3838    }
3839
3840
3841    static char *
3842    ngx_http_proxy_store(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3843    {
3844        ngx_http_proxy_loc_conf_t *plcf = conf;
3845
3846        ngx_str_t                 *value;
3847        ngx_http_script_compile_t  sc;
3848
3849        if (plcf->upstream.store != NGX_CONF_UNSET) {
3850            return "is duplicate";
3851        }
3852
3853        value = cf->args->elts;
3854
3855        if (ngx_strcmp(value[1].data, "off") == 0) {
3856            plcf->upstream.store = 0;
3857            return NGX_CONF_OK;
3858        }
3859
3860    #if (NGX_HTTP_CACHE)
3861        if (plcf->upstream.cache > 0) {
```

```
3862            return "is incompatible with \"proxy_cache\"";
3863        }
3864    #endif
3865
3866        plcf->upstream.store = 1;
3867
3868        if (ngx_strcmp(value[1].data, "on") == 0) {
3869            return NGX_CONF_OK;
3870        }
3871
3872        /* include the terminating '\0' into script */
3873        value[1].len++;
3874
3875        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
3876
3877        sc.cf = cf;
3878        sc.source = &value[1];
3879        sc.lengths = &plcf->upstream.store_lengths;
3880        sc.values = &plcf->upstream.store_values;
3881        sc.variables = ngx_http_script_variables_count(&value[1]);
3882        sc.complete_lengths = 1;
3883        sc.complete_values = 1;
3884
3885        if (ngx_http_script_compile(&sc) != NGX_OK) {
3886            return NGX_CONF_ERROR;
3887        }
3888
3889        return NGX_CONF_OK;
3890    }
3891
3892
3893    #if (NGX_HTTP_CACHE)
3894
3895    static char *
3896    ngx_http_proxy_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3897    {
3898        ngx_http_proxy_loc_conf_t *plcf = conf;
3899
3900        ngx_str_t                         *value;
3901        ngx_http_complex_value_t           cv;
3902        ngx_http_compile_complex_value_t   ccv;
3903
3904        value = cf->args->elts;
3905
3906        if (plcf->upstream.cache != NGX_CONF_UNSET) {
3907            return "is duplicate";
3908        }
3909
3910        if (ngx_strcmp(value[1].data, "off") == 0) {
3911            plcf->upstream.cache = 0;
3912            return NGX_CONF_OK;
3913        }
3914
3915        if (plcf->upstream.store > 0) {
3916            return "is incompatible with \"proxy_store\"";
3917        }
3918
3919        plcf->upstream.cache = 1;
3920
3921        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3922
3923        ccv.cf = cf;
3924        ccv.value = &value[1];
3925        ccv.complex_value = &cv;
3926
3927        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3928            return NGX_CONF_ERROR;
3929        }
3930
3931        if (cv.lengths != NULL) {
3932
3933            plcf->upstream.cache_value = ngx_palloc(cf->pool,
3934                                                    sizeof(ngx_http_complex_value_t));
3935            if (plcf->upstream.cache_value == NULL) {
3936                return NGX_CONF_ERROR;
3937            }
```

```
3938
3939            *plcf->upstream.cache_value = cv;
3940
3941            return NGX_CONF_OK;
3942        }
3943
3944        plcf->upstream.cache_zone = ngx_shared_memory_add(cf, &value[1], 0,
3945                                                          &ngx_http_proxy_module);
3946        if (plcf->upstream.cache_zone == NULL) {
3947            return NGX_CONF_ERROR;
3948        }
3949
3950        return NGX_CONF_OK;
3951    }
3952
3953
3954    static char *
3955    ngx_http_proxy_cache_key(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3956    {
3957        ngx_http_proxy_loc_conf_t *plcf = conf;
3958
3959        ngx_str_t                         *value;
3960        ngx_http_compile_complex_value_t   ccv;
3961
3962        value = cf->args->elts;
3963
3964        if (plcf->cache_key.value.data) {
3965            return "is duplicate";
3966        }
3967
3968        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
3969
3970        ccv.cf = cf;
3971        ccv.value = &value[1];
3972        ccv.complex_value = &plcf->cache_key;
3973
3974        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
3975            return NGX_CONF_ERROR;
3976        }
3977
3978        return NGX_CONF_OK;
3979    }
3980
3981    #endif
3982
3983
3984    #if (NGX_HTTP_SSL)
3985
3986    static char *
3987    ngx_http_proxy_ssl_password_file(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
3988    {
3989        ngx_http_proxy_loc_conf_t *plcf = conf;
3990
3991        ngx_str_t  *value;
3992
3993        if (plcf->ssl_passwords != NGX_CONF_UNSET_PTR) {
3994            return "is duplicate";
3995        }
3996
3997        value = cf->args->elts;
3998
3999        plcf->ssl_passwords = ngx_ssl_read_password_file(cf, &value[1]);
4000
4001        if (plcf->ssl_passwords == NULL) {
4002            return NGX_CONF_ERROR;
4003        }
4004
4005        return NGX_CONF_OK;
4006    }
4007
4008    #endif
4009
4010
4011    static char *
4012    ngx_http_proxy_lowat_check(ngx_conf_t *cf, void *post, void *data)
4013    {
```

```
#if (NGX_FREEBSD)
    ssize_t *np = data;

    if ((u_long) *np >= ngx_freebsd_net_inet_tcp_sendspace) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"proxy_send_lowat\" must be less than %d "
                           "(sysctl net.inet.tcp.sendspace)",
                           ngx_freebsd_net_inet_tcp_sendspace);

        return NGX_CONF_ERROR;
    }

#elif !(NGX_HAVE_SO_SNDLOWAT)
    ssize_t *np = data;

    ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                       "\"proxy_send_lowat\" is not supported, ignored");

    *np = 0;

#endif

    return NGX_CONF_OK;
}


#if (NGX_HTTP_SSL)

static ngx_int_t
ngx_http_proxy_set_ssl(ngx_conf_t *cf, ngx_http_proxy_loc_conf_t *plcf)
{
    ngx_pool_cleanup_t  *cln;

    plcf->upstream.ssl = ngx_pcalloc(cf->pool, sizeof(ngx_ssl_t));
    if (plcf->upstream.ssl == NULL) {
        return NGX_ERROR;
    }

    plcf->upstream.ssl->log = cf->log;

    if (ngx_ssl_create(plcf->upstream.ssl, plcf->ssl_protocols, NULL)
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    cln = ngx_pool_cleanup_add(cf->pool, 0);
    if (cln == NULL) {
        return NGX_ERROR;
    }

    cln->handler = ngx_ssl_cleanup_ctx;
    cln->data = plcf->upstream.ssl;

    if (plcf->ssl_certificate.len) {

        if (plcf->ssl_certificate_key.len == 0) {
            ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                          "no \"proxy_ssl_certificate_key\" is defined "
                          "for certificate \"%V\"", &plcf->ssl_certificate);
            return NGX_ERROR;
        }

        if (ngx_ssl_certificate(cf, plcf->upstream.ssl, &plcf->ssl_certificate,
                                &plcf->ssl_certificate_key, plcf->ssl_passwords)
            != NGX_OK)
        {
            return NGX_ERROR;
        }
    }

    if (SSL_CTX_set_cipher_list(plcf->upstream.ssl->ctx,
                                (const char *) plcf->ssl_ciphers.data)
        == 0)
    {
        ngx_ssl_error(NGX_LOG_EMERG, cf->log, 0,
```

```
4090                        "SSL_CTX_set_cipher_list(\"%V\") failed",
4091                        &plcf->ssl_ciphers);
4092          return NGX_ERROR;
4093      }
4094
4095      if (plcf->upstream.ssl_verify) {
4096          if (plcf->ssl_trusted_certificate.len == 0) {
4097              ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
4098                        "no proxy_ssl_trusted_certificate for proxy_ssl_verify");
4099              return NGX_ERROR;
4100          }
4101
4102          if (ngx_ssl_trusted_certificate(cf, plcf->upstream.ssl,
4103                                          &plcf->ssl_trusted_certificate,
4104                                          plcf->ssl_verify_depth)
4105              != NGX_OK)
4106          {
4107              return NGX_ERROR;
4108          }
4109
4110          if (ngx_ssl_crl(cf, plcf->upstream.ssl, &plcf->ssl_crl) != NGX_OK) {
4111              return NGX_ERROR;
4112          }
4113      }
4114
4115      return NGX_OK;
4116  }
4117
4118  #endif
4119
4120
4121  static void
4122  ngx_http_proxy_set_vars(ngx_url_t *u, ngx_http_proxy_vars_t *v)
4123  {
4124      if (u->family != AF_UNIX) {
4125
4126          if (u->no_port || u->port == u->default_port) {
4127
4128              v->host_header = u->host;
4129
4130              if (u->default_port == 80) {
4131                  ngx_str_set(&v->port, "80");
4132
4133              } else {
4134                  ngx_str_set(&v->port, "443");
4135              }
4136
4137          } else {
4138              v->host_header.len = u->host.len + 1 + u->port_text.len;
4139              v->host_header.data = u->host.data;
4140              v->port = u->port_text;
4141          }
4142
4143          v->key_start.len += v->host_header.len;
4144
4145      } else {
4146          ngx_str_set(&v->host_header, "localhost");
4147          ngx_str_null(&v->port);
4148          v->key_start.len += sizeof("unix:") - 1 + u->host.len + 1;
4149      }
4150
4151      v->uri = u->uri;
4152  }
```

One Level Up Top Level

src/http/modules/ngx_http_random_index_module.c - nginx-1.7.10

Global variables defined

- ngx_http_random_index_commands
- ngx_http_random_index_module
- ngx_http_random_index_module_ctx

Data types defined

- ngx_http_random_index_loc_conf_t

Functions defined

- ngx_http_random_index_create_loc_conf
- ngx_http_random_index_error
- ngx_http_random_index_handler
- ngx_http_random_index_init
- ngx_http_random_index_merge_loc_conf

Macros defined

- NGX_HTTP_RANDOM_INDEX_PREALLOCATE

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_flag_t  enable;
15  } ngx_http_random_index_loc_conf_t;
16
17
18  #define NGX_HTTP_RANDOM_INDEX_PREALLOCATE  50
19
20
21  static ngx_int_t ngx_http_random_index_error(ngx_http_request_t *r,
22      ngx_dir_t *dir, ngx_str_t *name);
23  static ngx_int_t ngx_http_random_index_init(ngx_conf_t *cf);
24  static void *ngx_http_random_index_create_loc_conf(ngx_conf_t *cf);
25  static char *ngx_http_random_index_merge_loc_conf(ngx_conf_t *cf,
26      void *parent, void *child);
27
28
29  static ngx_command_t  ngx_http_random_index_commands[] = {
30
31      { ngx_string("random_index"),
```

```
      NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_random_index_loc_conf_t, enable),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_random_index_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_random_index_init,            /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_random_index_create_loc_conf, /* create location configuration */
    ngx_http_random_index_merge_loc_conf   /* merge location configuration */
};


ngx_module_t  ngx_http_random_index_module = {
    NGX_MODULE_V1,
    &ngx_http_random_index_module_ctx,     /* module context */
    ngx_http_random_index_commands,        /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_random_index_handler(ngx_http_request_t *r)
{
    u_char                          *last, *filename;
    size_t                           len, allocated, root;
    ngx_err_t                        err;
    ngx_int_t                        rc;
    ngx_str_t                        path, uri, *name;
    ngx_dir_t                        dir;
    ngx_uint_t                       n, level;
    ngx_array_t                      names;
    ngx_http_random_index_loc_conf_t  *rlcf;

    if (r->uri.data[r->uri.len - 1] != '/') {
        return NGX_DECLINED;
    }

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD|NGX_HTTP_POST))) {
        return NGX_DECLINED;
    }

    rlcf = ngx_http_get_module_loc_conf(r, ngx_http_random_index_module);

    if (!rlcf->enable) {
        return NGX_DECLINED;
    }

#if (NGX_HAVE_D_TYPE)
    len = NGX_DIR_MASK_LEN;
#else
    len = NGX_HTTP_RANDOM_INDEX_PREALLOCATE;
#endif

    last = ngx_http_map_uri_to_path(r, &path, &root, len);
    if (last == NULL) {
```

```
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    allocated = path.len;

    path.len = last - path.data - 1;
    path.data[path.len] = '\0';

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http random index: \"%s\"", path.data);

    if (ngx_open_dir(&path, &dir) == NGX_ERROR) {
        err = ngx_errno;

        if (err == NGX_ENOENT
            || err == NGX_ENOTDIR
            || err == NGX_ENAMETOOLONG)
        {
            level = NGX_LOG_ERR;
            rc = NGX_HTTP_NOT_FOUND;

        } else if (err == NGX_EACCES) {
            level = NGX_LOG_ERR;
            rc = NGX_HTTP_FORBIDDEN;

        } else {
            level = NGX_LOG_CRIT;
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        ngx_log_error(level, r->connection->log, err,
                      ngx_open_dir_n " \"%s\" failed", path.data);

        return rc;
    }

    if (ngx_array_init(&names, r->pool, 32, sizeof(ngx_str_t)) != NGX_OK) {
        return ngx_http_random_index_error(r, &dir, &path);
    }

    filename = path.data;
    filename[path.len] = '/';

    for ( ;; ) {
        ngx_set_errno(0);

        if (ngx_read_dir(&dir) == NGX_ERROR) {
            err = ngx_errno;

            if (err != NGX_ENOMOREFILES) {
                ngx_log_error(NGX_LOG_CRIT, r->connection->log, err,
                              ngx_read_dir_n " \"%V\" failed", &path);
                return ngx_http_random_index_error(r, &dir, &path);
            }

            break;
        }

        ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http random index file: \"%s\"", ngx_de_name(&dir));

        if (ngx_de_name(&dir)[0] == '.') {
            continue;
        }

        len = ngx_de_namelen(&dir);

        if (dir.type == 0 || ngx_de_is_link(&dir)) {

            /* 1 byte for '/' and 1 byte for terminating '\0' */

            if (path.len + 1 + len + 1 > allocated) {
                allocated = path.len + 1 + len + 1
                                     + NGX_HTTP_RANDOM_INDEX_PREALLOCATE;

                filename = ngx_pnalloc(r->pool, allocated);
```

```
                if (filename == NULL) {
                    return ngx_http_random_index_error(r, &dir, &path);
                }

                last = ngx_cpystrn(filename, path.data, path.len + 1);
                *last++ = '/';
            }

            ngx_cpystrn(last, ngx_de_name(&dir), len + 1);

            if (ngx_de_info(filename, &dir) == NGX_FILE_ERROR) {
                err = ngx_errno;

                if (err != NGX_ENOENT) {
                    ngx_log_error(NGX_LOG_CRIT, r->connection->log, err,
                                  ngx_de_info_n " \"%s\" failed", filename);
                    return ngx_http_random_index_error(r, &dir, &path);
                }

                if (ngx_de_link_info(filename, &dir) == NGX_FILE_ERROR) {
                    ngx_log_error(NGX_LOG_CRIT, r->connection->log, ngx_errno,
                                  ngx_de_link_info_n " \"%s\" failed",
                                  filename);
                    return ngx_http_random_index_error(r, &dir, &path);
                }
            }
        }

        if (!ngx_de_is_file(&dir)) {
            continue;
        }

        name = ngx_array_push(&names);
        if (name == NULL) {
            return ngx_http_random_index_error(r, &dir, &path);
        }

        name->len = len;

        name->data = ngx_pnalloc(r->pool, len);
        if (name->data == NULL) {
            return ngx_http_random_index_error(r, &dir, &path);
        }

        ngx_memcpy(name->data, ngx_de_name(&dir), len);
    }

    if (ngx_close_dir(&dir) == NGX_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, ngx_errno,
                      ngx_close_dir_n " \"%s\" failed", &path);
    }

    n = names.nelts;

    if (n == 0) {
        return NGX_DECLINED;
    }

    name = names.elts;

    n = (ngx_uint_t) (((uint64_t) ngx_random() * n) / 0x80000000);

    uri.len = r->uri.len + name[n].len;

    uri.data = ngx_pnalloc(r->pool, uri.len);
    if (uri.data == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    last = ngx_copy(uri.data, r->uri.data, r->uri.len);
    ngx_memcpy(last, name[n].data, name[n].len);

    return ngx_http_internal_redirect(r, &uri, &r->args);
}
```

```
static ngx_int_t
ngx_http_random_index_error(ngx_http_request_t *r, ngx_dir_t *dir,
    ngx_str_t *name)
{
    if (ngx_close_dir(dir) == NGX_ERROR) {
        ngx_log_error(NGX_LOG_ALERT, r->connection->log, ngx_errno,
                      ngx_close_dir_n " \"%V\" failed", name);
    }

    return NGX_HTTP_INTERNAL_SERVER_ERROR;
}


static void *
ngx_http_random_index_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_random_index_loc_conf_t  *conf;

    conf = ngx_palloc(cf->pool, sizeof(ngx_http_random_index_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->enable = NGX_CONF_UNSET;

    return conf;
}


static char *
ngx_http_random_index_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_random_index_loc_conf_t *prev = parent;
    ngx_http_random_index_loc_conf_t *conf = child;

    ngx_conf_merge_value(conf->enable, prev->enable, 0);

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_random_index_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt        *h;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    *h = ngx_http_random_index_handler;

    return NGX_OK;
}
```

One Level Up Top Level

src/http/modules/ngx_http_range_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_body_filter
- ngx_http_next_header_filter
- ngx_http_range_body_filter_module
- ngx_http_range_body_filter_module_ctx
- ngx_http_range_header_filter_module
- ngx_http_range_header_filter_module_ctx

Data types defined

- ngx_http_range_filter_ctx_t
- ngx_http_range_t

Functions defined

- ngx_http_range_body_filter
- ngx_http_range_body_filter_init
- ngx_http_range_header_filter
- ngx_http_range_header_filter_init
- ngx_http_range_multipart_body
- ngx_http_range_multipart_header
- ngx_http_range_not_satisfiable
- ngx_http_range_parse
- ngx_http_range_singlepart_body
- ngx_http_range_singlepart_header
- ngx_http_range_test_overlapped

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  /*
14   * the single part format:
```

```
 *
 * "HTTP/1.0 206 Partial Content" CRLF
 * ... header ...
 * "Content-Type: image/jpeg" CRLF
 * "Content-Length: SIZE" CRLF
 * "Content-Range: bytes START-END/SIZE" CRLF
 * CRLF
 * ... data ...
 *
 *
 * the multipart format:
 *
 * "HTTP/1.0 206 Partial Content" CRLF
 * ... header ...
 * "Content-Type: multipart/byteranges; boundary=0123456789" CRLF
 * CRLF
 * CRLF
 * "--0123456789" CRLF
 * "Content-Type: image/jpeg" CRLF
 * "Content-Range: bytes START0-END0/SIZE" CRLF
 * CRLF
 * ... data ...
 * CRLF
 * "--0123456789" CRLF
 * "Content-Type: image/jpeg" CRLF
 * "Content-Range: bytes START1-END1/SIZE" CRLF
 * CRLF
 * ... data ...
 * CRLF
 * "--0123456789--" CRLF
 */


typedef struct {
    off_t        start;
    off_t        end;
    ngx_str_t    content_range;
} ngx_http_range_t;


typedef struct {
    off_t        offset;
    ngx_str_t    boundary_header;
    ngx_array_t  ranges;
} ngx_http_range_filter_ctx_t;


static ngx_int_t ngx_http_range_parse(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx, ngx_uint_t ranges);
static ngx_int_t ngx_http_range_singlepart_header(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx);
static ngx_int_t ngx_http_range_multipart_header(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx);
static ngx_int_t ngx_http_range_not_satisfiable(ngx_http_request_t *r);
static ngx_int_t ngx_http_range_test_overlapped(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx, ngx_chain_t *in);
static ngx_int_t ngx_http_range_singlepart_body(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx, ngx_chain_t *in);
static ngx_int_t ngx_http_range_multipart_body(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx, ngx_chain_t *in);

static ngx_int_t ngx_http_range_header_filter_init(ngx_conf_t *cf);
static ngx_int_t ngx_http_range_body_filter_init(ngx_conf_t *cf);


static ngx_http_module_t  ngx_http_range_header_filter_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_range_header_filter_init,     /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
```

```
    NULL,                                  /* merge location configuration */
};


ngx_module_t  ngx_http_range_header_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_range_header_filter_module_ctx, /* module context */
    NULL,                                  /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
     NULL,                                  /* init master */
     NULL,                                  /* init module */
     NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
};


static ngx_http_module_t  ngx_http_range_body_filter_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_range_body_filter_init,       /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL,                                  /* merge location configuration */
};


ngx_module_t  ngx_http_range_body_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_range_body_filter_module_ctx, /* module context */
    NULL,                                  /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static ngx_int_t
ngx_http_range_header_filter(ngx_http_request_t *r)
{
    time_t                       if_range_time;
    ngx_str_t                   *if_range, *etag;
    ngx_uint_t                   ranges;
    ngx_http_core_loc_conf_t    *clcf;
    ngx_http_range_filter_ctx_t  *ctx;

    if (r->http_version < NGX_HTTP_VERSION_10
        || r->headers_out.status != NGX_HTTP_OK
        || r != r->main
        || r->headers_out.content_length_n == -1
        || !r->allow_ranges)
    {
        return ngx_http_next_header_filter(r);
    }

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    if (clcf->max_ranges == 0) {
```

```
        return ngx_http_next_header_filter(r);
    }

    if (r->headers_in.range == NULL
        || r->headers_in.range->value.len < 7
        || ngx_strncasecmp(r->headers_in.range->value.data,
                           (u_char *) "bytes=", 6)
           != 0)
    {
        goto next_filter;
    }

    if (r->headers_in.if_range) {

        if_range = &r->headers_in.if_range->value;

        if (if_range->len >= 2 && if_range->data[if_range->len - 1] == '"') {

            if (r->headers_out.etag == NULL) {
                goto next_filter;
            }

            etag = &r->headers_out.etag->value;

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http ir:%V etag:%V", if_range, etag);

            if (if_range->len != etag->len
                || ngx_strncmp(if_range->data, etag->data, etag->len) != 0)
            {
                goto next_filter;
            }

            goto parse;
        }

        if (r->headers_out.last_modified_time == (time_t) -1) {
            goto next_filter;
        }

        if_range_time = ngx_http_parse_time(if_range->data, if_range->len);

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http ir:%d lm:%d",
                       if_range_time, r->headers_out.last_modified_time);

        if (if_range_time != r->headers_out.last_modified_time) {
            goto next_filter;
        }
    }

parse:

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_range_filter_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    if (ngx_array_init(&ctx->ranges, r->pool, 1, sizeof(ngx_http_range_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    ranges = r->single_range ? 1 : clcf->max_ranges;

    switch (ngx_http_range_parse(r, ctx, ranges)) {

    case NGX_OK:
        ngx_http_set_ctx(r, ctx, ngx_http_range_body_filter_module);

        r->headers_out.status = NGX_HTTP_PARTIAL_CONTENT;
        r->headers_out.status_line.len = 0;

        if (ctx->ranges.nelts == 1) {
            return ngx_http_range_singlepart_header(r, ctx);
```

```
        }

        return ngx_http_range_multipart_header(r, ctx);

    case NGX_HTTP_RANGE_NOT_SATISFIABLE:
        return ngx_http_range_not_satisfiable(r);

    case NGX_ERROR:
        return NGX_ERROR;

    default: /* NGX_DECLINED */
        break;
    }

next_filter:

    r->headers_out.accept_ranges = ngx_list_push(&r->headers_out.headers);
    if (r->headers_out.accept_ranges == NULL) {
        return NGX_ERROR;
    }

    r->headers_out.accept_ranges->hash = 1;
    ngx_str_set(&r->headers_out.accept_ranges->key, "Accept-Ranges");
    ngx_str_set(&r->headers_out.accept_ranges->value, "bytes");

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_range_parse(ngx_http_request_t *r, ngx_http_range_filter_ctx_t *ctx,
    ngx_uint_t ranges)
{
    u_char            *p;
    off_t              start, end, size, content_length;
    ngx_uint_t         suffix;
    ngx_http_range_t  *range;

    p = r->headers_in.range->value.data + 6;
    size = 0;
    content_length = r->headers_out.content_length_n;

    for ( ;; ) {
        start = 0;
        end = 0;
        suffix = 0;

        while (*p == ' ') { p++; }

        if (*p != '-') {
            if (*p < '0' || *p > '9') {
                return NGX_HTTP_RANGE_NOT_SATISFIABLE;
            }

            while (*p >= '0' && *p <= '9') {
                start = start * 10 + *p++ - '0';
            }

            while (*p == ' ') { p++; }

            if (*p++ != '-') {
                return NGX_HTTP_RANGE_NOT_SATISFIABLE;
            }

            while (*p == ' ') { p++; }

            if (*p == ',' || *p == '\0') {
                end = content_length;
                goto found;
            }

        } else {
            suffix = 1;
            p++;
        }

```

```
        if (*p < '0' || *p > '9') {
            return NGX_HTTP_RANGE_NOT_SATISFIABLE;
        }

        while (*p >= '0' && *p <= '9') {
            end = end * 10 + *p++ - '0';
        }

        while (*p == ' ') { p++; }

        if (*p != ',' && *p != '\0') {
            return NGX_HTTP_RANGE_NOT_SATISFIABLE;
        }

        if (suffix) {
            start = content_length - end;
            end = content_length - 1;
        }

        if (end >= content_length) {
            end = content_length;

        } else {
            end++;
        }

    found:

        if (start < end) {
            range = ngx_array_push(&ctx->ranges);
            if (range == NULL) {
                return NGX_ERROR;
            }

            range->start = start;
            range->end = end;

            size += end - start;

            if (ranges-- == 0) {
                return NGX_DECLINED;
            }
        }

        if (*p++ != ',') {
            break;
        }
    }

    if (ctx->ranges.nelts == 0) {
        return NGX_HTTP_RANGE_NOT_SATISFIABLE;
    }

    if (size > content_length) {
        return NGX_DECLINED;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_range_singlepart_header(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx)
{
    ngx_table_elt_t    *content_range;
    ngx_http_range_t  *range;

    content_range = ngx_list_push(&r->headers_out.headers);
    if (content_range == NULL) {
        return NGX_ERROR;
    }

    r->headers_out.content_range = content_range;

    content_range->hash = 1;
```

```
    ngx_str_set(&content_range->key, "Content-Range");

    content_range->value.data = ngx_pnalloc(r->pool,
                                    sizeof("bytes -/") - 1 + 3 * NGX_OFF_T_LEN);
    if (content_range->value.data == NULL) {
        return NGX_ERROR;
    }

    /* "Content-Range: bytes SSSS-EEEE/TTTT" header */

    range = ctx->ranges.elts;

    content_range->value.len = ngx_sprintf(content_range->value.data,
                                           "bytes %O-%O/%O",
                                           range->start, range->end - 1,
                                           r->headers_out.content_length_n)
                               - content_range->value.data;

    r->headers_out.content_length_n = range->end - range->start;

    if (r->headers_out.content_length) {
        r->headers_out.content_length->hash = 0;
        r->headers_out.content_length = NULL;
    }

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_range_multipart_header(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx)
{
    size_t              len;
    ngx_uint_t          i;
    ngx_http_range_t   *range;
    ngx_atomic_uint_t   boundary;

    len = sizeof(CRLF "--") - 1 + NGX_ATOMIC_T_LEN
          + sizeof(CRLF "Content-Type: ") - 1
          + r->headers_out.content_type.len
          + sizeof(CRLF "Content-Range: bytes ") - 1;

    if (r->headers_out.content_type_len == r->headers_out.content_type.len
        && r->headers_out.charset.len)
    {
        len += sizeof("; charset=") - 1 + r->headers_out.charset.len;
    }

    ctx->boundary_header.data = ngx_pnalloc(r->pool, len);
    if (ctx->boundary_header.data == NULL) {
        return NGX_ERROR;
    }

    boundary = ngx_next_temp_number(0);

    /*
     * The boundary header of the range:
     * CRLF
     * "--0123456789" CRLF
     * "Content-Type: image/jpeg" CRLF
     * "Content-Range: bytes "
     */

    if (r->headers_out.content_type_len == r->headers_out.content_type.len
        && r->headers_out.charset.len)
    {
        ctx->boundary_header.len = ngx_sprintf(ctx->boundary_header.data,
                                               CRLF "--%0muA" CRLF
                                               "Content-Type: %V; charset=%V" CRLF
                                               "Content-Range: bytes ",
                                               boundary,
                                               &r->headers_out.content_type,
                                               &r->headers_out.charset)
                                   - ctx->boundary_header.data;

```

```
    } else if (r->headers_out.content_type.len) {
        ctx->boundary_header.len = ngx_sprintf(ctx->boundary_header.data,
                                               CRLF "--%0muA" CRLF
                                               "Content-Type: %V" CRLF
                                               "Content-Range: bytes ",
                                               boundary,
                                               &r->headers_out.content_type)
                                   - ctx->boundary_header.data;

    } else {
        ctx->boundary_header.len = ngx_sprintf(ctx->boundary_header.data,
                                               CRLF "--%0muA" CRLF
                                               "Content-Range: bytes ",
                                               boundary)
                                   - ctx->boundary_header.data;
    }

    r->headers_out.content_type.data =
        ngx_pnalloc(r->pool,
                    sizeof("Content-Type: multipart/byteranges; boundary=") - 1
                    + NGX_ATOMIC_T_LEN);

    if (r->headers_out.content_type.data == NULL) {
        return NGX_ERROR;
    }

    r->headers_out.content_type_lowcase = NULL;

    /* "Content-Type: multipart/byteranges; boundary=0123456789" */

    r->headers_out.content_type.len =
                           ngx_sprintf(r->headers_out.content_type.data,
                                       "multipart/byteranges; boundary=%0muA",
                                       boundary)
                           - r->headers_out.content_type.data;

    r->headers_out.content_type_len = r->headers_out.content_type.len;

    r->headers_out.charset.len = 0;

    /* the size of the last boundary CRLF "--0123456789--" CRLF */

    len = sizeof(CRLF "--") - 1 + NGX_ATOMIC_T_LEN + sizeof("--" CRLF) - 1;

    range = ctx->ranges.elts;
    for (i = 0; i < ctx->ranges.nelts; i++) {

        /* the size of the range: "SSSS-EEEE/TTTT" CRLF CRLF */

        range[i].content_range.data =
                               ngx_pnalloc(r->pool, 3 * NGX_OFF_T_LEN + 2 + 4);

        if (range[i].content_range.data == NULL) {
            return NGX_ERROR;
        }

        range[i].content_range.len = ngx_sprintf(range[i].content_range.data,
                                                 "%O-%O/%O" CRLF CRLF,
                                                 range[i].start, range[i].end - 1,
                                                 r->headers_out.content_length_n)
                                     - range[i].content_range.data;

        len += ctx->boundary_header.len + range[i].content_range.len
                                    + (size_t) (range[i].end - range[i].start);
    }

    r->headers_out.content_length_n = len;

    if (r->headers_out.content_length) {
        r->headers_out.content_length->hash = 0;
        r->headers_out.content_length = NULL;
    }

    return ngx_http_next_header_filter(r);
}
```

```
547
548 static ngx_int_t
549 ngx_http_range_not_satisfiable(ngx_http_request_t *r)
550 {
551     ngx_table_elt_t  *content_range;
552
553     r->headers_out.status = NGX_HTTP_RANGE_NOT_SATISFIABLE;
554
555     content_range = ngx_list_push(&r->headers_out.headers);
556     if (content_range == NULL) {
557         return NGX_ERROR;
558     }
559
560     r->headers_out.content_range = content_range;
561
562     content_range->hash = 1;
563     ngx_str_set(&content_range->key, "Content-Range");
564
565     content_range->value.data = ngx_pnalloc(r->pool,
566                                             sizeof("bytes */") - 1 + NGX_OFF_T_LEN);
567     if (content_range->value.data == NULL) {
568         return NGX_ERROR;
569     }
570
571     content_range->value.len = ngx_sprintf(content_range->value.data,
572                                            "bytes */%O",
573                                            r->headers_out.content_length_n)
574                                - content_range->value.data;
575
576     ngx_http_clear_content_length(r);
577
578     return NGX_HTTP_RANGE_NOT_SATISFIABLE;
579 }
580
581
582 static ngx_int_t
583 ngx_http_range_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
584 {
585     ngx_http_range_filter_ctx_t  *ctx;
586
587     if (in == NULL) {
588         return ngx_http_next_body_filter(r, in);
589     }
590
591     ctx = ngx_http_get_module_ctx(r, ngx_http_range_body_filter_module);
592
593     if (ctx == NULL) {
594         return ngx_http_next_body_filter(r, in);
595     }
596
597     if (ctx->ranges.nelts == 1) {
598         return ngx_http_range_singlepart_body(r, ctx, in);
599     }
600
601     /*
602      * multipart ranges are supported only if whole body is in a single buffer
603      */
604
605     if (ngx_buf_special(in->buf)) {
606         return ngx_http_next_body_filter(r, in);
607     }
608
609     if (ngx_http_range_test_overlapped(r, ctx, in) != NGX_OK) {
610         return NGX_ERROR;
611     }
612
613     return ngx_http_range_multipart_body(r, ctx, in);
614 }
615
616
617 static ngx_int_t
618 ngx_http_range_test_overlapped(ngx_http_request_t *r,
619     ngx_http_range_filter_ctx_t *ctx, ngx_chain_t *in)
620 {
621     off_t              start, last;
622     ngx_buf_t         *buf;
```

```
    ngx_uint_t         i;
    ngx_http_range_t  *range;

    if (ctx->offset) {
        goto overlapped;
    }

    buf = in->buf;

    if (!buf->last_buf) {
        start = ctx->offset;
        last = ctx->offset + ngx_buf_size(buf);

        range = ctx->ranges.elts;
        for (i = 0; i < ctx->ranges.nelts; i++) {
            if (start > range[i].start || last < range[i].end) {
                goto overlapped;
            }
        }
    }

    ctx->offset = ngx_buf_size(buf);

    return NGX_OK;

overlapped:

    ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
                  "range in overlapped buffers");

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_range_singlepart_body(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx, ngx_chain_t *in)
{
    off_t              start, last;
    ngx_buf_t         *buf;
    ngx_chain_t       *out, *cl, **ll;
    ngx_http_range_t  *range;

    out = NULL;
    ll = &out;
    range = ctx->ranges.elts;

    for (cl = in; cl; cl = cl->next) {

        buf = cl->buf;

        start = ctx->offset;
        last = ctx->offset + ngx_buf_size(buf);

        ctx->offset = last;

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http range body buf: %O-%O", start, last);

        if (ngx_buf_special(buf)) {
            *ll = cl;
            ll = &cl->next;
            continue;
        }

        if (range->end <= start || range->start >= last) {

            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http range body skip");

            if (buf->in_file) {
                buf->file_pos = buf->file_last;
            }

            buf->pos = buf->last;
            buf->sync = 1;
```

```
            continue;
        }

        if (range->start > start) {

            if (buf->in_file) {
                buf->file_pos += range->start - start;
            }

            if (ngx_buf_in_memory(buf)) {
                buf->pos += (size_t) (range->start - start);
            }
        }

        if (range->end <= last) {

            if (buf->in_file) {
                buf->file_last -= last - range->end;
            }

            if (ngx_buf_in_memory(buf)) {
                buf->last -= (size_t) (last - range->end);
            }

            buf->last_buf = 1;
            *ll = cl;
            cl->next = NULL;

            break;
        }

        *ll = cl;
        ll = &cl->next;
    }

    if (out == NULL) {
        return NGX_OK;
    }

    return ngx_http_next_body_filter(r, out);
}


static ngx_int_t
ngx_http_range_multipart_body(ngx_http_request_t *r,
    ngx_http_range_filter_ctx_t *ctx, ngx_chain_t *in)
{
    ngx_buf_t         *b, *buf;
    ngx_uint_t         i;
    ngx_chain_t       *out, *hcl, *rcl, *dcl, **ll;
    ngx_http_range_t  *range;

    ll = &out;
    buf = in->buf;
    range = ctx->ranges.elts;

    for (i = 0; i < ctx->ranges.nelts; i++) {

        /*
         * The boundary header of the range:
         * CRLF
         * "--0123456789" CRLF
         * "Content-Type: image/jpeg" CRLF
         * "Content-Range: bytes "
         */

        b = ngx_calloc_buf(r->pool);
        if (b == NULL) {
            return NGX_ERROR;
        }

        b->memory = 1;
        b->pos = ctx->boundary_header.data;
        b->last = ctx->boundary_header.data + ctx->boundary_header.len;

```

```
        hcl = ngx_alloc_chain_link(r->pool);
        if (hcl == NULL) {
            return NGX_ERROR;
        }

        hcl->buf = b;


        /* "SSSS-EEEE/TTTT" CRLF CRLF */

        b = ngx_calloc_buf(r->pool);
        if (b == NULL) {
            return NGX_ERROR;
        }

        b->temporary = 1;
        b->pos = range[i].content_range.data;
        b->last = range[i].content_range.data + range[i].content_range.len;

        rcl = ngx_alloc_chain_link(r->pool);
        if (rcl == NULL) {
            return NGX_ERROR;
        }

        rcl->buf = b;


        /* the range data */

        b = ngx_calloc_buf(r->pool);
        if (b == NULL) {
            return NGX_ERROR;
        }

        b->in_file = buf->in_file;
        b->temporary = buf->temporary;
        b->memory = buf->memory;
        b->mmap = buf->mmap;
        b->file = buf->file;

        if (buf->in_file) {
            b->file_pos = buf->file_pos + range[i].start;
            b->file_last = buf->file_pos + range[i].end;
        }

        if (ngx_buf_in_memory(buf)) {
            b->pos = buf->pos + (size_t) range[i].start;
            b->last = buf->pos + (size_t) range[i].end;
        }

        dcl = ngx_alloc_chain_link(r->pool);
        if (dcl == NULL) {
            return NGX_ERROR;
        }

        dcl->buf = b;

        *ll = hcl;
        hcl->next = rcl;
        rcl->next = dcl;
        ll = &dcl->next;
    }

    /* the last boundary CRLF "--0123456789--" CRLF  */

    b = ngx_calloc_buf(r->pool);
    if (b == NULL) {
        return NGX_ERROR;
    }

    b->temporary = 1;
    b->last_buf = 1;

    b->pos = ngx_pnalloc(r->pool, sizeof(CRLF "--") - 1 + NGX_ATOMIC_T_LEN
                                  + sizeof("--" CRLF) - 1);
    if (b->pos == NULL) {
```

```
851         return NGX_ERROR;
852     }
853 
854     b->last = ngx_cpymem(b->pos, ctx->boundary_header.data,
855                          sizeof(CRLF "--") - 1 + NGX_ATOMIC_T_LEN);
856     *b->last++ = '-'; *b->last++ = '-';
857     *b->last++ = CR; *b->last++ = LF;
858 
859     hcl = ngx_alloc_chain_link(r->pool);
860     if (hcl == NULL) {
861         return NGX_ERROR;
862     }
863 
864     hcl->buf = b;
865     hcl->next = NULL;
866 
867     *ll = hcl;
868 
869     return ngx_http_next_body_filter(r, out);
870 }
871 
872 
873 static ngx_int_t
874 ngx_http_range_header_filter_init(ngx_conf_t *cf)
875 {
876     ngx_http_next_header_filter = ngx_http_top_header_filter;
877     ngx_http_top_header_filter = ngx_http_range_header_filter;
878 
879     return NGX_OK;
880 }
881 
882 
883 static ngx_int_t
884 ngx_http_range_body_filter_init(ngx_conf_t *cf)
885 {
886     ngx_http_next_body_filter = ngx_http_top_body_filter;
887     ngx_http_top_body_filter = ngx_http_range_body_filter;
888 
889     return NGX_OK;
890 }
```

One Level Up Top Level

src/http/modules/ngx_http_realip_module.c - nginx-1.7.10

Global variables defined

- ngx_http_realip_commands
- ngx_http_realip_module
- ngx_http_realip_module_ctx

Data types defined

- ngx_http_realip_ctx_t
- ngx_http_realip_loc_conf_t

Functions defined

- ngx_http_realip
- ngx_http_realip_cleanup
- ngx_http_realip_create_loc_conf
- ngx_http_realip_from
- ngx_http_realip_handler
- ngx_http_realip_init
- ngx_http_realip_merge_loc_conf
- ngx_http_realip_set_addr

Macros defined

- NGX_HTTP_REALIP_HEADER
- NGX_HTTP_REALIP_PROXY
- NGX_HTTP_REALIP_XFWD
- NGX_HTTP_REALIP_XREALIP

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define NGX_HTTP_REALIP_XREALIP  0
14  #define NGX_HTTP_REALIP_XFWD     1
15  #define NGX_HTTP_REALIP_HEADER   2
```

```
#define NGX_HTTP_REALIP_PROXY    3


typedef struct {
    ngx_array_t       *from;     /* array of ngx_cidr_t */
    ngx_uint_t         type;
    ngx_uint_t         hash;
    ngx_str_t          header;
    ngx_flag_t         recursive;
} ngx_http_realip_loc_conf_t;


typedef struct {
    ngx_connection_t  *connection;
    struct sockaddr   *sockaddr;
    socklen_t          socklen;
    ngx_str_t          addr_text;
} ngx_http_realip_ctx_t;


static ngx_int_t ngx_http_realip_handler(ngx_http_request_t *r);
static ngx_int_t ngx_http_realip_set_addr(ngx_http_request_t *r,
    ngx_addr_t *addr);
static void ngx_http_realip_cleanup(void *data);
static char *ngx_http_realip_from(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_realip(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static void *ngx_http_realip_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_realip_merge_loc_conf(ngx_conf_t *cf,
    void *parent, void *child);
static ngx_int_t ngx_http_realip_init(ngx_conf_t *cf);


static ngx_command_t  ngx_http_realip_commands[] = {

    { ngx_string("set_real_ip_from"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_realip_from,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("real_ip_header"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_realip,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("real_ip_recursive"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_realip_loc_conf_t, recursive),
      NULL },

      ngx_null_command
};



static ngx_http_module_t  ngx_http_realip_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_realip_init,                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_realip_create_loc_conf,       /* create location configuration */
    ngx_http_realip_merge_loc_conf         /* merge location configuration */
};


```

```
ngx_module_t  ngx_http_realip_module = {
    NGX_MODULE_V1,
    &ngx_http_realip_module_ctx,           /* module context */
    ngx_http_realip_commands,              /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_realip_handler(ngx_http_request_t *r)
{
    u_char                     *p;
    size_t                      len;
    ngx_str_t                  *value;
    ngx_uint_t                  i, hash;
    ngx_addr_t                  addr;
    ngx_array_t                *xfwd;
    ngx_list_part_t            *part;
    ngx_table_elt_t            *header;
    ngx_connection_t           *c;
    ngx_http_realip_ctx_t       *ctx;
    ngx_http_realip_loc_conf_t  *rlcf;

    ctx = ngx_http_get_module_ctx(r, ngx_http_realip_module);

    if (ctx) {
        return NGX_DECLINED;
    }

    rlcf = ngx_http_get_module_loc_conf(r, ngx_http_realip_module);

    if (rlcf->from == NULL) {
        return NGX_DECLINED;
    }

    switch (rlcf->type) {

    case NGX_HTTP_REALIP_XREALIP:

        if (r->headers_in.x_real_ip == NULL) {
            return NGX_DECLINED;
        }

        value = &r->headers_in.x_real_ip->value;
        xfwd = NULL;

        break;

    case NGX_HTTP_REALIP_XFWD:

        xfwd = &r->headers_in.x_forwarded_for;

        if (xfwd->elts == NULL) {
            return NGX_DECLINED;
        }

        value = NULL;

        break;

    case NGX_HTTP_REALIP_PROXY:

        value = &r->connection->proxy_protocol_addr;

        if (value->len == 0) {
            return NGX_DECLINED;
        }

```

```
        xfwd = NULL;

        break;

    default: /* NGX_HTTP_REALIP_HEADER */

        part = &r->headers_in.headers.part;
        header = part->elts;

        hash = rlcf->hash;
        len = rlcf->header.len;
        p = rlcf->header.data;

        for (i = 0; /* void */ ; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            if (hash == header[i].hash
                && len == header[i].key.len
                && ngx_strncmp(p, header[i].lowcase_key, len) == 0)
            {
                value = &header[i].value;
                xfwd = NULL;

                goto found;
            }
        }

        return NGX_DECLINED;
    }

found:

    c = r->connection;

    addr.sockaddr = c->sockaddr;
    addr.socklen = c->socklen;
    /* addr.name = c->addr_text; */

    if (ngx_http_get_forwarded_addr(r, &addr, xfwd, value, rlcf->from,
                                    rlcf->recursive)
        != NGX_DECLINED)
    {
        return ngx_http_realip_set_addr(r, &addr);
    }

    return NGX_DECLINED;
}


static ngx_int_t
ngx_http_realip_set_addr(ngx_http_request_t *r, ngx_addr_t *addr)
{
    size_t                  len;
    u_char                 *p;
    u_char                  text[NGX_SOCKADDR_STRLEN];
    ngx_connection_t       *c;
    ngx_pool_cleanup_t     *cln;
    ngx_http_realip_ctx_t  *ctx;

    cln = ngx_pool_cleanup_add(r->pool, sizeof(ngx_http_realip_ctx_t));
    if (cln == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ctx = cln->data;
    ngx_http_set_ctx(r, ctx, ngx_http_realip_module);

```

```
    c = r->connection;

    len = ngx_sock_ntop(addr->sockaddr, addr->socklen, text,
                        NGX_SOCKADDR_STRLEN, 0);
    if (len == 0) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    p = ngx_pnalloc(c->pool, len);
    if (p == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ngx_memcpy(p, text, len);

    cln->handler = ngx_http_realip_cleanup;

    ctx->connection = c;
    ctx->sockaddr = c->sockaddr;
    ctx->socklen = c->socklen;
    ctx->addr_text = c->addr_text;

    c->sockaddr = addr->sockaddr;
    c->socklen = addr->socklen;
    c->addr_text.len = len;
    c->addr_text.data = p;

    return NGX_DECLINED;
}


static void
ngx_http_realip_cleanup(void *data)
{
    ngx_http_realip_ctx_t *ctx = data;

    ngx_connection_t  *c;

    c = ctx->connection;

    c->sockaddr = ctx->sockaddr;
    c->socklen = ctx->socklen;
    c->addr_text = ctx->addr_text;
}


static char *
ngx_http_realip_from(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_realip_loc_conf_t *rlcf = conf;

    ngx_int_t                   rc;
    ngx_str_t                  *value;
    ngx_cidr_t                 *cidr;

    value = cf->args->elts;

    if (rlcf->from == NULL) {
        rlcf->from = ngx_array_create(cf->pool, 2,
                                      sizeof(ngx_cidr_t));
        if (rlcf->from == NULL) {
            return NGX_CONF_ERROR;
        }
    }

    cidr = ngx_array_push(rlcf->from);
    if (cidr == NULL) {
        return NGX_CONF_ERROR;
    }

#if (NGX_HAVE_UNIX_DOMAIN)

    if (ngx_strcmp(value[1].data, "unix:") == 0) {
         cidr->family = AF_UNIX;
         return NGX_CONF_OK;
    }
```

```
#endif

    rc = ngx_ptocidr(&value[1], cidr);

    if (rc == NGX_ERROR) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "invalid parameter \"%V\"",
                           &value[1]);
        return NGX_CONF_ERROR;
    }

    if (rc == NGX_DONE) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "low address bits of %V are meaningless", &value[1]);
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_realip(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_realip_loc_conf_t *rlcf = conf;

    ngx_str_t  *value;

    value = cf->args->elts;

    if (ngx_strcmp(value[1].data, "X-Real-IP") == 0) {
        rlcf->type = NGX_HTTP_REALIP_XREALIP;
        return NGX_CONF_OK;
    }

    if (ngx_strcmp(value[1].data, "X-Forwarded-For") == 0) {
        rlcf->type = NGX_HTTP_REALIP_XFWD;
        return NGX_CONF_OK;
    }

    if (ngx_strcmp(value[1].data, "proxy_protocol") == 0) {
        rlcf->type = NGX_HTTP_REALIP_PROXY;
        return NGX_CONF_OK;
    }

    rlcf->type = NGX_HTTP_REALIP_HEADER;
    rlcf->hash = ngx_hash_strlow(value[1].data, value[1].data, value[1].len);
    rlcf->header = value[1];

    return NGX_CONF_OK;
}


static void *
ngx_http_realip_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_realip_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_realip_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->from = NULL;
     *     conf->hash = 0;
     *     conf->header = { 0, NULL };
     */

    conf->type = NGX_CONF_UNSET_UINT;
    conf->recursive = NGX_CONF_UNSET;

    return conf;
}

```

```
396
397 static char *
398 ngx_http_realip_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
399 {
400     ngx_http_realip_loc_conf_t  *prev = parent;
401     ngx_http_realip_loc_conf_t  *conf = child;
402
403     if (conf->from == NULL) {
404         conf->from = prev->from;
405     }
406
407     ngx_conf_merge_uint_value(conf->type, prev->type, NGX_HTTP_REALIP_XREALIP);
408     ngx_conf_merge_value(conf->recursive, prev->recursive, 0);
409
410     if (conf->header.len == 0) {
411         conf->hash = prev->hash;
412         conf->header = prev->header;
413     }
414
415     return NGX_CONF_OK;
416 }
417
418
419 static ngx_int_t
420 ngx_http_realip_init(ngx_conf_t *cf)
421 {
422     ngx_http_handler_pt        *h;
423     ngx_http_core_main_conf_t  *cmcf;
424
425     cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
426
427     h = ngx_array_push(&cmcf->phases[NGX_HTTP_POST_READ_PHASE].handlers);
428     if (h == NULL) {
429         return NGX_ERROR;
430     }
431
432     *h = ngx_http_realip_handler;
433
434     h = ngx_array_push(&cmcf->phases[NGX_HTTP_PREACCESS_PHASE].handlers);
435     if (h == NULL) {
436         return NGX_ERROR;
437     }
438
439     *h = ngx_http_realip_handler;
440
441     return NGX_OK;
442 }
```

One Level Up Top Level

src/http/modules/ngx_http_referer_module.c - nginx-1.7.10

Global variables defined

- ngx_http_referer_commands
- ngx_http_referer_module
- ngx_http_referer_module_ctx

Data types defined

- ngx_http_referer_conf_t

Functions defined

- ngx_http_add_referer
- ngx_http_add_regex_referer
- ngx_http_add_regex_server_name
- ngx_http_cmp_referer_wildcards
- ngx_http_referer_create_conf
- ngx_http_referer_merge_conf
- ngx_http_referer_variable
- ngx_http_valid_referers

Macros defined

- NGX_HTTP_REFERER_NO_URI_PART

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  #define NGX_HTTP_REFERER_NO_URI_PART  ((void *) 4)
14
15
16  typedef struct {
17      ngx_hash_combined_t      hash;
18
19  #if (NGX_PCRE)
20      ngx_array_t             *regex;
21      ngx_array_t             *server_name_regex;
22  #endif
23
24      ngx_flag_t               no_referer;
```

```
    ngx_flag_t               blocked_referer;
    ngx_flag_t               server_names;

    ngx_hash_keys_arrays_t  *keys;

    ngx_uint_t               referer_hash_max_size;
    ngx_uint_t               referer_hash_bucket_size;
} ngx_http_referer_conf_t;


static void * ngx_http_referer_create_conf(ngx_conf_t *cf);
static char * ngx_http_referer_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);
static char *ngx_http_valid_referers(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static ngx_int_t ngx_http_add_referer(ngx_conf_t *cf,
    ngx_hash_keys_arrays_t *keys, ngx_str_t *value, ngx_str_t *uri);
static ngx_int_t ngx_http_add_regex_referer(ngx_conf_t *cf,
    ngx_http_referer_conf_t *rlcf, ngx_str_t *name);
#if (NGX_PCRE)
static ngx_int_t ngx_http_add_regex_server_name(ngx_conf_t *cf,
    ngx_http_referer_conf_t *rlcf, ngx_http_regex_t *regex);
#endif
static int ngx_libc_cdecl ngx_http_cmp_referer_wildcards(const void *one,
    const void *two);


static ngx_command_t  ngx_http_referer_commands[] = {

    { ngx_string("valid_referers"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_valid_referers,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("referer_hash_max_size"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_referer_conf_t, referer_hash_max_size),
      NULL },

    { ngx_string("referer_hash_bucket_size"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_referer_conf_t, referer_hash_bucket_size),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_referer_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_referer_create_conf,          /* create location configuration */
    ngx_http_referer_merge_conf            /* merge location configuration */
};


ngx_module_t  ngx_http_referer_module = {
    NGX_MODULE_V1,
    &ngx_http_referer_module_ctx,          /* module context */
    ngx_http_referer_commands,             /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
     NULL,                                  /* init module */
```

```
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_referer_variable(ngx_http_request_t *r, ngx_http_variable_value_t *v,
     uintptr_t data)
{
    u_char                     *p, *ref, *last;
    size_t                      len;
    ngx_str_t                  *uri;
    ngx_uint_t                  i, key;
    ngx_http_referer_conf_t    *rlcf;
    u_char                      buf[256];
#if (NGX_PCRE)
    ngx_int_t                   rc;
    ngx_str_t                   referer;
#endif

    rlcf = ngx_http_get_module_loc_conf(r, ngx_http_referer_module);

    if (rlcf->hash.hash.buckets == NULL
        && rlcf->hash.wc_head == NULL
        && rlcf->hash.wc_tail == NULL
#if (NGX_PCRE)
        && rlcf->regex == NULL
        && rlcf->server_name_regex == NULL
#endif
       )
    {
        goto valid;
    }

    if (r->headers_in.referer == NULL) {
        if (rlcf->no_referer) {
            goto valid;
        }

        goto invalid;
    }

    len = r->headers_in.referer->value.len;
    ref = r->headers_in.referer->value.data;

    if (len >= sizeof("http://i.ru") - 1) {
        last = ref + len;

        if (ngx_strncasecmp(ref, (u_char *) "http://", 7) == 0) {
            ref += 7;
            len -= 7;
            goto valid_scheme;

        } else if (ngx_strncasecmp(ref, (u_char *) "https://", 8) == 0) {
            ref += 8;
            len -= 8;
            goto valid_scheme;
        }
    }

    if (rlcf->blocked_referer) {
        goto valid;
    }

    goto invalid;

valid_scheme:

    i = 0;
    key = 0;

    for (p = ref; p < last; p++) {
```

```
        if (*p == '/' || *p == ':') {
            break;
        }

        if (i == 256) {
            goto invalid;
        }

        buf[i] = ngx_tolower(*p);
        key = ngx_hash(key, buf[i++]);
    }

    uri = ngx_hash_find_combined(&rlcf->hash, key, buf, p - ref);

    if (uri) {
        goto uri;
    }

#if (NGX_PCRE)

    if (rlcf->server_name_regex) {
        referer.len = p - ref;
        referer.data = buf;

        rc = ngx_regex_exec_array(rlcf->server_name_regex, &referer,
                                  r->connection->log);

        if (rc == NGX_OK) {
            goto valid;
        }

        if (rc == NGX_ERROR) {
            return rc;
        }

        /* NGX_DECLINED */
    }

    if (rlcf->regex) {
        referer.len = len;
        referer.data = ref;

        rc = ngx_regex_exec_array(rlcf->regex, &referer, r->connection->log);

        if (rc == NGX_OK) {
            goto valid;
        }

        if (rc == NGX_ERROR) {
            return rc;
        }

        /* NGX_DECLINED */
    }

#endif

invalid:

    *v = ngx_http_variable_true_value;

    return NGX_OK;

uri:

    for ( /* void */ ; p < last; p++) {
        if (*p == '/') {
            break;
        }
    }

    len = last - p;

    if (uri == NGX_HTTP_REFERER_NO_URI_PART) {
        goto valid;
    }
```

```
253
254     if (len < uri->len || ngx_strncmp(uri->data, p, uri->len) != 0) {
255         goto invalid;
256     }
257
258 valid:
259
260     *v = ngx_http_variable_null_value;
261
262     return NGX_OK;
263 }
264
265
266 static void *
267 ngx_http_referer_create_conf(ngx_conf_t *cf)
268 {
269     ngx_http_referer_conf_t  *conf;
270
271     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_referer_conf_t));
272     if (conf == NULL) {
273         return NULL;
274     }
275
276     /*
277      * set by ngx_pcalloc():
278      *
279      *     conf->hash = { NULL };
280      *     conf->server_names = 0;
281      *     conf->keys = NULL;
282      */
283
284 #if (NGX_PCRE)
285     conf->regex = NGX_CONF_UNSET_PTR;
286     conf->server_name_regex = NGX_CONF_UNSET_PTR;
287 #endif
288
289     conf->no_referer = NGX_CONF_UNSET;
290     conf->blocked_referer = NGX_CONF_UNSET;
291     conf->referer_hash_max_size = NGX_CONF_UNSET_UINT;
292     conf->referer_hash_bucket_size = NGX_CONF_UNSET_UINT;
293
294     return conf;
295 }
296
297
298 static char *
299 ngx_http_referer_merge_conf(ngx_conf_t *cf, void *parent, void *child)
300 {
301     ngx_http_referer_conf_t *prev = parent;
302     ngx_http_referer_conf_t *conf = child;
303
304     ngx_uint_t                 n;
305     ngx_hash_init_t            hash;
306     ngx_http_server_name_t    *sn;
307     ngx_http_core_srv_conf_t  *cscf;
308
309     if (conf->keys == NULL) {
310         conf->hash = prev->hash;
311
312 #if (NGX_PCRE)
313         ngx_conf_merge_ptr_value(conf->regex, prev->regex, NULL);
314         ngx_conf_merge_ptr_value(conf->server_name_regex,
315                                  prev->server_name_regex, NULL);
316 #endif
317         ngx_conf_merge_value(conf->no_referer, prev->no_referer, 0);
318         ngx_conf_merge_value(conf->blocked_referer, prev->blocked_referer, 0);
319         ngx_conf_merge_uint_value(conf->referer_hash_max_size,
320                                   prev->referer_hash_max_size, 2048);
321         ngx_conf_merge_uint_value(conf->referer_hash_bucket_size,
322                                   prev->referer_hash_bucket_size, 64);
323
324         return NGX_CONF_OK;
325     }
326
327     if (conf->server_names == 1) {
328         cscf = ngx_http_conf_get_module_srv_conf(cf, ngx_http_core_module);
```

```
        sn = cscf->server_names.elts;
        for (n = 0; n < cscf->server_names.nelts; n++) {

#if (NGX_PCRE)
            if (sn[n].regex) {

                if (ngx_http_add_regex_server_name(cf, conf, sn[n].regex)
                    != NGX_OK)
                {
                    return NGX_CONF_ERROR;
                }

                continue;
            }
#endif

            if (ngx_http_add_referer(cf, conf->keys, &sn[n].name, NULL)
                != NGX_OK)
            {
                return NGX_CONF_ERROR;
            }
        }
    }

    if ((conf->no_referer == 1 || conf->blocked_referer == 1)
        && conf->keys->keys.nelts == 0
        && conf->keys->dns_wc_head.nelts == 0
        && conf->keys->dns_wc_tail.nelts == 0)
    {
        ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
                      "the \"none\" or \"blocked\" referers are specified "
                      "in the \"valid_referers\" directive "
                      "without any valid referer");
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_uint_value(conf->referer_hash_max_size,
                              prev->referer_hash_max_size, 2048);
    ngx_conf_merge_uint_value(conf->referer_hash_bucket_size,
                              prev->referer_hash_bucket_size, 64);
    conf->referer_hash_bucket_size = ngx_align(conf->referer_hash_bucket_size,
                                               ngx_cacheline_size);

    hash.key = ngx_hash_key_lc;
    hash.max_size = conf->referer_hash_max_size;
    hash.bucket_size = conf->referer_hash_bucket_size;
    hash.name = "referer_hash";
    hash.pool = cf->pool;

    if (conf->keys->keys.nelts) {
        hash.hash = &conf->hash.hash;
        hash.temp_pool = NULL;

        if (ngx_hash_init(&hash, conf->keys->keys.elts, conf->keys->keys.nelts)
            != NGX_OK)
        {
            return NGX_CONF_ERROR;
        }
    }

    if (conf->keys->dns_wc_head.nelts) {

        ngx_qsort(conf->keys->dns_wc_head.elts,
                  (size_t) conf->keys->dns_wc_head.nelts,
                  sizeof(ngx_hash_key_t),
                  ngx_http_cmp_referer_wildcards);

        hash.hash = NULL;
        hash.temp_pool = cf->temp_pool;

        if (ngx_hash_wildcard_init(&hash, conf->keys->dns_wc_head.elts,
                                   conf->keys->dns_wc_head.nelts)
            != NGX_OK)
        {
            return NGX_CONF_ERROR;
```

```
        }

        conf->hash.wc_head = (ngx_hash_wildcard_t *) hash.hash;
    }

    if (conf->keys->dns_wc_tail.nelts) {

        ngx_qsort(conf->keys->dns_wc_tail.elts,
                  (size_t) conf->keys->dns_wc_tail.nelts,
                  sizeof(ngx_hash_key_t),
                  ngx_http_cmp_referer_wildcards);

        hash.hash = NULL;
        hash.temp_pool = cf->temp_pool;

        if (ngx_hash_wildcard_init(&hash, conf->keys->dns_wc_tail.elts,
                                   conf->keys->dns_wc_tail.nelts)
            != NGX_OK)
        {
            return NGX_CONF_ERROR;
        }

        conf->hash.wc_tail = (ngx_hash_wildcard_t *) hash.hash;
    }

#if (NGX_PCRE)
    ngx_conf_merge_ptr_value(conf->regex, prev->regex, NULL);
    ngx_conf_merge_ptr_value(conf->server_name_regex, prev->server_name_regex,
                             NULL);
#endif

    if (conf->no_referer == NGX_CONF_UNSET) {
        conf->no_referer = 0;
    }

    if (conf->blocked_referer == NGX_CONF_UNSET) {
        conf->blocked_referer = 0;
    }

    conf->keys = NULL;

    return NGX_CONF_OK;
}


static char *
ngx_http_valid_referers(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_referer_conf_t  *rlcf = conf;

    u_char                    *p;
    ngx_str_t                 *value, uri, name;
    ngx_uint_t                 i;
    ngx_http_variable_t       *var;

    ngx_str_set(&name, "invalid_referer");

    var = ngx_http_add_variable(cf, &name, NGX_HTTP_VAR_CHANGEABLE);
    if (var == NULL) {
        return NGX_CONF_ERROR;
    }

    var->get_handler = ngx_http_referer_variable;

    if (rlcf->keys == NULL) {
        rlcf->keys = ngx_pcalloc(cf->temp_pool, sizeof(ngx_hash_keys_arrays_t));
        if (rlcf->keys == NULL) {
            return NGX_CONF_ERROR;
        }

        rlcf->keys->pool = cf->pool;
        rlcf->keys->temp_pool = cf->pool;

        if (ngx_hash_keys_array_init(rlcf->keys, NGX_HASH_SMALL) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
```

```
    }

    value = cf->args->elts;

    for (i = 1; i < cf->args->nelts; i++) {
        if (value[i].len == 0) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid referer \"%V\"", &value[i]);
            return NGX_CONF_ERROR;
        }

        if (ngx_strcmp(value[i].data, "none") == 0) {
            rlcf->no_referer = 1;
            continue;
        }

        if (ngx_strcmp(value[i].data, "blocked") == 0) {
            rlcf->blocked_referer = 1;
            continue;
        }

        if (ngx_strcmp(value[i].data, "server_names") == 0) {
            rlcf->server_names = 1;
            continue;
        }

        if (value[i].data[0] == '~') {
            if (ngx_http_add_regex_referer(cf, rlcf, &value[i]) != NGX_OK) {
                return NGX_CONF_ERROR;
            }

            continue;
        }

        ngx_str_null(&uri);

        p = (u_char *) ngx_strchr(value[i].data, '/');

        if (p) {
            uri.len = (value[i].data + value[i].len) - p;
            uri.data = p;
            value[i].len = p - value[i].data;
        }

        if (ngx_http_add_referer(cf, rlcf->keys, &value[i], &uri) != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_add_referer(ngx_conf_t *cf, ngx_hash_keys_arrays_t *keys,
    ngx_str_t *value, ngx_str_t *uri)
{
    ngx_int_t   rc;
    ngx_str_t  *u;

    if (uri == NULL || uri->len == 0) {
        u = NGX_HTTP_REFERER_NO_URI_PART;

    } else {
        u = ngx_palloc(cf->pool, sizeof(ngx_str_t));
        if (u == NULL) {
            return NGX_ERROR;
        }

        *u = *uri;
    }

    rc = ngx_hash_add_key(keys, value, u, NGX_HASH_WILDCARD_KEY);

    if (rc == NGX_OK) {
        return NGX_OK;
```

```
    }

    if (rc == NGX_DECLINED) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid hostname or wildcard \"%V\"", value);
    }

    if (rc == NGX_BUSY) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "conflicting parameter \"%V\"", value);
    }

    return NGX_ERROR;
}


static ngx_int_t
ngx_http_add_regex_referer(ngx_conf_t *cf, ngx_http_referer_conf_t *rlcf,
    ngx_str_t *name)
{
#if (NGX_PCRE)
    ngx_regex_elt_t      *re;
    ngx_regex_compile_t   rc;
    u_char                errstr[NGX_MAX_CONF_ERRSTR];

    if (name->len == 1) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "empty regex in \"%V\"", name);
        return NGX_ERROR;
    }

    if (rlcf->regex == NGX_CONF_UNSET_PTR) {
        rlcf->regex = ngx_array_create(cf->pool, 2, sizeof(ngx_regex_elt_t));
        if (rlcf->regex == NULL) {
            return NGX_ERROR;
        }
    }

    re = ngx_array_push(rlcf->regex);
    if (re == NULL) {
        return NGX_ERROR;
    }

    name->len--;
    name->data++;

    ngx_memzero(&rc, sizeof(ngx_regex_compile_t));

    rc.pattern = *name;
    rc.pool = cf->pool;
    rc.options = NGX_REGEX_CASELESS;
    rc.err.len = NGX_MAX_CONF_ERRSTR;
    rc.err.data = errstr;

    if (ngx_regex_compile(&rc) != NGX_OK) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0, "%V", &rc.err);
        return NGX_ERROR;
    }

    re->regex = rc.regex;
    re->name = name->data;

    return NGX_OK;

#else

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "the using of the regex \"%V\" requires PCRE library",
                       name);

    return NGX_ERROR;

#endif
}


#if (NGX_PCRE)
```

```
633
634 static ngx_int_t
635 ngx_http_add_regex_server_name(ngx_conf_t *cf, ngx_http_referer_conf_t *rlcf,
636     ngx_http_regex_t *regex)
637 {
638     ngx_regex_elt_t  *re;
639
640     if (rlcf->server_name_regex == NGX_CONF_UNSET_PTR) {
641         rlcf->server_name_regex = ngx_array_create(cf->pool, 2,
642                                                    sizeof(ngx_regex_elt_t));
643         if (rlcf->server_name_regex == NULL) {
644             return NGX_ERROR;
645         }
646     }
647
648     re = ngx_array_push(rlcf->server_name_regex);
649     if (re == NULL) {
650         return NGX_ERROR;
651     }
652
653     re->regex = regex->regex;
654     re->name = regex->name.data;
655
656     return NGX_OK;
657 }
658
659 #endif
660
661
662 static int ngx_libc_cdecl
663 ngx_http_cmp_referer_wildcards(const void *one, const void *two)
664 {
665     ngx_hash_key_t  *first, *second;
666
667     first = (ngx_hash_key_t *) one;
668     second = (ngx_hash_key_t *) two;
669
670     return ngx_dns_strcmp(first->key.data, second->key.data);
671 }
```

One Level Up Top Level

src/http/modules/ngx_http_rewrite_module.c - nginx-1.7.10

Global variables defined

- ngx_http_rewrite_commands
- ngx_http_rewrite_module
- ngx_http_rewrite_module_ctx

Data types defined

- ngx_http_rewrite_loc_conf_t

Functions defined

- ngx_http_rewrite
- ngx_http_rewrite_break
- ngx_http_rewrite_create_loc_conf
- ngx_http_rewrite_handler
- ngx_http_rewrite_if
- ngx_http_rewrite_if_condition
- ngx_http_rewrite_init
- ngx_http_rewrite_merge_loc_conf
- ngx_http_rewrite_return
- ngx_http_rewrite_set
- ngx_http_rewrite_value
- ngx_http_rewrite_var
- ngx_http_rewrite_variable

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_array_t  *codes;        /* uintptr_t */
15
16      ngx_uint_t    stack_size;
17
18      ngx_flag_t    log;
19      ngx_flag_t    uninitialized_variable_warn;
```

```
} ngx_http_rewrite_loc_conf_t;


static void *ngx_http_rewrite_create_loc_conf(ngx_conf_t *cf);
static char *ngx_http_rewrite_merge_loc_conf(ngx_conf_t *cf,
    void *parent, void *child);
static ngx_int_t ngx_http_rewrite_init(ngx_conf_t *cf);
static char *ngx_http_rewrite(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
static char *ngx_http_rewrite_return(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_rewrite_break(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_rewrite_if(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char * ngx_http_rewrite_if_condition(ngx_conf_t *cf,
    ngx_http_rewrite_loc_conf_t *lcf);
static char *ngx_http_rewrite_variable(ngx_conf_t *cf,
    ngx_http_rewrite_loc_conf_t *lcf, ngx_str_t *value);
static char *ngx_http_rewrite_set(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char * ngx_http_rewrite_value(ngx_conf_t *cf,
    ngx_http_rewrite_loc_conf_t *lcf, ngx_str_t *value);


static ngx_command_t  ngx_http_rewrite_commands[] = {

    { ngx_string("rewrite"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_SIF_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                       |NGX_CONF_TAKE23,
      ngx_http_rewrite,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("return"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_SIF_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                       |NGX_CONF_TAKE12,
      ngx_http_rewrite_return,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("break"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_SIF_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                       |NGX_CONF_NOARGS,
      ngx_http_rewrite_break,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("if"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_BLOCK|NGX_CONF_1MORE,
      ngx_http_rewrite_if,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("set"),
      NGX_HTTP_SRV_CONF|NGX_HTTP_SIF_CONF|NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF
                       |NGX_CONF_TAKE2,
      ngx_http_rewrite_set,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("rewrite_log"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_SIF_CONF|NGX_HTTP_LOC_CONF
                        |NGX_HTTP_LIF_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_rewrite_loc_conf_t, log),
      NULL },

    { ngx_string("uninitialized_variable_warn"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_SIF_CONF|NGX_HTTP_LOC_CONF
                        |NGX_HTTP_LIF_CONF|NGX_CONF_FLAG,
```

```
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_rewrite_loc_conf_t, uninitialized_variable_warn),
      NULL },

       ngx_null_command
};


static ngx_http_module_t  ngx_http_rewrite_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_rewrite_init,                 /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_rewrite_create_loc_conf,      /* create location configuration */
    ngx_http_rewrite_merge_loc_conf        /* merge location configuration */
};


ngx_module_t  ngx_http_rewrite_module = {
    NGX_MODULE_V1,
    &ngx_http_rewrite_module_ctx,          /* module context */
    ngx_http_rewrite_commands,             /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_rewrite_handler(ngx_http_request_t *r)
{
    ngx_int_t                    index;
    ngx_http_script_code_pt      code;
    ngx_http_script_engine_t    *e;
    ngx_http_core_srv_conf_t    *cscf;
    ngx_http_core_main_conf_t   *cmcf;
    ngx_http_rewrite_loc_conf_t  *rlcf;

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);
    cscf = ngx_http_get_module_srv_conf(r, ngx_http_core_module);
    index = cmcf->phase_engine.location_rewrite_index;

    if (r->phase_handler == index && r->loc_conf == cscf->ctx->loc_conf) {
        /* skipping location rewrite phase for server null location */
        return NGX_DECLINED;
    }

    rlcf = ngx_http_get_module_loc_conf(r, ngx_http_rewrite_module);

    if (rlcf->codes == NULL) {
        return NGX_DECLINED;
    }

    e = ngx_pcalloc(r->pool, sizeof(ngx_http_script_engine_t));
    if (e == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    e->sp = ngx_pcalloc(r->pool,
                        rlcf->stack_size * sizeof(ngx_http_variable_value_t));
    if (e->sp == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

```

```
    e->ip = rlcf->codes->elts;
    e->request = r;
    e->quote = 1;
    e->log = rlcf->log;
    e->status = NGX_DECLINED;

    while (*(uintptr_t *) e->ip) {
        code = *(ngx_http_script_code_pt *) e->ip;
        code(e);
    }

    if (e->status < NGX_HTTP_BAD_REQUEST) {
        return e->status;
    }

    if (r->err_status == 0) {
        return e->status;
    }

    return r->err_status;
}


static ngx_int_t
ngx_http_rewrite_var(ngx_http_request_t *r, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    ngx_http_variable_t          *var;
    ngx_http_core_main_conf_t    *cmcf;
    ngx_http_rewrite_loc_conf_t  *rlcf;

    rlcf = ngx_http_get_module_loc_conf(r, ngx_http_rewrite_module);

    if (rlcf->uninitialized_variable_warn == 0) {
        *v = ngx_http_variable_null_value;
        return NGX_OK;
    }

    cmcf = ngx_http_get_module_main_conf(r, ngx_http_core_module);

    var = cmcf->variables.elts;

    /*
     * the ngx_http_rewrite_module sets variables directly in r->variables,
     * and they should be handled by ngx_http_get_indexed_variable(),
     * so the handler is called only if the variable is not initialized
     */

    ngx_log_error(NGX_LOG_WARN, r->connection->log, 0,
                  "using uninitialized \"%V\" variable", &var[data].name);

    *v = ngx_http_variable_null_value;

    return NGX_OK;
}


static void *
ngx_http_rewrite_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_rewrite_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_rewrite_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->stack_size = NGX_CONF_UNSET_UINT;
    conf->log = NGX_CONF_UNSET;
    conf->uninitialized_variable_warn = NGX_CONF_UNSET;

    return conf;
}


static char *
```

```
ngx_http_rewrite_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_rewrite_loc_conf_t *prev = parent;
    ngx_http_rewrite_loc_conf_t *conf = child;

    uintptr_t  *code;

    ngx_conf_merge_value(conf->log, prev->log, 0);
    ngx_conf_merge_value(conf->uninitialized_variable_warn,
                         prev->uninitialized_variable_warn, 1);
    ngx_conf_merge_uint_value(conf->stack_size, prev->stack_size, 10);

    if (conf->codes == NULL) {
        return NGX_CONF_OK;
    }

    if (conf->codes == prev->codes) {
        return NGX_CONF_OK;
    }

    code = ngx_array_push_n(conf->codes, sizeof(uintptr_t));
    if (code == NULL) {
        return NGX_CONF_ERROR;
    }

    *code = (uintptr_t) NULL;

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_rewrite_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt        *h;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_SERVER_REWRITE_PHASE].handlers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    *h = ngx_http_rewrite_handler;

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_REWRITE_PHASE].handlers);
    if (h == NULL) {
        return NGX_ERROR;
    }

    *h = ngx_http_rewrite_handler;

    return NGX_OK;
}


static char *
ngx_http_rewrite(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_rewrite_loc_conf_t  *lcf = conf;

    ngx_str_t                         *value;
    ngx_uint_t                         last;
    ngx_regex_compile_t                rc;
    ngx_http_script_code_pt           *code;
    ngx_http_script_compile_t          sc;
    ngx_http_script_regex_code_t      *regex;
    ngx_http_script_regex_end_code_t  *regex_end;
    u_char                             errstr[NGX_MAX_CONF_ERRSTR];

    regex = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                       sizeof(ngx_http_script_regex_code_t));
    if (regex == NULL) {
        return NGX_CONF_ERROR;
    }
```

```
    ngx_memzero(regex, sizeof(ngx_http_script_regex_code_t));

    value = cf->args->elts;

    ngx_memzero(&rc, sizeof(ngx_regex_compile_t));

    rc.pattern = value[1];
    rc.err.len = NGX_MAX_CONF_ERRSTR;
    rc.err.data = errstr;

    /* TODO: NGX_REGEX_CASELESS */

    regex->regex = ngx_http_regex_compile(cf, &rc);
    if (regex->regex == NULL) {
        return NGX_CONF_ERROR;
    }

    regex->code = ngx_http_script_regex_start_code;
    regex->uri = 1;
    regex->name = value[1];

    if (value[2].data[value[2].len - 1] == '?') {

        /* the last "?" drops the original arguments */
        value[2].len--;

    } else {
        regex->add_args = 1;
    }

    last = 0;

    if (ngx_strncmp(value[2].data, "http://", sizeof("http://") - 1) == 0
        || ngx_strncmp(value[2].data, "https://", sizeof("https://") - 1) == 0
        || ngx_strncmp(value[2].data, "$scheme", sizeof("$scheme") - 1) == 0)
    {
        regex->status = NGX_HTTP_MOVED_TEMPORARILY;
        regex->redirect = 1;
        last = 1;
    }

    if (cf->args->nelts == 4) {
        if (ngx_strcmp(value[3].data, "last") == 0) {
            last = 1;

        } else if (ngx_strcmp(value[3].data, "break") == 0) {
            regex->break_cycle = 1;
            last = 1;

        } else if (ngx_strcmp(value[3].data, "redirect") == 0) {
            regex->status = NGX_HTTP_MOVED_TEMPORARILY;
            regex->redirect = 1;
            last = 1;

        } else if (ngx_strcmp(value[3].data, "permanent") == 0) {
            regex->status = NGX_HTTP_MOVED_PERMANENTLY;
            regex->redirect = 1;
            last = 1;

        } else {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid parameter \"%V\"", &value[3]);
            return NGX_CONF_ERROR;
        }
    }

    ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

    sc.cf = cf;
    sc.source = &value[2];
    sc.lengths = &regex->lengths;
    sc.values = &lcf->codes;
    sc.variables = ngx_http_script_variables_count(&value[2]);
    sc.main = regex;
    sc.complete_lengths = 1;
```

```
    sc.compile_args = !regex->redirect;

    if (ngx_http_script_compile(&sc) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    regex = sc.main;

    regex->size = sc.size;
    regex->args = sc.args;

    if (sc.variables == 0 && !sc.dup_capture) {
        regex->lengths = NULL;
    }

    regex_end = ngx_http_script_add_code(lcf->codes,
                                         sizeof(ngx_http_script_regex_end_code_t),
                                         &regex);
    if (regex_end == NULL) {
        return NGX_CONF_ERROR;
    }

    regex_end->code = ngx_http_script_regex_end_code;
    regex_end->uri = regex->uri;
    regex_end->args = regex->args;
    regex_end->add_args = regex->add_args;
    regex_end->redirect = regex->redirect;

    if (last) {
        code = ngx_http_script_add_code(lcf->codes, sizeof(uintptr_t), &regex);
        if (code == NULL) {
            return NGX_CONF_ERROR;
        }

        *code = NULL;
    }

    regex->next = (u_char *) lcf->codes->elts + lcf->codes->nelts
                                              - (u_char *) regex;

    return NGX_CONF_OK;
}


static char *
ngx_http_rewrite_return(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_rewrite_loc_conf_t  *lcf = conf;

    u_char                       *p;
    ngx_str_t                    *value, *v;
    ngx_http_script_return_code_t     *ret;
    ngx_http_compile_complex_value_t   ccv;

    ret = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                     sizeof(ngx_http_script_return_code_t));
    if (ret == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    ngx_memzero(ret, sizeof(ngx_http_script_return_code_t));

    ret->code = ngx_http_script_return_code;

    p = value[1].data;

    ret->status = ngx_atoi(p, value[1].len);

    if (ret->status == (uintptr_t) NGX_ERROR) {

        if (cf->args->nelts == 2
            && (ngx_strncmp(p, "http://", sizeof("http://") - 1) == 0
                || ngx_strncmp(p, "https://", sizeof("https://") - 1) == 0
                || ngx_strncmp(p, "$scheme", sizeof("$scheme") - 1) == 0))
```

```
        {
            ret->status = NGX_HTTP_MOVED_TEMPORARILY;
            v = &value[1];

        } else {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid return code \"%V\"", &value[1]);
            return NGX_CONF_ERROR;
        }

    } else {

        if (ret->status > 999) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid return code \"%V\"", &value[1]);
            return NGX_CONF_ERROR;
        }

        if (cf->args->nelts == 2) {
            return NGX_CONF_OK;
        }

        v = &value[2];
    }

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = v;
    ccv.complex_value = &ret->text;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_rewrite_break(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_rewrite_loc_conf_t *lcf = conf;

    ngx_http_script_code_pt  *code;

    code = ngx_http_script_start_code(cf->pool, &lcf->codes, sizeof(uintptr_t));
    if (code == NULL) {
        return NGX_CONF_ERROR;
    }

    *code = ngx_http_script_break_code;

    return NGX_CONF_OK;
}


static char *
ngx_http_rewrite_if(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_rewrite_loc_conf_t  *lcf = conf;

    void                         *mconf;
    char                         *rv;
    u_char                       *elts;
    ngx_uint_t                    i;
    ngx_conf_t                    save;
    ngx_http_module_t            *module;
    ngx_http_conf_ctx_t          *ctx, *pctx;
    ngx_http_core_loc_conf_t     *clcf, *pclcf;
    ngx_http_script_if_code_t    *if_code;
    ngx_http_rewrite_loc_conf_t  *nlcf;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_conf_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
```

```
    }

    pctx = cf->ctx;
    ctx->main_conf = pctx->main_conf;
    ctx->srv_conf = pctx->srv_conf;

    ctx->loc_conf = ngx_pcalloc(cf->pool, sizeof(void *) * ngx_http_max_module);
    if (ctx->loc_conf == NULL) {
        return NGX_CONF_ERROR;
    }

    for (i = 0; ngx_modules[i]; i++) {
        if (ngx_modules[i]->type != NGX_HTTP_MODULE) {
            continue;
        }

        module = ngx_modules[i]->ctx;

        if (module->create_loc_conf) {

            mconf = module->create_loc_conf(cf);
            if (mconf == NULL) {
                 return NGX_CONF_ERROR;
            }

            ctx->loc_conf[ngx_modules[i]->ctx_index] = mconf;
        }
    }

    pclcf = pctx->loc_conf[ngx_http_core_module.ctx_index];

    clcf = ctx->loc_conf[ngx_http_core_module.ctx_index];
    clcf->loc_conf = ctx->loc_conf;
    clcf->name = pclcf->name;
    clcf->noname = 1;

    if (ngx_http_add_location(cf, &pclcf->locations, clcf) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (ngx_http_rewrite_if_condition(cf, lcf) != NGX_CONF_OK) {
        return NGX_CONF_ERROR;
    }

    if_code = ngx_array_push_n(lcf->codes, sizeof(ngx_http_script_if_code_t));
    if (if_code == NULL) {
        return NGX_CONF_ERROR;
    }

    if_code->code = ngx_http_script_if_code;

    elts = lcf->codes->elts;


    /* the inner directives must be compiled to the same code array */

    nlcf = ctx->loc_conf[ngx_http_rewrite_module.ctx_index];
    nlcf->codes = lcf->codes;


    save = *cf;
    cf->ctx = ctx;

    if (pclcf->name.len == 0) {
        if_code->loc_conf = NULL;
        cf->cmd_type = NGX_HTTP_SIF_CONF;

    } else {
        if_code->loc_conf = ctx->loc_conf;
        cf->cmd_type = NGX_HTTP_LIF_CONF;
    }

    rv = ngx_conf_parse(cf, NULL);

    *cf = save;

```

```
    if (rv != NGX_CONF_OK) {
        return rv;
    }


    if (elts != lcf->codes->elts) {
        if_code = (ngx_http_script_if_code_t *)
                  ((u_char *) if_code + ((u_char *) lcf->codes->elts - elts));
    }

    if_code->next = (u_char *) lcf->codes->elts + lcf->codes->nelts
                                                - (u_char *) if_code;

    /* the code array belong to parent block */

    nlcf->codes = NULL;

    return NGX_CONF_OK;
}


static char *
ngx_http_rewrite_if_condition(ngx_conf_t *cf, ngx_http_rewrite_loc_conf_t *lcf)
{
    u_char                        *p;
    size_t                         len;
    ngx_str_t                     *value;
    ngx_uint_t                     cur, last;
    ngx_regex_compile_t            rc;
    ngx_http_script_code_pt       *code;
    ngx_http_script_file_code_t   *fop;
    ngx_http_script_regex_code_t  *regex;
    u_char                         errstr[NGX_MAX_CONF_ERRSTR];

    value = cf->args->elts;
    last = cf->args->nelts - 1;

    if (value[1].len < 1 || value[1].data[0] != '(') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid condition \"%V\"", &value[1]);
        return NGX_CONF_ERROR;
    }

    if (value[1].len == 1) {
        cur = 2;

    } else {
        cur = 1;
        value[1].len--;
        value[1].data++;
    }

    if (value[last].len < 1 || value[last].data[value[last].len - 1] != ')') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid condition \"%V\"", &value[last]);
        return NGX_CONF_ERROR;
    }

    if (value[last].len == 1) {
        last--;

    } else {
        value[last].len--;
        value[last].data[value[last].len] = '\0';
    }

    len = value[cur].len;
    p = value[cur].data;

    if (len > 1 && p[0] == '$') {

        if (cur != last && cur + 2 != last) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid condition \"%V\"", &value[cur]);
            return NGX_CONF_ERROR;
        }
```

```
        if (ngx_http_rewrite_variable(cf, lcf, &value[cur]) != NGX_CONF_OK) {
            return NGX_CONF_ERROR;
        }

        if (cur == last) {
            return NGX_CONF_OK;
        }

        cur++;

        len = value[cur].len;
        p = value[cur].data;

        if (len == 1 && p[0] == '=') {

            if (ngx_http_rewrite_value(cf, lcf, &value[last]) != NGX_CONF_OK) {
                return NGX_CONF_ERROR;
            }

            code = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                              sizeof(uintptr_t));
            if (code == NULL) {
                return NGX_CONF_ERROR;
            }

            *code = ngx_http_script_equal_code;

            return NGX_CONF_OK;
        }

        if (len == 2 && p[0] == '!' && p[1] == '=') {

            if (ngx_http_rewrite_value(cf, lcf, &value[last]) != NGX_CONF_OK) {
                return NGX_CONF_ERROR;
            }

            code = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                              sizeof(uintptr_t));
            if (code == NULL) {
                return NGX_CONF_ERROR;
            }

            *code = ngx_http_script_not_equal_code;
            return NGX_CONF_OK;
        }

        if ((len == 1 && p[0] == '~')
            || (len == 2 && p[0] == '~' && p[1] == '*')
            || (len == 2 && p[0] == '!' && p[1] == '~')
            || (len == 3 && p[0] == '!' && p[1] == '~' && p[2] == '*'))
        {
            regex = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                               sizeof(ngx_http_script_regex_code_t));
            if (regex == NULL) {
                return NGX_CONF_ERROR;
            }

            ngx_memzero(regex, sizeof(ngx_http_script_regex_code_t));

            ngx_memzero(&rc, sizeof(ngx_regex_compile_t));

            rc.pattern = value[last];
            rc.options = (p[len - 1] == '*') ? NGX_REGEX_CASELESS : 0;
            rc.err.len = NGX_MAX_CONF_ERRSTR;
            rc.err.data = errstr;

            regex->regex = ngx_http_regex_compile(cf, &rc);
            if (regex->regex == NULL) {
                return NGX_CONF_ERROR;
            }

            regex->code = ngx_http_script_regex_start_code;
            regex->next = sizeof(ngx_http_script_regex_code_t);
            regex->test = 1;
            if (p[0] == '!') {
```

```
                regex->negative_test = 1;
            }
            regex->name = value[last];

            return NGX_CONF_OK;
        }

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "unexpected \"%V\" in condition", &value[cur]);
        return NGX_CONF_ERROR;

    } else if ((len == 2 && p[0] == '-')
               || (len == 3 && p[0] == '!' && p[1] == '-'))
    {
        if (cur + 1 != last) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "invalid condition \"%V\"", &value[cur]);
            return NGX_CONF_ERROR;
        }

        value[last].data[value[last].len] = '\0';
        value[last].len++;

        if (ngx_http_rewrite_value(cf, lcf, &value[last]) != NGX_CONF_OK) {
            return NGX_CONF_ERROR;
        }

        fop = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                         sizeof(ngx_http_script_file_code_t));
        if (fop == NULL) {
            return NGX_CONF_ERROR;
        }

        fop->code = ngx_http_script_file_code;

        if (p[1] == 'f') {
            fop->op = ngx_http_script_file_plain;
            return NGX_CONF_OK;
        }

        if (p[1] == 'd') {
            fop->op = ngx_http_script_file_dir;
            return NGX_CONF_OK;
        }

        if (p[1] == 'e') {
            fop->op = ngx_http_script_file_exists;
            return NGX_CONF_OK;
        }

        if (p[1] == 'x') {
            fop->op = ngx_http_script_file_exec;
            return NGX_CONF_OK;
        }

        if (p[0] == '!') {
            if (p[2] == 'f') {
                fop->op = ngx_http_script_file_not_plain;
                return NGX_CONF_OK;
            }

            if (p[2] == 'd') {
                fop->op = ngx_http_script_file_not_dir;
                return NGX_CONF_OK;
            }

            if (p[2] == 'e') {
                fop->op = ngx_http_script_file_not_exists;
                return NGX_CONF_OK;
            }

            if (p[2] == 'x') {
                fop->op = ngx_http_script_file_not_exec;
                return NGX_CONF_OK;
            }
        }
```

```
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid condition \"%V\"", &value[cur]);
        return NGX_CONF_ERROR;
    }

    ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                       "invalid condition \"%V\"", &value[cur]);

    return NGX_CONF_ERROR;
}


static char *
ngx_http_rewrite_variable(ngx_conf_t *cf, ngx_http_rewrite_loc_conf_t *lcf,
    ngx_str_t *value)
{
    ngx_int_t                   index;
    ngx_http_script_var_code_t  *var_code;

    value->len--;
    value->data++;

    index = ngx_http_get_variable_index(cf, value);

    if (index == NGX_ERROR) {
        return NGX_CONF_ERROR;
    }

    var_code = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                          sizeof(ngx_http_script_var_code_t));
    if (var_code == NULL) {
        return NGX_CONF_ERROR;
    }

    var_code->code = ngx_http_script_var_code;
    var_code->index = index;

    return NGX_CONF_OK;
}


static char *
ngx_http_rewrite_set(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_rewrite_loc_conf_t  *lcf = conf;

    ngx_int_t                             index;
    ngx_str_t                            *value;
    ngx_http_variable_t                  *v;
    ngx_http_script_var_code_t           *vcode;
    ngx_http_script_var_handler_code_t   *vhcode;

    value = cf->args->elts;

    if (value[1].data[0] != '$') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid variable name \"%V\"", &value[1]);
        return NGX_CONF_ERROR;
    }

    value[1].len--;
    value[1].data++;

    v = ngx_http_add_variable(cf, &value[1], NGX_HTTP_VAR_CHANGEABLE);
    if (v == NULL) {
        return NGX_CONF_ERROR;
    }

    index = ngx_http_get_variable_index(cf, &value[1]);
    if (index == NGX_ERROR) {
        return NGX_CONF_ERROR;
    }

    if (v->get_handler == NULL
        && ngx_strncasecmp(value[1].data, (u_char *) "http_", 5) != 0
```

```
        && ngx_strncasecmp(value[1].data, (u_char *) "sent_http_", 10) != 0
        && ngx_strncasecmp(value[1].data, (u_char *) "upstream_http_", 14) != 0
        && ngx_strncasecmp(value[1].data, (u_char *) "cookie_", 7) != 0
        && ngx_strncasecmp(value[1].data, (u_char *) "upstream_cookie_", 16)
           != 0
        && ngx_strncasecmp(value[1].data, (u_char *) "arg_", 4) != 0)
    {
        v->get_handler = ngx_http_rewrite_var;
        v->data = index;
    }

    if (ngx_http_rewrite_value(cf, lcf, &value[2]) != NGX_CONF_OK) {
        return NGX_CONF_ERROR;
    }

    if (v->set_handler) {
        vhcode = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                    sizeof(ngx_http_script_var_handler_code_t));
        if (vhcode == NULL) {
            return NGX_CONF_ERROR;
        }

        vhcode->code = ngx_http_script_var_set_handler_code;
        vhcode->handler = v->set_handler;
        vhcode->data = v->data;

        return NGX_CONF_OK;
    }

    vcode = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                       sizeof(ngx_http_script_var_code_t));
    if (vcode == NULL) {
        return NGX_CONF_ERROR;
    }

    vcode->code = ngx_http_script_set_var_code;
    vcode->index = (uintptr_t) index;

    return NGX_CONF_OK;
}


static char *
ngx_http_rewrite_value(ngx_conf_t *cf, ngx_http_rewrite_loc_conf_t *lcf,
    ngx_str_t *value)
{
    ngx_int_t                               n;
    ngx_http_script_compile_t               sc;
    ngx_http_script_value_code_t           *val;
    ngx_http_script_complex_value_code_t   *complex;

    n = ngx_http_script_variables_count(value);

    if (n == 0) {
        val = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                         sizeof(ngx_http_script_value_code_t));
        if (val == NULL) {
            return NGX_CONF_ERROR;
        }

        n = ngx_atoi(value->data, value->len);

        if (n == NGX_ERROR) {
            n = 0;
        }

        val->code = ngx_http_script_value_code;
        val->value = (uintptr_t) n;
         val->text_len = (uintptr_t) value->len;
         val->text_data = (uintptr_t) value->data;

         return NGX_CONF_OK;
     }

     complex = ngx_http_script_start_code(cf->pool, &lcf->codes,
                                    sizeof(ngx_http_script_complex_value_code_t));
```

```
1008      if (complex == NULL) {
1009          return NGX_CONF_ERROR;
1010      }
1011
1012      complex->code = ngx_http_script_complex_value_code;
1013      complex->lengths = NULL;
1014
1015      ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
1016
1017      sc.cf = cf;
1018      sc.source = value;
1019      sc.lengths = &complex->lengths;
1020      sc.values = &lcf->codes;
1021      sc.variables = n;
1022      sc.complete_lengths = 1;
1023
1024      if (ngx_http_script_compile(&sc) != NGX_OK) {
1025          return NGX_CONF_ERROR;
1026      }
1027
1028      return NGX_CONF_OK;
1029  }
```

One Level Up Top Level

src/http/modules/ngx_http_scgi_module.c - nginx-1.7.10

Global variables defined

- ngx_http_scgi_cache_headers
- ngx_http_scgi_commands
- ngx_http_scgi_hide_headers
- ngx_http_scgi_module
- ngx_http_scgi_module
- ngx_http_scgi_module_ctx
- ngx_http_scgi_next_upstream_masks
- ngx_http_scgi_temp_path

Data types defined

- ngx_http_scgi_loc_conf_t
- ngx_http_scgi_main_conf_t
- ngx_http_scgi_params_t

Functions defined

- ngx_http_scgi_abort_request
- ngx_http_scgi_cache
- ngx_http_scgi_cache_key
- ngx_http_scgi_create_key
- ngx_http_scgi_create_loc_conf
- ngx_http_scgi_create_main_conf
- ngx_http_scgi_create_request
- ngx_http_scgi_eval
- ngx_http_scgi_finalize_request
- ngx_http_scgi_handler
- ngx_http_scgi_init_params
- ngx_http_scgi_merge_loc_conf
- ngx_http_scgi_pass
- ngx_http_scgi_process_header
- ngx_http_scgi_process_status_line
- ngx_http_scgi_reinit_request

- ngx_http_scgi_store

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  * Copyright (C) Manlio Perillo (manlio.perillo@gmail.com)
6  */
7
8
9  #include <ngx_config.h>
10  #include <ngx_core.h>
11  #include <ngx_http.h>
12
13
14  typedef struct {
15      ngx_array_t                caches;  /* ngx_http_file_cache_t * */
16  } ngx_http_scgi_main_conf_t;
17
18
19  typedef struct {
20      ngx_array_t               *flushes;
21      ngx_array_t               *lengths;
22      ngx_array_t               *values;
23      ngx_uint_t                 number;
24      ngx_hash_t                 hash;
25  } ngx_http_scgi_params_t;
26
27
28  typedef struct {
29      ngx_http_upstream_conf_t   upstream;
30
31      ngx_http_scgi_params_t     params;
32  #if (NGX_HTTP_CACHE)
33      ngx_http_scgi_params_t     params_cache;
34  #endif
35      ngx_array_t               *params_source;
36
37      ngx_array_t               *scgi_lengths;
38      ngx_array_t               *scgi_values;
39
40  #if (NGX_HTTP_CACHE)
41      ngx_http_complex_value_t   cache_key;
42  #endif
43  } ngx_http_scgi_loc_conf_t;
44
45
46  static ngx_int_t ngx_http_scgi_eval(ngx_http_request_t *r,
47      ngx_http_scgi_loc_conf_t *scf);
48  static ngx_int_t ngx_http_scgi_create_request(ngx_http_request_t *r);
49  static ngx_int_t ngx_http_scgi_reinit_request(ngx_http_request_t *r);
50  static ngx_int_t ngx_http_scgi_process_status_line(ngx_http_request_t *r);
51  static ngx_int_t ngx_http_scgi_process_header(ngx_http_request_t *r);
52  static void ngx_http_scgi_abort_request(ngx_http_request_t *r);
53  static void ngx_http_scgi_finalize_request(ngx_http_request_t *r, ngx_int_t rc);
54
55  static void *ngx_http_scgi_create_main_conf(ngx_conf_t *cf);
56  static void *ngx_http_scgi_create_loc_conf(ngx_conf_t *cf);
57  static char *ngx_http_scgi_merge_loc_conf(ngx_conf_t *cf, void *parent,
58      void *child);
59  static ngx_int_t ngx_http_scgi_init_params(ngx_conf_t *cf,
60      ngx_http_scgi_loc_conf_t *conf, ngx_http_scgi_params_t *params,
61      ngx_keyval_t *default_params);
62
63  static char *ngx_http_scgi_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf);
64  static char *ngx_http_scgi_store(ngx_conf_t *cf, ngx_command_t *cmd,
65      void *conf);
66
67  #if (NGX_HTTP_CACHE)
68  static ngx_int_t ngx_http_scgi_create_key(ngx_http_request_t *r);
69  static char *ngx_http_scgi_cache(ngx_conf_t *cf, ngx_command_t *cmd,
70      void *conf);
```

```
static char *ngx_http_scgi_cache_key(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
#endif


static ngx_conf_bitmask_t ngx_http_scgi_next_upstream_masks[] = {
    { ngx_string("error"), NGX_HTTP_UPSTREAM_FT_ERROR },
    { ngx_string("timeout"), NGX_HTTP_UPSTREAM_FT_TIMEOUT },
    { ngx_string("invalid_header"), NGX_HTTP_UPSTREAM_FT_INVALID_HEADER },
    { ngx_string("http_500"), NGX_HTTP_UPSTREAM_FT_HTTP_500 },
    { ngx_string("http_503"), NGX_HTTP_UPSTREAM_FT_HTTP_503 },
    { ngx_string("http_403"), NGX_HTTP_UPSTREAM_FT_HTTP_403 },
    { ngx_string("http_404"), NGX_HTTP_UPSTREAM_FT_HTTP_404 },
    { ngx_string("updating"), NGX_HTTP_UPSTREAM_FT_UPDATING },
    { ngx_string("off"), NGX_HTTP_UPSTREAM_FT_OFF },
    { ngx_null_string, 0 }
};


ngx_module_t  ngx_http_scgi_module;


static ngx_command_t ngx_http_scgi_commands[] = {

    { ngx_string("scgi_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_scgi_pass,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

     { ngx_string("scgi_store"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
       ngx_http_scgi_store,
       NGX_HTTP_LOC_CONF_OFFSET,
       0,
       NULL },

     { ngx_string("scgi_store_access"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
       ngx_conf_set_access_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_scgi_loc_conf_t, upstream.store_access),
       NULL },

     { ngx_string("scgi_buffering"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
       ngx_conf_set_flag_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_scgi_loc_conf_t, upstream.buffering),
       NULL },

     { ngx_string("scgi_ignore_client_abort"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
       ngx_conf_set_flag_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_scgi_loc_conf_t, upstream.ignore_client_abort),
       NULL },

     { ngx_string("scgi_bind"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
       ngx_http_upstream_bind_set_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_scgi_loc_conf_t, upstream.local),
       NULL },

     { ngx_string("scgi_connect_timeout"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
       ngx_conf_set_msec_slot,
       NGX_HTTP_LOC_CONF_OFFSET,
       offsetof(ngx_http_scgi_loc_conf_t, upstream.connect_timeout),
       NULL },

     { ngx_string("scgi_send_timeout"),
       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
       ngx_conf_set_msec_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.send_timeout),
      NULL },

    { ngx_string("scgi_buffer_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.buffer_size),
      NULL },

    { ngx_string("scgi_pass_request_headers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.pass_request_headers),
      NULL },

    { ngx_string("scgi_pass_request_body"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.pass_request_body),
      NULL },

    { ngx_string("scgi_intercept_errors"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.intercept_errors),
      NULL },

    { ngx_string("scgi_read_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.read_timeout),
      NULL },

    { ngx_string("scgi_buffers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_conf_set_bufs_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.bufs),
      NULL },

    { ngx_string("scgi_busy_buffers_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.busy_buffers_size_conf),
      NULL },

    { ngx_string("scgi_force_ranges"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.force_ranges),
      NULL },

    { ngx_string("scgi_limit_rate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.limit_rate),
      NULL },

#if (NGX_HTTP_CACHE)

    { ngx_string("scgi_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_scgi_cache,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

```

```
    { ngx_string("scgi_cache_key"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_scgi_cache_key,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("scgi_cache_path"),
      NGX_HTTP_MAIN_CONF|NGX_CONF_2MORE,
      ngx_http_file_cache_set_slot,
      NGX_HTTP_MAIN_CONF_OFFSET,
      offsetof(ngx_http_scgi_main_conf_t, caches),
      &ngx_http_scgi_module },

    { ngx_string("scgi_cache_bypass"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_set_predicate_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_bypass),
      NULL },

    { ngx_string("scgi_no_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_set_predicate_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.no_cache),
      NULL },

    { ngx_string("scgi_cache_valid"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_file_cache_valid_set_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_valid),
      NULL },

    { ngx_string("scgi_cache_min_uses"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_min_uses),
      NULL },

    { ngx_string("scgi_cache_use_stale"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_use_stale),
      &ngx_http_scgi_next_upstream_masks },

    { ngx_string("scgi_cache_methods"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_methods),
      &ngx_http_upstream_cache_method_mask },

    { ngx_string("scgi_cache_lock"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_lock),
      NULL },

    { ngx_string("scgi_cache_lock_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_lock_timeout),
      NULL },

    { ngx_string("scgi_cache_lock_age"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_lock_age),
      NULL },
```

```
299
300     { ngx_string("scgi_cache_revalidate"),
301       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
302       ngx_conf_set_flag_slot,
303       NGX_HTTP_LOC_CONF_OFFSET,
304       offsetof(ngx_http_scgi_loc_conf_t, upstream.cache_revalidate),
305       NULL },
306
307 #endif
308
309     { ngx_string("scgi_temp_path"),
310       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1234,
311       ngx_conf_set_path_slot,
312       NGX_HTTP_LOC_CONF_OFFSET,
313       offsetof(ngx_http_scgi_loc_conf_t, upstream.temp_path),
314       NULL },
315
316     { ngx_string("scgi_max_temp_file_size"),
317       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
318       ngx_conf_set_size_slot,
319       NGX_HTTP_LOC_CONF_OFFSET,
320       offsetof(ngx_http_scgi_loc_conf_t, upstream.max_temp_file_size_conf),
321       NULL },
322
323     { ngx_string("scgi_temp_file_write_size"),
324       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
325       ngx_conf_set_size_slot,
326       NGX_HTTP_LOC_CONF_OFFSET,
327       offsetof(ngx_http_scgi_loc_conf_t, upstream.temp_file_write_size_conf),
328       NULL },
329
330     { ngx_string("scgi_next_upstream"),
331       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
332       ngx_conf_set_bitmask_slot,
333       NGX_HTTP_LOC_CONF_OFFSET,
334       offsetof(ngx_http_scgi_loc_conf_t, upstream.next_upstream),
335       &ngx_http_scgi_next_upstream_masks },
336
337     { ngx_string("scgi_next_upstream_tries"),
338       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
339       ngx_conf_set_num_slot,
340       NGX_HTTP_LOC_CONF_OFFSET,
341       offsetof(ngx_http_scgi_loc_conf_t, upstream.next_upstream_tries),
342       NULL },
343
344     { ngx_string("scgi_next_upstream_timeout"),
345       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
346       ngx_conf_set_msec_slot,
347       NGX_HTTP_LOC_CONF_OFFSET,
348       offsetof(ngx_http_scgi_loc_conf_t, upstream.next_upstream_timeout),
349       NULL },
350
351     { ngx_string("scgi_param"),
352       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE23,
353       ngx_http_upstream_param_set_slot,
354       NGX_HTTP_LOC_CONF_OFFSET,
355       offsetof(ngx_http_scgi_loc_conf_t, params_source),
356       NULL },
357
358     { ngx_string("scgi_pass_header"),
359       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
360       ngx_conf_set_str_array_slot,
361       NGX_HTTP_LOC_CONF_OFFSET,
362       offsetof(ngx_http_scgi_loc_conf_t, upstream.pass_headers),
363       NULL },
364
365     { ngx_string("scgi_hide_header"),
366       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
367       ngx_conf_set_str_array_slot,
368       NGX_HTTP_LOC_CONF_OFFSET,
369       offsetof(ngx_http_scgi_loc_conf_t, upstream.hide_headers),
370       NULL },
371
372     { ngx_string("scgi_ignore_headers"),
373       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
374       ngx_conf_set_bitmask_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_scgi_loc_conf_t, upstream.ignore_headers),
      &ngx_http_upstream_ignore_headers_masks },

      ngx_null_command
};


static ngx_http_module_t ngx_http_scgi_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_scgi_create_main_conf,        /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_scgi_create_loc_conf,         /* create location configuration */
    ngx_http_scgi_merge_loc_conf           /* merge location configuration */
};


ngx_module_t ngx_http_scgi_module = {
    NGX_MODULE_V1,
    &ngx_http_scgi_module_ctx,             /* module context */
    ngx_http_scgi_commands,                /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t ngx_http_scgi_hide_headers[] = {
    ngx_string("Status"),
    ngx_string("X-Accel-Expires"),
    ngx_string("X-Accel-Redirect"),
    ngx_string("X-Accel-Limit-Rate"),
    ngx_string("X-Accel-Buffering"),
    ngx_string("X-Accel-Charset"),
    ngx_null_string
};


#if (NGX_HTTP_CACHE)

static ngx_keyval_t  ngx_http_scgi_cache_headers[] = {
    { ngx_string("HTTP_IF_MODIFIED_SINCE"),
      ngx_string("$upstream_cache_last_modified") },
    { ngx_string("HTTP_IF_UNMODIFIED_SINCE"), ngx_string("") },
    { ngx_string("HTTP_IF_NONE_MATCH"), ngx_string("$upstream_cache_etag") },
    { ngx_string("HTTP_IF_MATCH"), ngx_string("") },
    { ngx_string("HTTP_RANGE"), ngx_string("") },
    { ngx_string("HTTP_IF_RANGE"), ngx_string("") },
    { ngx_null_string, ngx_null_string }
};

#endif


static ngx_path_init_t ngx_http_scgi_temp_path = {
    ngx_string(NGX_HTTP_SCGI_TEMP_PATH), { 1, 2, 0 }
};


static ngx_int_t
ngx_http_scgi_handler(ngx_http_request_t *r)
{
    ngx_int_t                    rc;
    ngx_http_status_t           *status;
```

```
    ngx_http_upstream_t        *u;
    ngx_http_scgi_loc_conf_t   *scf;
#if (NGX_HTTP_CACHE)
    ngx_http_scgi_main_conf_t  *smcf;
#endif

    if (ngx_http_upstream_create(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    status = ngx_pcalloc(r->pool, sizeof(ngx_http_status_t));
    if (status == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    ngx_http_set_ctx(r, status, ngx_http_scgi_module);

    scf = ngx_http_get_module_loc_conf(r, ngx_http_scgi_module);

    if (scf->scgi_lengths) {
        if (ngx_http_scgi_eval(r, scf) != NGX_OK) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }
    }

    u = r->upstream;

    ngx_str_set(&u->schema, "scgi://");
    u->output.tag = (ngx_buf_tag_t) &ngx_http_scgi_module;

    u->conf = &scf->upstream;

#if (NGX_HTTP_CACHE)
    smcf = ngx_http_get_module_main_conf(r, ngx_http_scgi_module);

    u->caches = &smcf->caches;
    u->create_key = ngx_http_scgi_create_key;
#endif

    u->create_request = ngx_http_scgi_create_request;
    u->reinit_request = ngx_http_scgi_reinit_request;
    u->process_header = ngx_http_scgi_process_status_line;
    u->abort_request = ngx_http_scgi_abort_request;
    u->finalize_request = ngx_http_scgi_finalize_request;
    r->state = 0;

    u->buffering = scf->upstream.buffering;

    u->pipe = ngx_pcalloc(r->pool, sizeof(ngx_event_pipe_t));
    if (u->pipe == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    u->pipe->input_filter = ngx_event_pipe_copy_input_filter;
    u->pipe->input_ctx = r;

    rc = ngx_http_read_client_request_body(r, ngx_http_upstream_init);

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        return rc;
    }

    return NGX_DONE;
}


static ngx_int_t
ngx_http_scgi_eval(ngx_http_request_t *r, ngx_http_scgi_loc_conf_t * scf)
{
    ngx_url_t             url;
    ngx_http_upstream_t  *u;

    ngx_memzero(&url, sizeof(ngx_url_t));

    if (ngx_http_script_run(r, &url.url, scf->scgi_lengths->elts, 0,
                            scf->scgi_values->elts)
```

```
        == NULL)
    {
        return NGX_ERROR;
    }

    url.no_resolve = 1;

    if (ngx_parse_url(r->pool, &url) != NGX_OK) {
        if (url.err) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "%s in upstream \"%V\"", url.err, &url.url);
        }

        return NGX_ERROR;
    }

    u = r->upstream;

    u->resolved = ngx_pcalloc(r->pool, sizeof(ngx_http_upstream_resolved_t));
    if (u->resolved == NULL) {
        return NGX_ERROR;
    }

    if (url.addrs && url.addrs[0].sockaddr) {
        u->resolved->sockaddr = url.addrs[0].sockaddr;
        u->resolved->socklen = url.addrs[0].socklen;
        u->resolved->naddrs = 1;
        u->resolved->host = url.addrs[0].name;

    } else {
        u->resolved->host = url.host;
        u->resolved->port = url.port;
        u->resolved->no_port = url.no_port;
    }

    return NGX_OK;
}


#if (NGX_HTTP_CACHE)

static ngx_int_t
ngx_http_scgi_create_key(ngx_http_request_t *r)
{
    ngx_str_t                  *key;
    ngx_http_scgi_loc_conf_t  *scf;

    key = ngx_array_push(&r->cache->keys);
    if (key == NULL) {
        return NGX_ERROR;
    }

    scf = ngx_http_get_module_loc_conf(r, ngx_http_scgi_module);

    if (ngx_http_complex_value(r, &scf->cache_key, key) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}

#endif


static ngx_int_t
ngx_http_scgi_create_request(ngx_http_request_t *r)
{
    off_t                        content_length_n;
    u_char                       ch, *key, *val, *lowcase_key;
    size_t                       len, key_len, val_len, allocated;
    ngx_buf_t                   *b;
    ngx_str_t                    content_length;
    ngx_uint_t                   i, n, hash, skip_empty, header_params;
    ngx_chain_t                 *cl, *body;
    ngx_list_part_t             *part;
    ngx_table_elt_t             *header, **ignored;
```

```
    ngx_http_scgi_params_t       *params;
    ngx_http_script_code_pt       code;
    ngx_http_script_engine_t      e, le;
    ngx_http_scgi_loc_conf_t     *scf;
    ngx_http_script_len_code_pt   lcode;
    u_char                        buffer[NGX_OFF_T_LEN];

    content_length_n = 0;
    body = r->upstream->request_bufs;

    while (body) {
        content_length_n += ngx_buf_size(body->buf);
        body = body->next;
    }

    content_length.data = buffer;
    content_length.len = ngx_sprintf(buffer, "%O", content_length_n) - buffer;

    len = sizeof("CONTENT_LENGTH") + content_length.len + 1;

    header_params = 0;
    ignored = NULL;

    scf = ngx_http_get_module_loc_conf(r, ngx_http_scgi_module);

#if (NGX_HTTP_CACHE)
    params = r->upstream->cacheable ? &scf->params_cache : &scf->params;
#else
    params = &scf->params;
#endif

    if (params->lengths) {
        ngx_memzero(&le, sizeof(ngx_http_script_engine_t));

        ngx_http_script_flush_no_cacheable_variables(r, params->flushes);
        le.flushed = 1;

        le.ip = params->lengths->elts;
        le.request = r;

        while (*(uintptr_t *) le.ip) {

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            key_len = lcode(&le);

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            skip_empty = lcode(&le);

            for (val_len = 0; *(uintptr_t *) le.ip; val_len += lcode(&le)) {
                lcode = *(ngx_http_script_len_code_pt *) le.ip;
            }
            le.ip += sizeof(uintptr_t);

            if (skip_empty && val_len == 0) {
                continue;
            }

            len += key_len + val_len + 1;
        }
    }

    if (scf->upstream.pass_request_headers) {

        allocated = 0;
        lowcase_key = NULL;

        if (params->number) {
            n = 0;
            part = &r->headers_in.headers.part;

            while (part) {
                n += part->nelts;
                part = part->next;
            }

            ignored = ngx_palloc(r->pool, n * sizeof(void *));
```

```
            if (ignored == NULL) {
                return NGX_ERROR;
            }
        }

        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            if (params->number) {
                if (allocated < header[i].key.len) {
                    allocated = header[i].key.len + 16;
                    lowcase_key = ngx_pnalloc(r->pool, allocated);
                    if (lowcase_key == NULL) {
                        return NGX_ERROR;
                    }
                }

                hash = 0;

                for (n = 0; n < header[i].key.len; n++) {
                    ch = header[i].key.data[n];

                    if (ch >= 'A' && ch <= 'Z') {
                        ch |= 0x20;

                    } else if (ch == '-') {
                        ch = '_';
                    }

                    hash = ngx_hash(hash, ch);
                    lowcase_key[n] = ch;
                }

                if (ngx_hash_find(&params->hash, hash, lowcase_key, n)) {
                    ignored[header_params++] = &header[i];
                    continue;
                }
            }

            len += sizeof("HTTP_") - 1 + header[i].key.len + 1
                + header[i].value.len + 1;
        }
    }

    /* netstring: "length:" + packet + "," */

    b = ngx_create_temp_buf(r->pool, NGX_SIZE_T_LEN + 1 + len + 1);
    if (b == NULL) {
        return NGX_ERROR;
    }

    cl = ngx_alloc_chain_link(r->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    cl->buf = b;

    b->last = ngx_sprintf(b->last, "%ui:CONTENT_LENGTH%Z%V%Z",
                          len, &content_length);

    if (params->lengths) {
        ngx_memzero(&e, sizeof(ngx_http_script_engine_t));

```

```
        e.ip = params->values->elts;
        e.pos = b->last;
        e.request = r;
        e.flushed = 1;

        le.ip = params->lengths->elts;

        while (*(uintptr_t *) le.ip) {

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            lcode(&le); /* key length */

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            skip_empty = lcode(&le);

            for (val_len = 0; *(uintptr_t *) le.ip; val_len += lcode(&le)) {
                lcode = *(ngx_http_script_len_code_pt *) le.ip;
            }
            le.ip += sizeof(uintptr_t);

            if (skip_empty && val_len == 0) {
                e.skip = 1;

                while (*(uintptr_t *) e.ip) {
                    code = *(ngx_http_script_code_pt *) e.ip;
                    code((ngx_http_script_engine_t *) &e);
                }
                e.ip += sizeof(uintptr_t);

                e.skip = 0;

                continue;
            }

#if (NGX_DEBUG)
            key = e.pos;
#endif
            code = *(ngx_http_script_code_pt *) e.ip;
            code((ngx_http_script_engine_t *) & e);

#if (NGX_DEBUG)
            val = e.pos;
#endif
            while (*(uintptr_t *) e.ip) {
                code = *(ngx_http_script_code_pt *) e.ip;
                code((ngx_http_script_engine_t *) &e);
            }
            *e.pos++ = '\0';
            e.ip += sizeof(uintptr_t);

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "scgi param: \"%s: %s\"", key, val);
        }

        b->last = e.pos;
    }

    if (scf->upstream.pass_request_headers) {

        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            for (n = 0; n < header_params; n++) {
                if (&header[i] == ignored[n]) {
```

```
                    goto next;
                }
            }

            key = b->last;
            b->last = ngx_cpymem(key, "HTTP_", sizeof("HTTP_") - 1);

            for (n = 0; n < header[i].key.len; n++) {
                ch = header[i].key.data[n];

                if (ch >= 'a' && ch <= 'z') {
                    ch &= ~0x20;

                } else if (ch == '-') {
                    ch = '_';
                }

                *b->last++ = ch;
            }

            *b->last++ = (u_char) 0;

            val = b->last;
            b->last = ngx_copy(val, header[i].value.data, header[i].value.len);
            *b->last++ = (u_char) 0;

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "scgi param: \"%s: %s\"", key, val);

        next:

            continue;
         }
    }

    *b->last++ = (u_char) ',';

    if (scf->upstream.pass_request_body) {
        body = r->upstream->request_bufs;
        r->upstream->request_bufs = cl;

        while (body) {
            b = ngx_alloc_buf(r->pool);
            if (b == NULL) {
                return NGX_ERROR;
            }

            ngx_memcpy(b, body->buf, sizeof(ngx_buf_t));

            cl->next = ngx_alloc_chain_link(r->pool);
            if (cl->next == NULL) {
                return NGX_ERROR;
            }

            cl = cl->next;
            cl->buf = b;

            body = body->next;
        }

    } else {
        r->upstream->request_bufs = cl;
    }

    cl->next = NULL;

    return NGX_OK;
}


static ngx_int_t
ngx_http_scgi_reinit_request(ngx_http_request_t *r)
{
    ngx_http_status_t  *status;

    status = ngx_http_get_module_ctx(r, ngx_http_scgi_module);
```

```
907
908     if (status == NULL) {
909         return NGX_OK;
910     }
911
912     status->code = 0;
913     status->count = 0;
914     status->start = NULL;
915     status->end = NULL;
916
917     r->upstream->process_header = ngx_http_scgi_process_status_line;
918     r->state = 0;
919
920     return NGX_OK;
921 }
922
923
924 static ngx_int_t
925 ngx_http_scgi_process_status_line(ngx_http_request_t *r)
926 {
927     size_t                len;
928     ngx_int_t             rc;
929     ngx_http_status_t    *status;
930     ngx_http_upstream_t  *u;
931
932     status = ngx_http_get_module_ctx(r, ngx_http_scgi_module);
933
934     if (status == NULL) {
935         return NGX_ERROR;
936     }
937
938     u = r->upstream;
939
940     rc = ngx_http_parse_status_line(r, &u->buffer, status);
941
942     if (rc == NGX_AGAIN) {
943         return rc;
944     }
945
946     if (rc == NGX_ERROR) {
947         u->process_header = ngx_http_scgi_process_header;
948         return ngx_http_scgi_process_header(r);
949     }
950
951     if (u->state && u->state->status == 0) {
952         u->state->status = status->code;
953     }
954
955     u->headers_in.status_n = status->code;
956
957     len = status->end - status->start;
958     u->headers_in.status_line.len = len;
959
960     u->headers_in.status_line.data = ngx_pnalloc(r->pool, len);
961     if (u->headers_in.status_line.data == NULL) {
962         return NGX_ERROR;
963     }
964
965     ngx_memcpy(u->headers_in.status_line.data, status->start, len);
966
967     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
968                    "http scgi status %ui \"%V\"",
969                    u->headers_in.status_n, &u->headers_in.status_line);
970
971     u->process_header = ngx_http_scgi_process_header;
972
973     return ngx_http_scgi_process_header(r);
974 }
975
976
977 static ngx_int_t
978 ngx_http_scgi_process_header(ngx_http_request_t *r)
979 {
980     ngx_str_t                      *status_line;
981     ngx_int_t                       rc, status;
982     ngx_table_elt_t                *h;
```

```
    ngx_http_upstream_t           *u;
    ngx_http_upstream_header_t    *hh;
    ngx_http_upstream_main_conf_t  *umcf;

    umcf = ngx_http_get_module_main_conf(r, ngx_http_upstream_module);

    for ( ;; ) {

        rc = ngx_http_parse_header_line(r, &r->upstream->buffer, 1);

        if (rc == NGX_OK) {

            /* a header line has been parsed successfully */

            h = ngx_list_push(&r->upstream->headers_in.headers);
            if (h == NULL) {
                return NGX_ERROR;
            }

            h->hash = r->header_hash;

            h->key.len = r->header_name_end - r->header_name_start;
            h->value.len = r->header_end - r->header_start;

            h->key.data = ngx_pnalloc(r->pool,
                                      h->key.len + 1 + h->value.len + 1
                                      + h->key.len);
            if (h->key.data == NULL) {
                return NGX_ERROR;
            }

            h->value.data = h->key.data + h->key.len + 1;
            h->lowcase_key = h->key.data + h->key.len + 1 + h->value.len + 1;

            ngx_memcpy(h->key.data, r->header_name_start, h->key.len);
            h->key.data[h->key.len] = '\0';
            ngx_memcpy(h->value.data, r->header_start, h->value.len);
            h->value.data[h->value.len] = '\0';

            if (h->key.len == r->lowcase_index) {
                ngx_memcpy(h->lowcase_key, r->lowcase_header, h->key.len);

            } else {
                ngx_strlow(h->lowcase_key, h->key.data, h->key.len);
            }

            hh = ngx_hash_find(&umcf->headers_in_hash, h->hash,
                               h->lowcase_key, h->key.len);

            if (hh && hh->handler(r, h, hh->offset) != NGX_OK) {
                return NGX_ERROR;
            }

            ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http scgi header: \"%V: %V\"", &h->key, &h->value);

            continue;
        }

        if (rc == NGX_HTTP_PARSE_HEADER_DONE) {

            /* a whole header has been parsed successfully */

            ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "http scgi header done");

            u = r->upstream;

            if (u->headers_in.status_n) {
                goto done;
            }

            if (u->headers_in.status) {
                status_line = &u->headers_in.status->value;

                status = ngx_atoi(status_line->data, 3);
```

```
1059                if (status == NGX_ERROR) {
1060                    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1061                                  "upstream sent invalid status \"%V\"",
1062                                  status_line);
1063                    return NGX_HTTP_UPSTREAM_INVALID_HEADER;
1064                }
1065
1066                u->headers_in.status_n = status;
1067                u->headers_in.status_line = *status_line;
1068
1069            } else if (u->headers_in.location) {
1070                u->headers_in.status_n = 302;
1071                ngx_str_set(&u->headers_in.status_line,
1072                            "302 Moved Temporarily");
1073
1074            } else {
1075                u->headers_in.status_n = 200;
1076                ngx_str_set(&u->headers_in.status_line, "200 OK");
1077            }
1078
1079            if (u->state && u->state->status == 0) {
1080                u->state->status = u->headers_in.status_n;
1081            }
1082
1083        done:
1084
1085            if (u->headers_in.status_n == NGX_HTTP_SWITCHING_PROTOCOLS
1086                && r->headers_in.upgrade)
1087            {
1088                u->upgrade = 1;
1089            }
1090
1091            return NGX_OK;
1092        }
1093
1094        if (rc == NGX_AGAIN) {
1095            return NGX_AGAIN;
1096        }
1097
1098        /* there was error while a header line parsing */
1099
1100        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
1101                      "upstream sent invalid header");
1102
1103        return NGX_HTTP_UPSTREAM_INVALID_HEADER;
1104    }
1105 }
1106
1107
1108 static void
1109 ngx_http_scgi_abort_request(ngx_http_request_t *r)
1110 {
1111     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1112                    "abort http scgi request");
1113
1114     return;
1115 }
1116
1117
1118 static void
1119 ngx_http_scgi_finalize_request(ngx_http_request_t *r, ngx_int_t rc)
1120 {
1121     ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1122                    "finalize http scgi request");
1123
1124     return;
1125 }
1126
1127
1128 static void *
1129 ngx_http_scgi_create_main_conf(ngx_conf_t *cf)
1130 {
1131     ngx_http_scgi_main_conf_t  *conf;
1132
1133     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_scgi_main_conf_t));
1134     if (conf == NULL) {
```

```
1135        return NULL;
1136    }
1137
1138 #if (NGX_HTTP_CACHE)
1139    if (ngx_array_init(&conf->caches, cf->pool, 4,
1140                       sizeof(ngx_http_file_cache_t *))
1141        != NGX_OK)
1142    {
1143        return NULL;
1144    }
1145 #endif
1146
1147    return conf;
1148 }
1149
1150
1151 static void *
1152 ngx_http_scgi_create_loc_conf(ngx_conf_t *cf)
1153 {
1154    ngx_http_scgi_loc_conf_t  *conf;
1155
1156    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_scgi_loc_conf_t));
1157    if (conf == NULL) {
1158        return NULL;
1159    }
1160
1161    conf->upstream.store = NGX_CONF_UNSET;
1162    conf->upstream.store_access = NGX_CONF_UNSET_UINT;
1163    conf->upstream.next_upstream_tries = NGX_CONF_UNSET_UINT;
1164    conf->upstream.buffering = NGX_CONF_UNSET;
1165    conf->upstream.ignore_client_abort = NGX_CONF_UNSET;
1166    conf->upstream.force_ranges = NGX_CONF_UNSET;
1167
1168    conf->upstream.local = NGX_CONF_UNSET_PTR;
1169
1170    conf->upstream.connect_timeout = NGX_CONF_UNSET_MSEC;
1171    conf->upstream.send_timeout = NGX_CONF_UNSET_MSEC;
1172    conf->upstream.read_timeout = NGX_CONF_UNSET_MSEC;
1173    conf->upstream.next_upstream_timeout = NGX_CONF_UNSET_MSEC;
1174
1175    conf->upstream.send_lowat = NGX_CONF_UNSET_SIZE;
1176    conf->upstream.buffer_size = NGX_CONF_UNSET_SIZE;
1177    conf->upstream.limit_rate = NGX_CONF_UNSET_SIZE;
1178
1179    conf->upstream.busy_buffers_size_conf = NGX_CONF_UNSET_SIZE;
1180    conf->upstream.max_temp_file_size_conf = NGX_CONF_UNSET_SIZE;
1181    conf->upstream.temp_file_write_size_conf = NGX_CONF_UNSET_SIZE;
1182
1183    conf->upstream.pass_request_headers = NGX_CONF_UNSET;
1184    conf->upstream.pass_request_body = NGX_CONF_UNSET;
1185
1186 #if (NGX_HTTP_CACHE)
1187    conf->upstream.cache = NGX_CONF_UNSET;
1188    conf->upstream.cache_min_uses = NGX_CONF_UNSET_UINT;
1189    conf->upstream.cache_bypass = NGX_CONF_UNSET_PTR;
1190    conf->upstream.no_cache = NGX_CONF_UNSET_PTR;
1191    conf->upstream.cache_valid = NGX_CONF_UNSET_PTR;
1192    conf->upstream.cache_lock = NGX_CONF_UNSET;
1193    conf->upstream.cache_lock_timeout = NGX_CONF_UNSET_MSEC;
1194    conf->upstream.cache_lock_age = NGX_CONF_UNSET_MSEC;
1195    conf->upstream.cache_revalidate = NGX_CONF_UNSET;
1196 #endif
1197
1198    conf->upstream.hide_headers = NGX_CONF_UNSET_PTR;
1199    conf->upstream.pass_headers = NGX_CONF_UNSET_PTR;
1200
1201    conf->upstream.intercept_errors = NGX_CONF_UNSET;
1202
1203    /* "scgi_cyclic_temp_file" is disabled */
1204    conf->upstream.cyclic_temp_file = 0;
1205
1206    conf->upstream.change_buffering = 1;
1207
1208    ngx_str_set(&conf->upstream.module, "scgi");
1209
1210    return conf;
```

```
}


static char *
ngx_http_scgi_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_scgi_loc_conf_t *prev = parent;
    ngx_http_scgi_loc_conf_t *conf = child;

    size_t                        size;
    ngx_int_t                     rc;
    ngx_hash_init_t               hash;
    ngx_http_core_loc_conf_t     *clcf;

#if (NGX_HTTP_CACHE)

    if (conf->upstream.store > 0) {
        conf->upstream.cache = 0;
    }

    if (conf->upstream.cache > 0) {
        conf->upstream.store = 0;
    }

#endif

    if (conf->upstream.store == NGX_CONF_UNSET) {
        ngx_conf_merge_value(conf->upstream.store, prev->upstream.store, 0);

        conf->upstream.store_lengths = prev->upstream.store_lengths;
        conf->upstream.store_values = prev->upstream.store_values;
    }

    ngx_conf_merge_uint_value(conf->upstream.store_access,
                              prev->upstream.store_access, 0600);

    ngx_conf_merge_uint_value(conf->upstream.next_upstream_tries,
                              prev->upstream.next_upstream_tries, 0);

    ngx_conf_merge_value(conf->upstream.buffering,
                              prev->upstream.buffering, 1);

    ngx_conf_merge_value(conf->upstream.ignore_client_abort,
                              prev->upstream.ignore_client_abort, 0);

    ngx_conf_merge_value(conf->upstream.force_ranges,
                              prev->upstream.force_ranges, 0);

    ngx_conf_merge_ptr_value(conf->upstream.local,
                              prev->upstream.local, NULL);

    ngx_conf_merge_msec_value(conf->upstream.connect_timeout,
                              prev->upstream.connect_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.send_timeout,
                              prev->upstream.send_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.read_timeout,
                              prev->upstream.read_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.next_upstream_timeout,
                              prev->upstream.next_upstream_timeout, 0);

    ngx_conf_merge_size_value(conf->upstream.send_lowat,
                              prev->upstream.send_lowat, 0);

    ngx_conf_merge_size_value(conf->upstream.buffer_size,
                              prev->upstream.buffer_size,
                              (size_t) ngx_pagesize);

    ngx_conf_merge_size_value(conf->upstream.limit_rate,
                              prev->upstream.limit_rate, 0);


    ngx_conf_merge_bufs_value(conf->upstream.bufs, prev->upstream.bufs,
                              8, ngx_pagesize);
```

```
1287
1288      if (conf->upstream.bufs.num < 2) {
1289          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1290                             "there must be at least 2 \"scgi_buffers\"");
1291          return NGX_CONF_ERROR;
1292      }
1293
1294
1295      size = conf->upstream.buffer_size;
1296      if (size < conf->upstream.bufs.size) {
1297          size = conf->upstream.bufs.size;
1298      }
1299
1300
1301      ngx_conf_merge_size_value(conf->upstream.busy_buffers_size_conf,
1302                                prev->upstream.busy_buffers_size_conf,
1303                                NGX_CONF_UNSET_SIZE);
1304
1305      if (conf->upstream.busy_buffers_size_conf == NGX_CONF_UNSET_SIZE) {
1306          conf->upstream.busy_buffers_size = 2 * size;
1307      } else {
1308          conf->upstream.busy_buffers_size =
1309              conf->upstream.busy_buffers_size_conf;
1310      }
1311
1312      if (conf->upstream.busy_buffers_size < size) {
1313          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1314              "\"scgi_busy_buffers_size\" must be equal to or greater "
1315              "than the maximum of the value of \"scgi_buffer_size\" and "
1316              "one of the \"scgi_buffers\"");
1317
1318          return NGX_CONF_ERROR;
1319      }
1320
1321      if (conf->upstream.busy_buffers_size
1322          > (conf->upstream.bufs.num - 1) * conf->upstream.bufs.size)
1323      {
1324          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1325              "\"scgi_busy_buffers_size\" must be less than "
1326              "the size of all \"scgi_buffers\" minus one buffer");
1327
1328          return NGX_CONF_ERROR;
1329      }
1330
1331
1332      ngx_conf_merge_size_value(conf->upstream.temp_file_write_size_conf,
1333                                prev->upstream.temp_file_write_size_conf,
1334                                NGX_CONF_UNSET_SIZE);
1335
1336      if (conf->upstream.temp_file_write_size_conf == NGX_CONF_UNSET_SIZE) {
1337          conf->upstream.temp_file_write_size = 2 * size;
1338      } else {
1339          conf->upstream.temp_file_write_size =
1340              conf->upstream.temp_file_write_size_conf;
1341      }
1342
1343      if (conf->upstream.temp_file_write_size < size) {
1344          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1345              "\"scgi_temp_file_write_size\" must be equal to or greater than "
1346              "the maximum of the value of \"scgi_buffer_size\" and "
1347              "one of the \"scgi_buffers\"");
1348
1349          return NGX_CONF_ERROR;
1350      }
1351
1352
1353      ngx_conf_merge_size_value(conf->upstream.max_temp_file_size_conf,
1354                                prev->upstream.max_temp_file_size_conf,
1355                                NGX_CONF_UNSET_SIZE);
1356
1357      if (conf->upstream.max_temp_file_size_conf == NGX_CONF_UNSET_SIZE) {
1358          conf->upstream.max_temp_file_size = 1024 * 1024 * 1024;
1359      } else {
1360          conf->upstream.max_temp_file_size =
1361              conf->upstream.max_temp_file_size_conf;
1362      }
```

```
1363
1364        if (conf->upstream.max_temp_file_size != 0
1365            && conf->upstream.max_temp_file_size < size)
1366        {
1367            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1368                "\"scgi_max_temp_file_size\" must be equal to zero to disable "
1369                "temporary files usage or must be equal to or greater than "
1370                "the maximum of the value of \"scgi_buffer_size\" and "
1371                "one of the \"scgi_buffers\"");
1372
1373            return NGX_CONF_ERROR;
1374        }
1375
1376
1377        ngx_conf_merge_bitmask_value(conf->upstream.ignore_headers,
1378                                     prev->upstream.ignore_headers,
1379                                     NGX_CONF_BITMASK_SET);
1380
1381
1382        ngx_conf_merge_bitmask_value(conf->upstream.next_upstream,
1383                                     prev->upstream.next_upstream,
1384                                     (NGX_CONF_BITMASK_SET
1385                                      |NGX_HTTP_UPSTREAM_FT_ERROR
1386                                      |NGX_HTTP_UPSTREAM_FT_TIMEOUT));
1387
1388        if (conf->upstream.next_upstream & NGX_HTTP_UPSTREAM_FT_OFF) {
1389            conf->upstream.next_upstream = NGX_CONF_BITMASK_SET
1390                                           |NGX_HTTP_UPSTREAM_FT_OFF;
1391        }
1392
1393        if (ngx_conf_merge_path_value(cf, &conf->upstream.temp_path,
1394                                      prev->upstream.temp_path,
1395                                      &ngx_http_scgi_temp_path)
1396            != NGX_OK)
1397        {
1398            return NGX_CONF_ERROR;
1399        }
1400
1401    #if (NGX_HTTP_CACHE)
1402
1403        if (conf->upstream.cache == NGX_CONF_UNSET) {
1404            ngx_conf_merge_value(conf->upstream.cache,
1405                                 prev->upstream.cache, 0);
1406
1407            conf->upstream.cache_zone = prev->upstream.cache_zone;
1408            conf->upstream.cache_value = prev->upstream.cache_value;
1409        }
1410
1411        if (conf->upstream.cache_zone && conf->upstream.cache_zone->data == NULL) {
1412            ngx_shm_zone_t  *shm_zone;
1413
1414            shm_zone = conf->upstream.cache_zone;
1415
1416            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1417                               "\"scgi_cache\" zone \"%V\" is unknown",
1418                               &shm_zone->shm.name);
1419
1420            return NGX_CONF_ERROR;
1421        }
1422
1423        ngx_conf_merge_uint_value(conf->upstream.cache_min_uses,
1424                                  prev->upstream.cache_min_uses, 1);
1425
1426        ngx_conf_merge_bitmask_value(conf->upstream.cache_use_stale,
1427                                     prev->upstream.cache_use_stale,
1428                                     (NGX_CONF_BITMASK_SET
1429                                      |NGX_HTTP_UPSTREAM_FT_OFF));
1430
1431        if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_OFF) {
1432            conf->upstream.cache_use_stale = NGX_CONF_BITMASK_SET
1433                                             |NGX_HTTP_UPSTREAM_FT_OFF;
1434        }
1435
1436        if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_ERROR) {
1437            conf->upstream.cache_use_stale |= NGX_HTTP_UPSTREAM_FT_NOLIVE;
1438        }
```

```
1439
1440      if (conf->upstream.cache_methods == 0) {
1441          conf->upstream.cache_methods = prev->upstream.cache_methods;
1442      }
1443
1444      conf->upstream.cache_methods |= NGX_HTTP_GET|NGX_HTTP_HEAD;
1445
1446      ngx_conf_merge_ptr_value(conf->upstream.cache_bypass,
1447                               prev->upstream.cache_bypass, NULL);
1448
1449      ngx_conf_merge_ptr_value(conf->upstream.no_cache,
1450                               prev->upstream.no_cache, NULL);
1451
1452      ngx_conf_merge_ptr_value(conf->upstream.cache_valid,
1453                               prev->upstream.cache_valid, NULL);
1454
1455      if (conf->cache_key.value.data == NULL) {
1456          conf->cache_key = prev->cache_key;
1457      }
1458
1459      if (conf->upstream.cache && conf->cache_key.value.data == NULL) {
1460          ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1461                             "no \"scgi_cache_key\" for \"scgi_cache\"");
1462      }
1463
1464      ngx_conf_merge_value(conf->upstream.cache_lock,
1465                               prev->upstream.cache_lock, 0);
1466
1467      ngx_conf_merge_msec_value(conf->upstream.cache_lock_timeout,
1468                               prev->upstream.cache_lock_timeout, 5000);
1469
1470      ngx_conf_merge_msec_value(conf->upstream.cache_lock_age,
1471                               prev->upstream.cache_lock_age, 5000);
1472
1473      ngx_conf_merge_value(conf->upstream.cache_revalidate,
1474                               prev->upstream.cache_revalidate, 0);
1475
1476  #endif
1477
1478      ngx_conf_merge_value(conf->upstream.pass_request_headers,
1479                           prev->upstream.pass_request_headers, 1);
1480      ngx_conf_merge_value(conf->upstream.pass_request_body,
1481                           prev->upstream.pass_request_body, 1);
1482
1483      ngx_conf_merge_value(conf->upstream.intercept_errors,
1484                           prev->upstream.intercept_errors, 0);
1485
1486      hash.max_size = 512;
1487      hash.bucket_size = ngx_align(64, ngx_cacheline_size);
1488      hash.name = "scgi_hide_headers_hash";
1489
1490      if (ngx_http_upstream_hide_headers_hash(cf, &conf->upstream,
1491              &prev->upstream, ngx_http_scgi_hide_headers, &hash)
1492          != NGX_OK)
1493      {
1494          return NGX_CONF_ERROR;
1495      }
1496
1497      clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
1498
1499      if (clcf->noname
1500          && conf->upstream.upstream == NULL && conf->scgi_lengths == NULL)
1501      {
1502          conf->upstream.upstream = prev->upstream.upstream;
1503          conf->scgi_lengths = prev->scgi_lengths;
1504          conf->scgi_values = prev->scgi_values;
1505      }
1506
1507      if (clcf->lmt_excpt && clcf->handler == NULL
1508          && (conf->upstream.upstream || conf->scgi_lengths))
1509      {
1510          clcf->handler = ngx_http_scgi_handler;
1511      }
1512
1513      if (conf->params_source == NULL) {
1514          conf->params = prev->params;
```

```
#if (NGX_HTTP_CACHE)
        conf->params_cache = prev->params_cache;
#endif
        conf->params_source = prev->params_source;
    }

    rc = ngx_http_scgi_init_params(cf, conf, &conf->params, NULL);
    if (rc != NGX_OK) {
        return NGX_CONF_ERROR;
    }

#if (NGX_HTTP_CACHE)

    if (conf->upstream.cache) {
        rc = ngx_http_scgi_init_params(cf, conf, &conf->params_cache,
                                       ngx_http_scgi_cache_headers);
        if (rc != NGX_OK) {
            return NGX_CONF_ERROR;
        }
    }

#endif

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_scgi_init_params(ngx_conf_t *cf, ngx_http_scgi_loc_conf_t *conf,
    ngx_http_scgi_params_t *params, ngx_keyval_t *default_params)
{
    u_char                     *p;
    size_t                      size;
    uintptr_t                  *code;
    ngx_uint_t                  i, nsrc;
    ngx_array_t                 headers_names, params_merged;
    ngx_keyval_t               *h;
    ngx_hash_key_t             *hk;
    ngx_hash_init_t             hash;
    ngx_http_upstream_param_t   *src, *s;
    ngx_http_script_compile_t   sc;
    ngx_http_script_copy_code_t  *copy;

    if (params->hash.buckets) {
        return NGX_OK;
    }

    if (conf->params_source == NULL && default_params == NULL) {
        params->hash.buckets = (void *) 1;
        return NGX_OK;
    }

    params->lengths = ngx_array_create(cf->pool, 64, 1);
    if (params->lengths == NULL) {
        return NGX_ERROR;
    }

    params->values = ngx_array_create(cf->pool, 512, 1);
    if (params->values == NULL) {
        return NGX_ERROR;
    }

    if (ngx_array_init(&headers_names, cf->temp_pool, 4, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (conf->params_source) {
        src = conf->params_source->elts;
        nsrc = conf->params_source->nelts;

    } else {
        src = NULL;
        nsrc = 0;
    }
```

```
1591
1592        if (default_params) {
1593            if (ngx_array_init(&params_merged, cf->temp_pool, 4,
1594                               sizeof(ngx_http_upstream_param_t))
1595                != NGX_OK)
1596            {
1597                return NGX_ERROR;
1598            }
1599
1600            for (i = 0; i < nsrc; i++) {
1601
1602                s = ngx_array_push(&params_merged);
1603                if (s == NULL) {
1604                    return NGX_ERROR;
1605                }
1606
1607                *s = src[i];
1608            }
1609
1610            h = default_params;
1611
1612            while (h->key.len) {
1613
1614                src = params_merged.elts;
1615                nsrc = params_merged.nelts;
1616
1617                for (i = 0; i < nsrc; i++) {
1618                    if (ngx_strcasecmp(h->key.data, src[i].key.data) == 0) {
1619                        goto next;
1620                    }
1621                }
1622
1623                s = ngx_array_push(&params_merged);
1624                if (s == NULL) {
1625                    return NGX_ERROR;
1626                }
1627
1628                s->key = h->key;
1629                s->value = h->value;
1630                s->skip_empty = 1;
1631
1632            next:
1633
1634                h++;
1635            }
1636
1637            src = params_merged.elts;
1638            nsrc = params_merged.nelts;
1639        }
1640
1641        for (i = 0; i < nsrc; i++) {
1642
1643            if (src[i].key.len > sizeof("HTTP_") - 1
1644                && ngx_strncmp(src[i].key.data, "HTTP_", sizeof("HTTP_") - 1) == 0)
1645            {
1646                hk = ngx_array_push(&headers_names);
1647                if (hk == NULL) {
1648                    return NGX_ERROR;
1649                }
1650
1651                hk->key.len = src[i].key.len - 5;
1652                hk->key.data = src[i].key.data + 5;
1653                hk->key_hash = ngx_hash_key_lc(hk->key.data, hk->key.len);
1654                hk->value = (void *) 1;
1655
1656                if (src[i].value.len == 0) {
1657                    continue;
1658                }
1659            }
1660
1661            copy = ngx_array_push_n(params->lengths,
1662                                    sizeof(ngx_http_script_copy_code_t));
1663            if (copy == NULL) {
1664                return NGX_ERROR;
1665            }
1666
```

```
        copy->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
        copy->len = src[i].key.len + 1;

        copy = ngx_array_push_n(params->lengths,
                                sizeof(ngx_http_script_copy_code_t));
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
        copy->len = src[i].skip_empty;


        size = (sizeof(ngx_http_script_copy_code_t)
                + src[i].key.len + 1 + sizeof(uintptr_t) - 1)
               & ~(sizeof(uintptr_t) - 1);

        copy = ngx_array_push_n(params->values, size);
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = ngx_http_script_copy_code;
        copy->len = src[i].key.len + 1;

        p = (u_char *) copy + sizeof(ngx_http_script_copy_code_t);
        (void) ngx_cpystrn(p, src[i].key.data, src[i].key.len + 1);


        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

        sc.cf = cf;
        sc.source = &src[i].value;
        sc.flushes = &params->flushes;
        sc.lengths = &params->lengths;
        sc.values = &params->values;

        if (ngx_http_script_compile(&sc) != NGX_OK) {
            return NGX_ERROR;
        }

        code = ngx_array_push_n(params->lengths, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;


        code = ngx_array_push_n(params->values, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;
    }

    code = ngx_array_push_n(params->lengths, sizeof(uintptr_t));
    if (code == NULL) {
        return NGX_ERROR;
    }

    *code = (uintptr_t) NULL;

    params->number = headers_names.nelts;

    hash.hash = &params->hash;
    hash.key = ngx_hash_key_lc;
    hash.max_size = 512;
    hash.bucket_size = 64;
    hash.name = "scgi_params_hash";
    hash.pool = cf->pool;
    hash.temp_pool = NULL;

    return ngx_hash_init(&hash, headers_names.elts, headers_names.nelts);
}
```

```
1743
1744
1745 static char *
1746 ngx_http_scgi_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1747 {
1748     ngx_http_scgi_loc_conf_t *scf = conf;
1749
1750     ngx_url_t                  u;
1751     ngx_str_t                 *value, *url;
1752     ngx_uint_t                 n;
1753     ngx_http_core_loc_conf_t  *clcf;
1754     ngx_http_script_compile_t  sc;
1755
1756     if (scf->upstream.upstream || scf->scgi_lengths) {
1757         return "is duplicate";
1758     }
1759
1760     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
1761     clcf->handler = ngx_http_scgi_handler;
1762
1763     value = cf->args->elts;
1764
1765     url = &value[1];
1766
1767     n = ngx_http_script_variables_count(url);
1768
1769     if (n) {
1770
1771         ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
1772
1773         sc.cf = cf;
1774         sc.source = url;
1775         sc.lengths = &scf->scgi_lengths;
1776         sc.values = &scf->scgi_values;
1777         sc.variables = n;
1778         sc.complete_lengths = 1;
1779         sc.complete_values = 1;
1780
1781         if (ngx_http_script_compile(&sc) != NGX_OK) {
1782             return NGX_CONF_ERROR;
1783         }
1784
1785         return NGX_CONF_OK;
1786     }
1787
1788     ngx_memzero(&u, sizeof(ngx_url_t));
1789
1790     u.url = value[1];
1791     u.no_resolve = 1;
1792
1793     scf->upstream.upstream = ngx_http_upstream_add(cf, &u, 0);
1794     if (scf->upstream.upstream == NULL) {
1795         return NGX_CONF_ERROR;
1796     }
1797
1798     if (clcf->name.data[clcf->name.len - 1] == '/') {
1799         clcf->auto_redirect = 1;
1800     }
1801
1802     return NGX_CONF_OK;
1803 }
1804
1805
1806 static char *
1807 ngx_http_scgi_store(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1808 {
1809     ngx_http_scgi_loc_conf_t *scf = conf;
1810
1811     ngx_str_t                 *value;
1812     ngx_http_script_compile_t  sc;
1813
1814     if (scf->upstream.store != NGX_CONF_UNSET) {
1815         return "is duplicate";
1816     }
1817
1818     value = cf->args->elts;
```

```
1819
1820        if (ngx_strcmp(value[1].data, "off") == 0) {
1821            scf->upstream.store = 0;
1822            return NGX_CONF_OK;
1823        }
1824
1825    #if (NGX_HTTP_CACHE)
1826        if (scf->upstream.cache > 0) {
1827            return "is incompatible with \"scgi_cache\"";
1828        }
1829    #endif
1830
1831        scf->upstream.store = 1;
1832
1833        if (ngx_strcmp(value[1].data, "on") == 0) {
1834            return NGX_CONF_OK;
1835        }
1836
1837        /* include the terminating '\0' into script */
1838        value[1].len++;
1839
1840        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));
1841
1842        sc.cf = cf;
1843        sc.source = &value[1];
1844        sc.lengths = &scf->upstream.store_lengths;
1845        sc.values = &scf->upstream.store_values;
1846        sc.variables = ngx_http_script_variables_count(&value[1]);
1847        sc.complete_lengths = 1;
1848        sc.complete_values = 1;
1849
1850        if (ngx_http_script_compile(&sc) != NGX_OK) {
1851            return NGX_CONF_ERROR;
1852        }
1853
1854        return NGX_CONF_OK;
1855    }
1856
1857
1858    #if (NGX_HTTP_CACHE)
1859
1860    static char *
1861    ngx_http_scgi_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1862    {
1863        ngx_http_scgi_loc_conf_t *scf = conf;
1864
1865        ngx_str_t                         *value;
1866        ngx_http_complex_value_t           cv;
1867        ngx_http_compile_complex_value_t   ccv;
1868
1869        value = cf->args->elts;
1870
1871        if (scf->upstream.cache != NGX_CONF_UNSET) {
1872            return "is duplicate";
1873        }
1874
1875        if (ngx_strcmp(value[1].data, "off") == 0) {
1876            scf->upstream.cache = 0;
1877            return NGX_CONF_OK;
1878        }
1879
1880        if (scf->upstream.store > 0) {
1881            return "is incompatible with \"scgi_store\"";
1882        }
1883
1884        scf->upstream.cache = 1;
1885
1886        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
1887
1888        ccv.cf = cf;
1889        ccv.value = &value[1];
1890        ccv.complex_value = &cv;
1891
1892        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
1893            return NGX_CONF_ERROR;
1894        }
```

```
1895
1896        if (cv.lengths != NULL) {
1897
1898            scf->upstream.cache_value = ngx_palloc(cf->pool,
1899                                                   sizeof(ngx_http_complex_value_t));
1900            if (scf->upstream.cache_value == NULL) {
1901                return NGX_CONF_ERROR;
1902            }
1903
1904            *scf->upstream.cache_value = cv;
1905
1906            return NGX_CONF_OK;
1907        }
1908
1909        scf->upstream.cache_zone = ngx_shared_memory_add(cf, &value[1], 0,
1910                                                         &ngx_http_scgi_module);
1911        if (scf->upstream.cache_zone == NULL) {
1912            return NGX_CONF_ERROR;
1913        }
1914
1915        return NGX_CONF_OK;
1916    }
1917
1918
1919    static char *
1920    ngx_http_scgi_cache_key(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
1921    {
1922        ngx_http_scgi_loc_conf_t *scf = conf;
1923
1924        ngx_str_t                         *value;
1925        ngx_http_compile_complex_value_t   ccv;
1926
1927        value = cf->args->elts;
1928
1929        if (scf->cache_key.value.data) {
1930            return "is duplicate";
1931        }
1932
1933        ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
1934
1935        ccv.cf = cf;
1936        ccv.value = &value[1];
1937        ccv.complex_value = &scf->cache_key;
1938
1939        if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
1940            return NGX_CONF_ERROR;
1941        }
1942
1943        return NGX_CONF_OK;
1944    }
1945
1946    #endif
```

One Level Up Top Level

src/http/modules/ngx_http_secure_link_module.c - nginx-1.7.10

Global variables defined

- ngx_http_secure_link_commands
- ngx_http_secure_link_expires_name
- ngx_http_secure_link_module
- ngx_http_secure_link_module_ctx
- ngx_http_secure_link_name

Data types defined

- ngx_http_secure_link_conf_t
- ngx_http_secure_link_ctx_t

Functions defined

- ngx_http_secure_link_add_variables
- ngx_http_secure_link_create_conf
- ngx_http_secure_link_expires_variable
- ngx_http_secure_link_merge_conf
- ngx_http_secure_link_old_variable
- ngx_http_secure_link_variable

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11  #include <ngx_md5.h>
12
13
14  typedef struct {
15      ngx_http_complex_value_t  *variable;
16      ngx_http_complex_value_t  *md5;
17      ngx_str_t                  secret;
18  } ngx_http_secure_link_conf_t;
19
20
21  typedef struct {
22      ngx_str_t                  expires;
23  } ngx_http_secure_link_ctx_t;
24
25
26  static ngx_int_t ngx_http_secure_link_old_variable(ngx_http_request_t *r,
27      ngx_http_secure_link_conf_t *conf, ngx_http_variable_value_t *v,
28      uintptr_t data);
```

```
static ngx_int_t ngx_http_secure_link_expires_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data);
static void *ngx_http_secure_link_create_conf(ngx_conf_t *cf);
static char *ngx_http_secure_link_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);
static ngx_int_t ngx_http_secure_link_add_variables(ngx_conf_t *cf);


static ngx_command_t  ngx_http_secure_link_commands[] = {

    { ngx_string("secure_link"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_set_complex_value_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_secure_link_conf_t, variable),
      NULL },

    { ngx_string("secure_link_md5"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_set_complex_value_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_secure_link_conf_t, md5),
      NULL },

    { ngx_string("secure_link_secret"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_secure_link_conf_t, secret),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_secure_link_module_ctx = {
    ngx_http_secure_link_add_variables,    /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_secure_link_create_conf,      /* create location configuration */
    ngx_http_secure_link_merge_conf        /* merge location configuration */
};


ngx_module_t  ngx_http_secure_link_module = {
    NGX_MODULE_V1,
    &ngx_http_secure_link_module_ctx,      /* module context */
    ngx_http_secure_link_commands,         /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t  ngx_http_secure_link_name = ngx_string("secure_link");
static ngx_str_t  ngx_http_secure_link_expires_name =
    ngx_string("secure_link_expires");


 static ngx_int_t
 ngx_http_secure_link_variable(ngx_http_request_t *r,
     ngx_http_variable_value_t *v, uintptr_t data)
 {
     u_char                     *p, *last;
```

```
    ngx_str_t                    val, hash;
    time_t                       expires;
    ngx_md5_t                    md5;
    ngx_http_secure_link_ctx_t   *ctx;
    ngx_http_secure_link_conf_t  *conf;
    u_char                       hash_buf[16], md5_buf[16];

    conf = ngx_http_get_module_loc_conf(r, ngx_http_secure_link_module);

    if (conf->secret.data) {
        return ngx_http_secure_link_old_variable(r, conf, v, data);
    }

    if (conf->variable == NULL || conf->md5 == NULL) {
        goto not_found;
    }

    if (ngx_http_complex_value(r, conf->variable, &val) != NGX_OK) {
        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "secure link: \"%V\"", &val);

    last = val.data + val.len;

    p = ngx_strlchr(val.data, last, ',');
    expires = 0;

    if (p) {
        val.len = p++ - val.data;

        expires = ngx_atotm(p, last - p);
        if (expires <= 0) {
            goto not_found;
        }

        ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_secure_link_ctx_t));
        if (ctx == NULL) {
            return NGX_ERROR;
        }

        ngx_http_set_ctx(r, ctx, ngx_http_secure_link_module);

        ctx->expires.len = last - p;
        ctx->expires.data = p;
    }

    if (val.len > 24) {
        goto not_found;
    }

    hash.len = 16;
    hash.data = hash_buf;

    if (ngx_decode_base64url(&hash, &val) != NGX_OK) {
        goto not_found;
    }

    if (hash.len != 16) {
        goto not_found;
    }

    if (ngx_http_complex_value(r, conf->md5, &val) != NGX_OK) {
        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "secure link md5: \"%V\"", &val);

    ngx_md5_init(&md5);
    ngx_md5_update(&md5, val.data, val.len);
    ngx_md5_final(md5_buf, &md5);

    if (ngx_memcmp(hash_buf, md5_buf, 16) != 0) {
        goto not_found;
```

```
    }

    v->data = (u_char *) ((expires && expires < ngx_time()) ? "0" : "1");
    v->len = 1;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;

    return NGX_OK;

not_found:

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_secure_link_old_variable(ngx_http_request_t *r,
    ngx_http_secure_link_conf_t *conf, ngx_http_variable_value_t *v,
    uintptr_t data)
{
    u_char      *p, *start, *end, *last;
    size_t       len;
    ngx_int_t    n;
    ngx_uint_t   i;
    ngx_md5_t    md5;
    u_char       hash[16];

    p = &r->unparsed_uri.data[1];
    last = r->unparsed_uri.data + r->unparsed_uri.len;

    while (p < last) {
        if (*p++ == '/') {
            start = p;
            goto md5_start;
        }
    }

    goto not_found;

md5_start:

    while (p < last) {
        if (*p++ == '/') {
            end = p - 1;
            goto url_start;
        }
    }

    goto not_found;

url_start:

    len = last - p;

    if (end - start != 32 || len == 0) {
        goto not_found;
    }

    ngx_md5_init(&md5);
    ngx_md5_update(&md5, p, len);
    ngx_md5_update(&md5, conf->secret.data, conf->secret.len);
    ngx_md5_final(hash, &md5);

    for (i = 0; i < 16; i++) {
        n = ngx_hextoi(&start[2 * i], 2);
        if (n == NGX_ERROR || n != hash[i]) {
            goto not_found;
        }
    }

    v->len = len;
    v->valid = 1;
    v->no_cacheable = 0;
```

```
    v->not_found = 0;
    v->data = p;

    return NGX_OK;

not_found:

    v->not_found = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_secure_link_expires_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_http_secure_link_ctx_t  *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_secure_link_module);

    if (ctx) {
        v->len = ctx->expires.len;
        v->valid = 1;
        v->no_cacheable = 0;
        v->not_found = 0;
        v->data = ctx->expires.data;

    } else {
        v->not_found = 1;
    }

    return NGX_OK;
}


static void *
ngx_http_secure_link_create_conf(ngx_conf_t *cf)
{
    ngx_http_secure_link_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_secure_link_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->variable = NULL;
     *     conf->md5 = NULL;
     *     conf->secret = { 0, NULL };
     */

    return conf;
}


static char *
ngx_http_secure_link_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_secure_link_conf_t *prev = parent;
    ngx_http_secure_link_conf_t *conf = child;

    if (conf->secret.data) {
        if (conf->variable || conf->md5) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "\"secure_link_secret\" cannot be mixed with "
                               "\"secure_link\" and \"secure_link_md5\"");
            return NGX_CONF_ERROR;
        }

        return NGX_CONF_OK;
    }

    if (conf->variable == NULL) {
```

```
333         conf->variable = prev->variable;
334     }
335
336     if (conf->md5 == NULL) {
337         conf->md5 = prev->md5;
338     }
339
340     if (conf->variable == NULL && conf->md5 == NULL) {
341         conf->secret = prev->secret;
342     }
343
344     return NGX_CONF_OK;
345 }
346
347
348 static ngx_int_t
349 ngx_http_secure_link_add_variables(ngx_conf_t *cf)
350 {
351     ngx_http_variable_t  *var;
352
353     var = ngx_http_add_variable(cf, &ngx_http_secure_link_name, 0);
354     if (var == NULL) {
355         return NGX_ERROR;
356     }
357
358     var->get_handler = ngx_http_secure_link_variable;
359
360     var = ngx_http_add_variable(cf, &ngx_http_secure_link_expires_name, 0);
361     if (var == NULL) {
362         return NGX_ERROR;
363     }
364
365     var->get_handler = ngx_http_secure_link_expires_variable;
366
367     return NGX_OK;
368 }
```

One Level Up Top Level

src/http/modules/ngx_http_split_clients_module.c - nginx-1.7.10

Global variables defined

- ngx_http_split_clients_commands
- ngx_http_split_clients_module
- ngx_http_split_clients_module_ctx

Data types defined

- ngx_http_split_clients_ctx_t
- ngx_http_split_clients_part_t

Functions defined

- ngx_conf_split_clients_block
- ngx_http_split_clients
- ngx_http_split_clients_variable

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      uint32_t                    percent;
15      ngx_http_variable_value_t   value;
16  } ngx_http_split_clients_part_t;
17
18
19  typedef struct {
20      ngx_http_complex_value_t    value;
21      ngx_array_t                 parts;
22  } ngx_http_split_clients_ctx_t;
23
24
25  static char *ngx_conf_split_clients_block(ngx_conf_t *cf, ngx_command_t *cmd,
26      void *conf);
27  static char *ngx_http_split_clients(ngx_conf_t *cf, ngx_command_t *dummy,
28      void *conf);
29
30  static ngx_command_t  ngx_http_split_clients_commands[] = {
31
32      { ngx_string("split_clients"),
33        NGX_HTTP_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_TAKE2,
34        ngx_conf_split_clients_block,
35        NGX_HTTP_MAIN_CONF_OFFSET,
36        0,
37        NULL },
38
39        ngx_null_command
```

```
};


static ngx_http_module_t  ngx_http_split_clients_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_split_clients_module = {
    NGX_MODULE_V1,
    &ngx_http_split_clients_module_ctx,    /* module context */
    ngx_http_split_clients_commands,       /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_split_clients_variable(ngx_http_request_t *r,
    ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_http_split_clients_ctx_t *ctx = (ngx_http_split_clients_ctx_t *) data;

    uint32_t                      hash;
    ngx_str_t                     val;
    ngx_uint_t                    i;
    ngx_http_split_clients_part_t  *part;

    *v = ngx_http_variable_null_value;

    if (ngx_http_complex_value(r, &ctx->value, &val) != NGX_OK) {
        return NGX_OK;
    }

    hash = ngx_murmur_hash2(val.data, val.len);

    part = ctx->parts.elts;

    for (i = 0; i < ctx->parts.nelts; i++) {

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http split: %uD %uD", hash, part[i].percent);

         if (hash < part[i].percent || part[i].percent == 0) {
             *v = part[i].value;
             return NGX_OK;
         }
     }

     return NGX_OK;
 }


 static char *
 ngx_conf_split_clients_block(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
 {
     char                             *rv;
     uint32_t                          sum, last;
     ngx_str_t                        *value, name;
```

```
    ngx_uint_t                          i;
    ngx_conf_t                          save;
    ngx_http_variable_t                *var;
    ngx_http_split_clients_ctx_t       *ctx;
    ngx_http_split_clients_part_t      *part;
    ngx_http_compile_complex_value_t    ccv;

    ctx = ngx_pcalloc(cf->pool, sizeof(ngx_http_split_clients_ctx_t));
    if (ctx == NULL) {
        return NGX_CONF_ERROR;
    }

    value = cf->args->elts;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &ctx->value;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    name = value[2];

    if (name.data[0] != '$') {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid variable name \"%V\"", &name);
        return NGX_CONF_ERROR;
    }

    name.len--;
    name.data++;

    var = ngx_http_add_variable(cf, &name, NGX_HTTP_VAR_CHANGEABLE);
    if (var == NULL) {
        return NGX_CONF_ERROR;
    }

    var->get_handler = ngx_http_split_clients_variable;
    var->data = (uintptr_t) ctx;

    if (ngx_array_init(&ctx->parts, cf->pool, 2,
                       sizeof(ngx_http_split_clients_part_t))
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    save = *cf;
    cf->ctx = ctx;
    cf->handler = ngx_http_split_clients;
    cf->handler_conf = conf;

    rv = ngx_conf_parse(cf, NULL);

    *cf = save;

    if (rv != NGX_CONF_OK) {
        return rv;
    }

    sum = 0;
    last = 0;
    part = ctx->parts.elts;

    for (i = 0; i < ctx->parts.nelts; i++) {
        sum = part[i].percent ? sum + part[i].percent : 10000;
        if (sum > 10000) {
            ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                               "percent total is greater than 100%%");
            return NGX_CONF_ERROR;
        }

        if (part[i].percent) {
```

```
192             last += part[i].percent * (uint64_t) 0xffffffff / 10000;
193             part[i].percent = last;
194         }
195     }
196
197     return rv;
198 }
199
200
201 static char *
202 ngx_http_split_clients(ngx_conf_t *cf, ngx_command_t *dummy, void *conf)
203 {
204     ngx_int_t                       n;
205     ngx_str_t                      *value;
206     ngx_http_split_clients_ctx_t   *ctx;
207     ngx_http_split_clients_part_t  *part;
208
209     ctx = cf->ctx;
210     value = cf->args->elts;
211
212     part = ngx_array_push(&ctx->parts);
213     if (part == NULL) {
214         return NGX_CONF_ERROR;
215     }
216
217     if (value[0].len == 1 && value[0].data[0] == '*') {
218         part->percent = 0;
219
220     } else {
221         if (value[0].len == 0 || value[0].data[value[0].len - 1] != '%') {
222             goto invalid;
223         }
224
225         n = ngx_atofp(value[0].data, value[0].len - 1, 2);
226         if (n == NGX_ERROR || n == 0) {
227             goto invalid;
228         }
229
230         part->percent = (uint32_t) n;
231     }
232
233     part->value.len = value[1].len;
234     part->value.valid = 1;
235     part->value.no_cacheable = 0;
236     part->value.not_found = 0;
237     part->value.data = value[1].data;
238
239     return NGX_CONF_OK;
240
241 invalid:
242
243     ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
244                        "invalid percent value \"%V\"", &value[0]);
245     return NGX_CONF_ERROR;
246 }
```

One Level Up Top Level

One Level Up Top Level

src/core/ngx_murmurhash.c - nginx-1.7.10

Functions defined

- ngx_murmur_hash2

Source code

```
1
2  /*
3  * Copyright (C) Austin Appleby
4  */
5
6
7  #include <ngx_config.h>
8  #include <ngx_core.h>
9
10
11  uint32_t
12  ngx_murmur_hash2(u_char *data, size_t len)
13  {
14      uint32_t  h, k;
15
16      h = 0 ^ len;
17
18      while (len >= 4) {
19          k  = data[0];
20          k |= data[1] << 8;
21          k |= data[2] << 16;
22          k |= data[3] << 24;
23
24          k *= 0x5bd1e995;
25          k ^= k >> 24;
26          k *= 0x5bd1e995;
27
28          h *= 0x5bd1e995;
29          h ^= k;
30
31          data += 4;
32          len -= 4;
33      }
34
35      switch (len) {
36      case 3:
37          h ^= data[2] << 16;
38      case 2:
39          h ^= data[1] << 8;
40      case 1:
41          h ^= data[0];
42          h *= 0x5bd1e995;
43      }
44
45      h ^= h >> 13;
46      h *= 0x5bd1e995;
47      h ^= h >> 15;
48
49      return h;
50  }
```

One Level Up Top Level

src/http/modules/ngx_http_static_module.c - nginx-1.7.10

Global variables defined

- ngx_http_static_module
- ngx_http_static_module_ctx

Functions defined

- ngx_http_static_handler
- ngx_http_static_init

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static ngx_int_t ngx_http_static_handler(ngx_http_request_t *r);
14  static ngx_int_t ngx_http_static_init(ngx_conf_t *cf);
15
16
17  ngx_http_module_t  ngx_http_static_module_ctx = {
18      NULL,                                  /* preconfiguration */
19      ngx_http_static_init,                  /* postconfiguration */
20
21      NULL,                                  /* create main configuration */
22      NULL,                                  /* init main configuration */
23
24      NULL,                                  /* create server configuration */
25      NULL,                                  /* merge server configuration */
26
27      NULL,                                  /* create location configuration */
28      NULL                                   /* merge location configuration */
29  };
30
31
32  ngx_module_t  ngx_http_static_module = {
33      NGX_MODULE_V1,
34      &ngx_http_static_module_ctx,           /* module context */
35      NULL,                                  /* module directives */
36      NGX_HTTP_MODULE,                       /* module type */
37      NULL,                                  /* init master */
38      NULL,                                  /* init module */
39      NULL,                                  /* init process */
40      NULL,                                  /* init thread */
41      NULL,                                  /* exit thread */
42      NULL,                                  /* exit process */
43      NULL,                                  /* exit master */
44      NGX_MODULE_V1_PADDING
45  };
46
47
48  static ngx_int_t
49  ngx_http_static_handler(ngx_http_request_t *r)
50  {
51      u_char                   *last, *location;
52      size_t                    root, len;
```

```
    ngx_str_t                  path;
    ngx_int_t                  rc;
    ngx_uint_t                 level;
    ngx_log_t                 *log;
    ngx_buf_t                 *b;
    ngx_chain_t                out;
    ngx_open_file_info_t       of;
    ngx_http_core_loc_conf_t  *clcf;

    if (!(r->method & (NGX_HTTP_GET|NGX_HTTP_HEAD|NGX_HTTP_POST))) {
        return NGX_HTTP_NOT_ALLOWED;
    }

    if (r->uri.data[r->uri.len - 1] == '/') {
        return NGX_DECLINED;
    }

    log = r->connection->log;

    /*
     * ngx_http_map_uri_to_path() allocates memory for terminating '\0'
     * so we do not need to reserve memory for '/' for possible redirect
     */

    last = ngx_http_map_uri_to_path(r, &path, &root, 0);
    if (last == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    path.len = last - path.data;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, log, 0,
                   "http filename: \"%s\"", path.data);

    clcf = ngx_http_get_module_loc_conf(r, ngx_http_core_module);

    ngx_memzero(&of, sizeof(ngx_open_file_info_t));

    of.read_ahead = clcf->read_ahead;
    of.directio = clcf->directio;
    of.valid = clcf->open_file_cache_valid;
    of.min_uses = clcf->open_file_cache_min_uses;
    of.errors = clcf->open_file_cache_errors;
    of.events = clcf->open_file_cache_events;

    if (ngx_http_set_disable_symlinks(r, clcf, &path, &of) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
     }

     if (ngx_open_cached_file(clcf->open_file_cache, &path, &of, r->pool)
         != NGX_OK)
     {
         switch (of.err) {

         case 0:
             return NGX_HTTP_INTERNAL_SERVER_ERROR;

         case NGX_ENOENT:
         case NGX_ENOTDIR:
         case NGX_ENAMETOOLONG:

             level = NGX_LOG_ERR;
             rc = NGX_HTTP_NOT_FOUND;
             break;

         case NGX_EACCES:
#if (NGX_HAVE_OPENAT)
         case NGX_EMLINK:
         case NGX_ELOOP:
#endif

             level = NGX_LOG_ERR;
             rc = NGX_HTTP_FORBIDDEN;
             break;

         default:
```

```
            level = NGX_LOG_CRIT;
            rc = NGX_HTTP_INTERNAL_SERVER_ERROR;
            break;
        }

        if (rc != NGX_HTTP_NOT_FOUND || clcf->log_not_found) {
            ngx_log_error(level, log, of.err,
                          "%s \"%s\" failed", of.failed, path.data);
        }

        return rc;
    }

    r->root_tested = !r->error_page;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, log, 0, "http static fd: %d", of.fd);

    if (of.is_dir) {

        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, log, 0, "http dir");

        ngx_http_clear_location(r);

        r->headers_out.location = ngx_palloc(r->pool, sizeof(ngx_table_elt_t));
        if (r->headers_out.location == NULL) {
            return NGX_HTTP_INTERNAL_SERVER_ERROR;
        }

        len = r->uri.len + 1;

        if (!clcf->alias && clcf->root_lengths == NULL && r->args.len == 0) {
            location = path.data + clcf->root.len;

            *last = '/';

        } else {
            if (r->args.len) {
                len += r->args.len + 1;
            }

            location = ngx_pnalloc(r->pool, len);
            if (location == NULL) {
                return NGX_HTTP_INTERNAL_SERVER_ERROR;
            }

            last = ngx_copy(location, r->uri.data, r->uri.len);

            *last = '/';

            if (r->args.len) {
                *++last = '?';
                ngx_memcpy(++last, r->args.data, r->args.len);
            }
        }

        /*
         * we do not need to set the r->headers_out.location->hash and
         * r->headers_out.location->key fields
         */

        r->headers_out.location->value.len = len;
        r->headers_out.location->value.data = location;

        return NGX_HTTP_MOVED_PERMANENTLY;
    }

#if !(NGX_WIN32) /* the not regular files are probably Unix specific */

    if (!of.is_file) {
        ngx_log_error(NGX_LOG_CRIT, log, 0,
                      "\"%s\" is not a regular file", path.data);

        return NGX_HTTP_NOT_FOUND;
    }

```

```
#endif

    if (r->method & NGX_HTTP_POST) {
        return NGX_HTTP_NOT_ALLOWED;
    }

    rc = ngx_http_discard_request_body(r);

    if (rc != NGX_OK) {
        return rc;
    }

    log->action = "sending response to client";

    r->headers_out.status = NGX_HTTP_OK;
    r->headers_out.content_length_n = of.size;
    r->headers_out.last_modified_time = of.mtime;

    if (ngx_http_set_etag(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (ngx_http_set_content_type(r) != NGX_OK) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    if (r != r->main && of.size == 0) {
        return ngx_http_send_header(r);
    }

    r->allow_ranges = 1;

    /* we need to allocate all before the header would be sent */

    b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
    if (b == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    b->file = ngx_pcalloc(r->pool, sizeof(ngx_file_t));
    if (b->file == NULL) {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    rc = ngx_http_send_header(r);

    if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
        return rc;
    }

    b->file_pos = 0;
    b->file_last = of.size;

    b->in_file = b->file_last ? 1: 0;
    b->last_buf = (r == r->main) ? 1: 0;
    b->last_in_chain = 1;

    b->file->fd = of.fd;
    b->file->name = path;
    b->file->log = log;
    b->file->directio = of.is_directio;

    out.buf = b;
    out.next = NULL;

    return ngx_http_output_filter(r, &out);
}


static ngx_int_t
ngx_http_static_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt        *h;
    ngx_http_core_main_conf_t  *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);
```

```
281
282     h = ngx_array_push(&cmcf->phases[NGX_HTTP_CONTENT_PHASE].handlers);
283     if (h == NULL) {
284         return NGX_ERROR;
285     }
286
287     *h = ngx_http_static_handler;
288
289     return NGX_OK;
290 }
```

One Level Up Top Level

src/http/modules/ngx_http_stub_status_module.c - nginx-1.7.10

Global variables defined

- ngx_http_status_commands
- ngx_http_stub_status_module
- ngx_http_stub_status_module_ctx
- ngx_http_stub_status_vars

Functions defined

- ngx_http_set_stub_status
- ngx_http_stub_status_add_variables
- ngx_http_stub_status_handler
- ngx_http_stub_status_variable

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  static ngx_int_t ngx_http_stub_status_handler(ngx_http_request_t *r);
14  static ngx_int_t ngx_http_stub_status_variable(ngx_http_request_t *r,
15      ngx_http_variable_value_t *v, uintptr_t data);
16  static ngx_int_t ngx_http_stub_status_add_variables(ngx_conf_t *cf);
17  static char *ngx_http_set_stub_status(ngx_conf_t *cf, ngx_command_t *cmd,
18      void *conf);
19
20
21  static ngx_command_t  ngx_http_status_commands[] = {
22
23      { ngx_string("stub_status"),
24        NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_NOARGS|NGX_CONF_TAKE1,
25        ngx_http_set_stub_status,
26        0,
27        0,
28        NULL },
29
30        ngx_null_command
31  };
32
33
34  static ngx_http_module_t  ngx_http_stub_status_module_ctx = {
35      ngx_http_stub_status_add_variables,    /* preconfiguration */
36      NULL,                                  /* postconfiguration */
37
38      NULL,                                  /* create main configuration */
39      NULL,                                  /* init main configuration */
40
41      NULL,                                  /* create server configuration */
42      NULL,                                  /* merge server configuration */
43
```

```
    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_stub_status_module = {
    NGX_MODULE_V1,
    &ngx_http_stub_status_module_ctx,      /* module context */
    ngx_http_status_commands,              /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_variable_t  ngx_http_stub_status_vars[] = {

    { ngx_string("connections_active"), NULL, ngx_http_stub_status_variable,
      0, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("connections_reading"), NULL, ngx_http_stub_status_variable,
      1, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("connections_writing"), NULL, ngx_http_stub_status_variable,
      2, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_string("connections_waiting"), NULL, ngx_http_stub_status_variable,
      3, NGX_HTTP_VAR_NOCACHEABLE, 0 },

    { ngx_null_string, NULL, NULL, 0, 0, 0 }
};


static ngx_int_t
ngx_http_stub_status_handler(ngx_http_request_t *r)
{
    size_t             size;
    ngx_int_t          rc;
    ngx_buf_t         *b;
    ngx_chain_t        out;
    ngx_atomic_int_t   ap, hn, ac, rq, rd, wr, wa;

    if (r->method != NGX_HTTP_GET && r->method != NGX_HTTP_HEAD) {
        return NGX_HTTP_NOT_ALLOWED;
    }

    rc = ngx_http_discard_request_body(r);

    if (rc != NGX_OK) {
        return rc;
     }

     r->headers_out.content_type_len = sizeof("text/plain") - 1;
     ngx_str_set(&r->headers_out.content_type, "text/plain");
     r->headers_out.content_type_lowcase = NULL;

     if (r->method == NGX_HTTP_HEAD) {
         r->headers_out.status = NGX_HTTP_OK;

         rc = ngx_http_send_header(r);

         if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
             return rc;
         }
     }

     size = sizeof("Active connections:  \n") + NGX_ATOMIC_T_LEN
            + sizeof("server accepts handled requests\n") - 1
            + 6 + 3 * NGX_ATOMIC_T_LEN
            + sizeof("Reading:  Writing:  Waiting:  \n") + 3 * NGX_ATOMIC_T_LEN;
```

```
120
121     b = ngx_create_temp_buf(r->pool, size);
122     if (b == NULL) {
123         return NGX_HTTP_INTERNAL_SERVER_ERROR;
124     }
125
126     out.buf = b;
127     out.next = NULL;
128
129     ap = *ngx_stat_accepted;
130     hn = *ngx_stat_handled;
131     ac = *ngx_stat_active;
132     rq = *ngx_stat_requests;
133     rd = *ngx_stat_reading;
134     wr = *ngx_stat_writing;
135     wa = *ngx_stat_waiting;
136
137     b->last = ngx_sprintf(b->last, "Active connections: %uA \n", ac);
138
139     b->last = ngx_cpymem(b->last, "server accepts handled requests\n",
140                          sizeof("server accepts handled requests\n") - 1);
141
142     b->last = ngx_sprintf(b->last, " %uA %uA %uA \n", ap, hn, rq);
143
144     b->last = ngx_sprintf(b->last, "Reading: %uA Writing: %uA Waiting: %uA \n",
145                           rd, wr, wa);
146
147     r->headers_out.status = NGX_HTTP_OK;
148     r->headers_out.content_length_n = b->last - b->pos;
149
150     b->last_buf = (r == r->main) ? 1 : 0;
151     b->last_in_chain = 1;
152
153     rc = ngx_http_send_header(r);
154
155     if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
156         return rc;
157     }
158
159     return ngx_http_output_filter(r, &out);
160 }
161
162
163 static ngx_int_t
164 ngx_http_stub_status_variable(ngx_http_request_t *r,
165     ngx_http_variable_value_t *v, uintptr_t data)
166 {
167     u_char            *p;
168     ngx_atomic_int_t   value;
169
170     p = ngx_pnalloc(r->pool, NGX_ATOMIC_T_LEN);
171     if (p == NULL) {
172         return NGX_ERROR;
173     }
174
175     switch (data) {
176     case 0:
177         value = *ngx_stat_active;
178         break;
179
180     case 1:
181         value = *ngx_stat_reading;
182         break;
183
184     case 2:
185         value = *ngx_stat_writing;
186         break;
187
188     case 3:
189         value = *ngx_stat_waiting;
190         break;
191
192     /* suppress warning */
193     default:
194         value = 0;
195         break;
```

```
196     }
197
198     v->len = ngx_sprintf(p, "%uA", value) - p;
199     v->valid = 1;
200     v->no_cacheable = 0;
201     v->not_found = 0;
202     v->data = p;
203
204     return NGX_OK;
205 }
206
207
208 static ngx_int_t
209 ngx_http_stub_status_add_variables(ngx_conf_t *cf)
210 {
211     ngx_http_variable_t  *var, *v;
212
213     for (v = ngx_http_stub_status_vars; v->name.len; v++) {
214         var = ngx_http_add_variable(cf, &v->name, v->flags);
215         if (var == NULL) {
216             return NGX_ERROR;
217         }
218
219         var->get_handler = v->get_handler;
220         var->data = v->data;
221     }
222
223     return NGX_OK;
224 }
225
226
227 static char *
228 ngx_http_set_stub_status(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
229 {
230     ngx_http_core_loc_conf_t  *clcf;
231
232     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
233     clcf->handler = ngx_http_stub_status_handler;
234
235     return NGX_CONF_OK;
236 }
```

One Level Up Top Level

src/http/modules/ngx_http_sub_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_body_filter
- ngx_http_next_header_filter
- ngx_http_sub_filter_commands
- ngx_http_sub_filter_module
- ngx_http_sub_filter_module_ctx

Data types defined

- ngx_http_sub_ctx_t
- ngx_http_sub_loc_conf_t
- ngx_http_sub_state_e

Functions defined

- ngx_http_sub_body_filter
- ngx_http_sub_create_conf
- ngx_http_sub_filter
- ngx_http_sub_filter_init
- ngx_http_sub_header_filter
- ngx_http_sub_merge_conf
- ngx_http_sub_output
- ngx_http_sub_parse

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_str_t                  match;
15      ngx_http_complex_value_t   value;
16
17      ngx_hash_t                 types;
18
19      ngx_flag_t                 once;
20      ngx_flag_t                 last_modified;
21
```

```
    ngx_array_t               *types_keys;
} ngx_http_sub_loc_conf_t;


typedef enum {
    sub_start_state = 0,
    sub_match_state,
} ngx_http_sub_state_e;


typedef struct {
    ngx_str_t                  match;
    ngx_str_t                  saved;
    ngx_str_t                  looked;

    ngx_uint_t                 once;   /* unsigned  once:1 */

    ngx_buf_t                 *buf;

    u_char                    *pos;
    u_char                    *copy_start;
    u_char                    *copy_end;

    ngx_chain_t               *in;
    ngx_chain_t               *out;
    ngx_chain_t              **last_out;
    ngx_chain_t               *busy;
    ngx_chain_t               *free;

    ngx_str_t                  sub;

    ngx_uint_t                 state;
} ngx_http_sub_ctx_t;


static ngx_int_t ngx_http_sub_output(ngx_http_request_t *r,
    ngx_http_sub_ctx_t *ctx);
static ngx_int_t ngx_http_sub_parse(ngx_http_request_t *r,
    ngx_http_sub_ctx_t *ctx);

static char * ngx_http_sub_filter(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static void *ngx_http_sub_create_conf(ngx_conf_t *cf);
static char *ngx_http_sub_merge_conf(ngx_conf_t *cf,
    void *parent, void *child);
static ngx_int_t ngx_http_sub_filter_init(ngx_conf_t *cf);


static ngx_command_t  ngx_http_sub_filter_commands[] = {

    { ngx_string("sub_filter"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_http_sub_filter,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("sub_filter_types"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_types_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_sub_loc_conf_t, types_keys),
      &ngx_http_html_default_types[0] },

    { ngx_string("sub_filter_once"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_sub_loc_conf_t, once),
      NULL },

    { ngx_string("sub_filter_last_modified"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_sub_loc_conf_t, last_modified),
```

```
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_sub_filter_module_ctx = {
    NULL,                                  /* preconfiguration */
    ngx_http_sub_filter_init,              /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_sub_create_conf,              /* create location configuration */
    ngx_http_sub_merge_conf                /* merge location configuration */
};


ngx_module_t  ngx_http_sub_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_sub_filter_module_ctx,       /* module context */
    ngx_http_sub_filter_commands,          /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static ngx_int_t
ngx_http_sub_header_filter(ngx_http_request_t *r)
{
    ngx_http_sub_ctx_t        *ctx;
    ngx_http_sub_loc_conf_t  *slcf;

    slcf = ngx_http_get_module_loc_conf(r, ngx_http_sub_filter_module);

    if (slcf->match.len == 0
        || r->headers_out.content_length_n == 0
        || ngx_http_test_content_type(r, &slcf->types) == NULL)
    {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_sub_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ctx->saved.data = ngx_pnalloc(r->pool, slcf->match.len);
    if (ctx->saved.data == NULL) {
        return NGX_ERROR;
    }

    ctx->looked.data = ngx_pnalloc(r->pool, slcf->match.len);
    if (ctx->looked.data == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_sub_filter_module);

    ctx->match = slcf->match;
    ctx->last_out = &ctx->out;

```

```
    r->filter_need_in_memory = 1;

    if (r == r->main) {
        ngx_http_clear_content_length(r);

        if (!slcf->last_modified) {
            ngx_http_clear_last_modified(r);
            ngx_http_clear_etag(r);

        } else {
            ngx_http_weak_etag(r);
        }
    }

    return ngx_http_next_header_filter(r);
}


static ngx_int_t
ngx_http_sub_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    ngx_int_t                  rc;
    ngx_buf_t                 *b;
    ngx_chain_t               *cl;
    ngx_http_sub_ctx_t        *ctx;
    ngx_http_sub_loc_conf_t   *slcf;

    ctx = ngx_http_get_module_ctx(r, ngx_http_sub_filter_module);

    if (ctx == NULL) {
        return ngx_http_next_body_filter(r, in);
    }

    if ((in == NULL
         && ctx->buf == NULL
         && ctx->in == NULL
         && ctx->busy == NULL))
    {
        return ngx_http_next_body_filter(r, in);
    }

    if (ctx->once && (ctx->buf == NULL || ctx->in == NULL)) {

        if (ctx->busy) {
            if (ngx_http_sub_output(r, ctx) == NGX_ERROR) {
                return NGX_ERROR;
            }
        }

        return ngx_http_next_body_filter(r, in);
    }

    /* add the incoming chain to the chain ctx->in */

    if (in) {
        if (ngx_chain_add_copy(r->pool, &ctx->in, in) != NGX_OK) {
            return NGX_ERROR;
        }
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "http sub filter \"%V\"", &r->uri);

    while (ctx->in || ctx->buf) {

        if (ctx->buf == NULL) {
            ctx->buf = ctx->in->buf;
            ctx->in = ctx->in->next;
            ctx->pos = ctx->buf->pos;
        }

        if (ctx->state == sub_start_state) {
            ctx->copy_start = ctx->pos;
            ctx->copy_end = ctx->pos;
        }

```

```
250         b = NULL;
251
252         while (ctx->pos < ctx->buf->last) {
253
254             ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
255                            "saved: \"%V\" state: %d", &ctx->saved, ctx->state);
256
257             rc = ngx_http_sub_parse(r, ctx);
258
259             ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
260                            "parse: %d, looked: \"%V\" %p-%p",
261                            rc, &ctx->looked, ctx->copy_start, ctx->copy_end);
262
263             if (rc == NGX_ERROR) {
264                 return rc;
265             }
266
267             if (ctx->saved.len) {
268
269                 ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
270                                "saved: \"%V\"", &ctx->saved);
271
272                 cl = ngx_chain_get_free_buf(r->pool, &ctx->free);
273                 if (cl == NULL) {
274                     return NGX_ERROR;
275                 }
276
277                 b = cl->buf;
278
279                 ngx_memzero(b, sizeof(ngx_buf_t));
280
281                 b->pos = ngx_pnalloc(r->pool, ctx->saved.len);
282                 if (b->pos == NULL) {
283                     return NGX_ERROR;
284                 }
285
286                 ngx_memcpy(b->pos, ctx->saved.data, ctx->saved.len);
287                 b->last = b->pos + ctx->saved.len;
288                 b->memory = 1;
289
290                 *ctx->last_out = cl;
291                 ctx->last_out = &cl->next;
292
293                 ctx->saved.len = 0;
294             }
295
296             if (ctx->copy_start != ctx->copy_end) {
297
298                 cl = ngx_chain_get_free_buf(r->pool, &ctx->free);
299                 if (cl == NULL) {
300                     return NGX_ERROR;
301                 }
302
303                 b = cl->buf;
304
305                 ngx_memcpy(b, ctx->buf, sizeof(ngx_buf_t));
306
307                 b->pos = ctx->copy_start;
308                 b->last = ctx->copy_end;
309                 b->shadow = NULL;
310                 b->last_buf = 0;
311                 b->last_in_chain = 0;
312                 b->recycled = 0;
313
314                 if (b->in_file) {
315                     b->file_last = b->file_pos + (b->last - ctx->buf->pos);
316                     b->file_pos += b->pos - ctx->buf->pos;
317                 }
318
319                 *ctx->last_out = cl;
320                 ctx->last_out = &cl->next;
321             }
322
323             if (ctx->state == sub_start_state) {
324                 ctx->copy_start = ctx->pos;
325                 ctx->copy_end = ctx->pos;
```

```
            } else {
                ctx->copy_start = NULL;
                ctx->copy_end = NULL;
            }

            if (ctx->looked.len > (size_t) (ctx->pos - ctx->buf->pos)) {
                ctx->saved.len = ctx->looked.len - (ctx->pos - ctx->buf->pos);
                ngx_memcpy(ctx->saved.data, ctx->looked.data, ctx->saved.len);
            }

            if (rc == NGX_AGAIN) {
                continue;
            }


            /* rc == NGX_OK */

            cl = ngx_chain_get_free_buf(r->pool, &ctx->free);
            if (cl == NULL) {
                return NGX_ERROR;
            }

            b = cl->buf;

            ngx_memzero(b, sizeof(ngx_buf_t));

            slcf = ngx_http_get_module_loc_conf(r, ngx_http_sub_filter_module);

            if (ctx->sub.data == NULL) {

                if (ngx_http_complex_value(r, &slcf->value, &ctx->sub)
                    != NGX_OK)
                {
                    return NGX_ERROR;
                }
            }

            if (ctx->sub.len) {
                b->memory = 1;
                b->pos = ctx->sub.data;
                b->last = ctx->sub.data + ctx->sub.len;

            } else {
                b->sync = 1;
            }

            *ctx->last_out = cl;
            ctx->last_out = &cl->next;

            ctx->once = slcf->once;

            continue;
        }

        if (ctx->looked.len
            && (ctx->buf->last_buf || ctx->buf->last_in_chain))
        {
            cl = ngx_chain_get_free_buf(r->pool, &ctx->free);
            if (cl == NULL) {
                return NGX_ERROR;
            }

            b = cl->buf;

            ngx_memzero(b, sizeof(ngx_buf_t));

            b->pos = ctx->looked.data;
            b->last = b->pos + ctx->looked.len;
            b->memory = 1;

            *ctx->last_out = cl;
            ctx->last_out = &cl->next;

            ctx->looked.len = 0;
        }
```

```
402
403         if (ctx->buf->last_buf || ctx->buf->flush || ctx->buf->sync
404             || ngx_buf_in_memory(ctx->buf))
405         {
406             if (b == NULL) {
407                 cl = ngx_chain_get_free_buf(r->pool, &ctx->free);
408                 if (cl == NULL) {
409                     return NGX_ERROR;
410                 }
411
412                 b = cl->buf;
413
414                 ngx_memzero(b, sizeof(ngx_buf_t));
415
416                 b->sync = 1;
417
418                 *ctx->last_out = cl;
419                 ctx->last_out = &cl->next;
420             }
421
422             b->last_buf = ctx->buf->last_buf;
423             b->last_in_chain = ctx->buf->last_in_chain;
424             b->flush = ctx->buf->flush;
425             b->shadow = ctx->buf;
426
427             b->recycled = ctx->buf->recycled;
428         }
429
430         ctx->buf = NULL;
431
432         ctx->saved.len = ctx->looked.len;
433         ngx_memcpy(ctx->saved.data, ctx->looked.data, ctx->looked.len);
434     }
435
436     if (ctx->out == NULL && ctx->busy == NULL) {
437         return NGX_OK;
438     }
439
440     return ngx_http_sub_output(r, ctx);
441 }
442
443
444 static ngx_int_t
445 ngx_http_sub_output(ngx_http_request_t *r, ngx_http_sub_ctx_t *ctx)
446 {
447     ngx_int_t     rc;
448     ngx_buf_t    *b;
449     ngx_chain_t  *cl;
450
451 #if 1
452     b = NULL;
453     for (cl = ctx->out; cl; cl = cl->next) {
454         ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
455                        "sub out: %p %p", cl->buf, cl->buf->pos);
456         if (cl->buf == b) {
457             ngx_log_error(NGX_LOG_ALERT, r->connection->log, 0,
458                           "the same buf was used in sub");
459             ngx_debug_point();
460             return NGX_ERROR;
461         }
462         b = cl->buf;
463     }
464 #endif
465
466     rc = ngx_http_next_body_filter(r, ctx->out);
467
468     if (ctx->busy == NULL) {
469         ctx->busy = ctx->out;
470
471     } else {
472         for (cl = ctx->busy; cl->next; cl = cl->next) { /* void */ }
473         cl->next = ctx->out;
474     }
475
476     ctx->out = NULL;
477     ctx->last_out = &ctx->out;
```

```
478
479     while (ctx->busy) {
480
481         cl = ctx->busy;
482         b = cl->buf;
483
484         if (ngx_buf_size(b) != 0) {
485             break;
486         }
487
488         if (b->shadow) {
489             b->shadow->pos = b->shadow->last;
490         }
491
492         ctx->busy = cl->next;
493
494         if (ngx_buf_in_memory(b) || b->in_file) {
495             /* add data bufs only to the free buf chain */
496
497             cl->next = ctx->free;
498             ctx->free = cl;
499         }
500     }
501
502     if (ctx->in || ctx->buf) {
503         r->buffered |= NGX_HTTP_SUB_BUFFERED;
504
505     } else {
506         r->buffered &= ~NGX_HTTP_SUB_BUFFERED;
507     }
508
509     return rc;
510 }
511
512
513 static ngx_int_t
514 ngx_http_sub_parse(ngx_http_request_t *r, ngx_http_sub_ctx_t *ctx)
515 {
516     u_char                *p, *last, *copy_end, ch, match;
517     size_t                 looked, i;
518     ngx_http_sub_state_e   state;
519
520     if (ctx->once) {
521         ctx->copy_start = ctx->pos;
522         ctx->copy_end = ctx->buf->last;
523         ctx->pos = ctx->buf->last;
524         ctx->looked.len = 0;
525
526         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0, "once");
527
528         return NGX_AGAIN;
529     }
530
531     state = ctx->state;
532     looked = ctx->looked.len;
533     last = ctx->buf->last;
534     copy_end = ctx->copy_end;
535
536     for (p = ctx->pos; p < last; p++) {
537
538         ch = *p;
539         ch = ngx_tolower(ch);
540
541         if (state == sub_start_state) {
542
543             /* the tight loop */
544
545             match = ctx->match.data[0];
546
547             for ( ;; ) {
548                 if (ch == match) {
549
550                     if (ctx->match.len == 1) {
551                         ctx->pos = p + 1;
552                         ctx->copy_end = p;
553
```

```
                    return NGX_OK;
                }

                copy_end = p;
                ctx->looked.data[0] = *p;
                looked = 1;
                state = sub_match_state;

                goto match_started;
            }

            if (++p == last) {
                break;
            }

            ch = *p;
            ch = ngx_tolower(ch);
        }

        ctx->state = state;
        ctx->pos = p;
        ctx->looked.len = looked;
        ctx->copy_end = p;

        if (ctx->copy_start == NULL) {
            ctx->copy_start = ctx->buf->pos;
        }

        return NGX_AGAIN;

    match_started:

        continue;
    }

    /* state == sub_match_state */

    if (ch == ctx->match.data[looked]) {
        ctx->looked.data[looked] = *p;
        looked++;

        if (looked == ctx->match.len) {

            ctx->state = sub_start_state;
            ctx->pos = p + 1;
            ctx->looked.len = 0;
            ctx->saved.len = 0;
            ctx->copy_end = copy_end;

            if (ctx->copy_start == NULL && copy_end) {
                ctx->copy_start = ctx->buf->pos;
            }

            return NGX_OK;
        }

    } else {
        /*
         * check if there is another partial match in previously
         * matched substring to catch cases like "aab" in "aaab"
         */

        ctx->looked.data[looked] = *p;
        looked++;

        for (i = 1; i < looked; i++) {
            if (ngx_strncasecmp(ctx->looked.data + i,
                                ctx->match.data, looked - i)
            == 0)
            {
                break;
            }
        }

        if (i < looked) {
            if (ctx->saved.len > i) {
```

```
                    ctx->saved.len = i;
                }

                if ((size_t) (p + 1 - ctx->buf->pos) >= looked - i) {
                    copy_end = p + 1 - (looked - i);
                }

                ngx_memmove(ctx->looked.data, ctx->looked.data + i, looked - i);
                looked = looked - i;

            } else {
                copy_end = p;
                looked = 0;
                state = sub_start_state;
            }

            if (ctx->saved.len) {
                p++;
                goto out;
            }
        }
    }

    ctx->saved.len = 0;

out:

    ctx->state = state;
    ctx->pos = p;
    ctx->looked.len = looked;

    ctx->copy_end = (state == sub_start_state) ? p : copy_end;

    if (ctx->copy_start == NULL && ctx->copy_end) {
        ctx->copy_start = ctx->buf->pos;
    }

    return NGX_AGAIN;
}


static char *
ngx_http_sub_filter(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_sub_loc_conf_t *slcf = conf;

    ngx_str_t                         *value;
    ngx_http_compile_complex_value_t   ccv;

    if (slcf->match.data) {
        return "is duplicate";
    }

    value = cf->args->elts;

    ngx_strlow(value[1].data, value[1].data, value[1].len);

    slcf->match = value[1];

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[2];
    ccv.complex_value = &slcf->value;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static void *
ngx_http_sub_create_conf(ngx_conf_t *cf)
{
```

```
706     ngx_http_sub_loc_conf_t  *slcf;
707
708     slcf = ngx_pcalloc(cf->pool, sizeof(ngx_http_sub_loc_conf_t));
709     if (slcf == NULL) {
710         return NULL;
711     }
712
713     /*
714      * set by ngx_pcalloc():
715      *
716      *     conf->match = { 0, NULL };
717      *     conf->types = { NULL };
718      *     conf->types_keys = NULL;
719      */
720
721     slcf->once = NGX_CONF_UNSET;
722     slcf->last_modified = NGX_CONF_UNSET;
723
724     return slcf;
725 }
726
727
728 static char *
729 ngx_http_sub_merge_conf(ngx_conf_t *cf, void *parent, void *child)
730 {
731     ngx_http_sub_loc_conf_t *prev = parent;
732     ngx_http_sub_loc_conf_t *conf = child;
733
734     ngx_conf_merge_value(conf->once, prev->once, 1);
735     ngx_conf_merge_str_value(conf->match, prev->match, "");
736     ngx_conf_merge_value(conf->last_modified, prev->last_modified, 0);
737
738     if (conf->value.value.data == NULL) {
739         conf->value = prev->value;
740     }
741
742     if (ngx_http_merge_types(cf, &conf->types_keys, &conf->types,
743                              &prev->types_keys, &prev->types,
744                              ngx_http_html_default_types)
745         != NGX_OK)
746     {
747         return NGX_CONF_ERROR;
748     }
749
750     return NGX_CONF_OK;
751 }
752
753
754 static ngx_int_t
755 ngx_http_sub_filter_init(ngx_conf_t *cf)
756 {
757     ngx_http_next_header_filter = ngx_http_top_header_filter;
758     ngx_http_top_header_filter = ngx_http_sub_header_filter;
759
760     ngx_http_next_body_filter = ngx_http_top_body_filter;
761     ngx_http_top_body_filter = ngx_http_sub_body_filter;
762
763     return NGX_OK;
764 }
```

One Level Up Top Level

src/http/modules/ngx_http_upstream_hash_module.c - nginx-1.7.10

Global variables defined

- ngx_http_upstream_hash_commands
- ngx_http_upstream_hash_module
- ngx_http_upstream_hash_module_ctx

Data types defined

- ngx_http_upstream_chash_point_t
- ngx_http_upstream_chash_points_t
- ngx_http_upstream_hash_peer_data_t
- ngx_http_upstream_hash_srv_conf_t

Functions defined

- ngx_http_upstream_add_chash_point
- ngx_http_upstream_find_chash_point
- ngx_http_upstream_get_chash_peer
- ngx_http_upstream_get_hash_peer
- ngx_http_upstream_hash
- ngx_http_upstream_hash_create_conf
- ngx_http_upstream_init_chash
- ngx_http_upstream_init_chash_peer
- ngx_http_upstream_init_hash
- ngx_http_upstream_init_hash_peer

Source code

```
1
2  /*
3  * Copyright (C) Roman Arutyunyan
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      uint32_t                         hash;
15      ngx_str_t                       *server;
16  } ngx_http_upstream_chash_point_t;
17
18
19  typedef struct {
```

```
    ngx_uint_t                          number;
    ngx_http_upstream_chash_point_t     point[1];
} ngx_http_upstream_chash_points_t;


typedef struct {
    ngx_http_complex_value_t            key;
    ngx_http_upstream_chash_points_t   *points;
} ngx_http_upstream_hash_srv_conf_t;


typedef struct {
    /* the round robin data must be first */
    ngx_http_upstream_rr_peer_data_t    rrp;
    ngx_http_upstream_hash_srv_conf_t  *conf;
    ngx_str_t                           key;
    ngx_uint_t                          tries;
    ngx_uint_t                          rehash;
    uint32_t                            hash;
    ngx_event_get_peer_pt               get_rr_peer;
} ngx_http_upstream_hash_peer_data_t;


static ngx_int_t ngx_http_upstream_init_hash(ngx_conf_t *cf,
    ngx_http_upstream_srv_conf_t *us);
static ngx_int_t ngx_http_upstream_init_hash_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us);
static ngx_int_t ngx_http_upstream_get_hash_peer(ngx_peer_connection_t *pc,
    void *data);

static ngx_int_t ngx_http_upstream_init_chash(ngx_conf_t *cf,
    ngx_http_upstream_srv_conf_t *us);
static void ngx_http_upstream_add_chash_point(
    ngx_http_upstream_chash_points_t *points, uint32_t hash, ngx_str_t *server);
static ngx_uint_t ngx_http_upstream_find_chash_point(
    ngx_http_upstream_chash_points_t *points, uint32_t hash);
static ngx_int_t ngx_http_upstream_init_chash_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us);
static ngx_int_t ngx_http_upstream_get_chash_peer(ngx_peer_connection_t *pc,
    void *data);

static void *ngx_http_upstream_hash_create_conf(ngx_conf_t *cf);
static char *ngx_http_upstream_hash(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


static ngx_command_t  ngx_http_upstream_hash_commands[] = {

    { ngx_string("hash"),
      NGX_HTTP_UPS_CONF|NGX_CONF_TAKE12,
      ngx_http_upstream_hash,
      NGX_HTTP_SRV_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_upstream_hash_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_http_upstream_hash_create_conf,    /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_upstream_hash_module = {
    NGX_MODULE_V1,
```

```
    &ngx_http_upstream_hash_module_ctx,    /* module context */
    ngx_http_upstream_hash_commands,       /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
     NULL,                                  /* init module */
     NULL,                                  /* init process */
     NULL,                                  /* init thread */
     NULL,                                  /* exit thread */
     NULL,                                  /* exit process */
     NULL,                                  /* exit master */
     NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_upstream_init_hash(ngx_conf_t *cf, ngx_http_upstream_srv_conf_t *us)
{
    if (ngx_http_upstream_init_round_robin(cf, us) != NGX_OK) {
        return NGX_ERROR;
    }

    us->peer.init = ngx_http_upstream_init_hash_peer;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_init_hash_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us)
{
    ngx_http_upstream_hash_srv_conf_t   *hcf;
    ngx_http_upstream_hash_peer_data_t  *hp;

    hp = ngx_palloc(r->pool, sizeof(ngx_http_upstream_hash_peer_data_t));
    if (hp == NULL) {
        return NGX_ERROR;
    }

    r->upstream->peer.data = &hp->rrp;

    if (ngx_http_upstream_init_round_robin_peer(r, us) != NGX_OK) {
        return NGX_ERROR;
    }

    r->upstream->peer.get = ngx_http_upstream_get_hash_peer;

    hcf = ngx_http_conf_upstream_srv_conf(us, ngx_http_upstream_hash_module);

    if (ngx_http_complex_value(r, &hcf->key, &hp->key) != NGX_OK) {
        return NGX_ERROR;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "upstream hash key:\"%V\"", &hp->key);

    hp->conf = hcf;
    hp->tries = 0;
    hp->rehash = 0;
    hp->hash = 0;
    hp->get_rr_peer = ngx_http_upstream_get_round_robin_peer;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_get_hash_peer(ngx_peer_connection_t *pc, void *data)
{
    ngx_http_upstream_hash_peer_data_t  *hp = data;

    time_t                        now;
    u_char                        buf[NGX_INT_T_LEN];
    size_t                        size;
    uint32_t                      hash;
    ngx_int_t                     w;
```

```
    uintptr_t                    m;
    ngx_uint_t                   i, n, p;
    ngx_http_upstream_rr_peer_t  *peer;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "get hash peer, try: %ui", pc->tries);

    if (hp->tries > 20 || hp->rrp.peers->single) {
        return hp->get_rr_peer(pc, &hp->rrp);
    }

    now = ngx_time();

    pc->cached = 0;
    pc->connection = NULL;

    for ( ;; ) {

        /*
         * Hash expression is compatible with Cache::Memcached:
         * ((crc32([REHASH] KEY) >> 16) & 0x7fff) + PREV_HASH
         * with REHASH omitted at the first iteration.
         */

        ngx_crc32_init(hash);

        if (hp->rehash > 0) {
            size = ngx_sprintf(buf, "%ui", hp->rehash) - buf;
            ngx_crc32_update(&hash, buf, size);
        }

        ngx_crc32_update(&hash, hp->key.data, hp->key.len);
        ngx_crc32_final(hash);

        hash = (hash >> 16) & 0x7fff;

        hp->hash += hash;
        hp->rehash++;

        if (!hp->rrp.peers->weighted) {
            p = hp->hash % hp->rrp.peers->number;

        } else {
            w = hp->hash % hp->rrp.peers->total_weight;

            for (i = 0; i < hp->rrp.peers->number; i++) {
                w -= hp->rrp.peers->peer[i].weight;
                if (w < 0) {
                    break;
                }
            }

            p = i;
        }

        n = p / (8 * sizeof(uintptr_t));
        m = (uintptr_t) 1 << p % (8 * sizeof(uintptr_t));

        if (hp->rrp.tried[n] & m) {
            goto next;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                       "get hash peer, value:%uD, peer:%ui", hp->hash, p);

        peer = &hp->rrp.peers->peer[p];

        if (peer->down) {
            goto next;
        }

        if (peer->max_fails
            && peer->fails >= peer->max_fails
            && now - peer->checked <= peer->fail_timeout)
        {
            goto next;
```

```
        }

        break;

    next:

        if (++hp->tries > 20) {
            return hp->get_rr_peer(pc, &hp->rrp);
        }
    }

    hp->rrp.current = p;

    pc->sockaddr = peer->sockaddr;
    pc->socklen = peer->socklen;
    pc->name = &peer->name;

    if (now - peer->checked > peer->fail_timeout) {
        peer->checked = now;
    }

    hp->rrp.tried[n] |= m;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_init_chash(ngx_conf_t *cf, ngx_http_upstream_srv_conf_t *us)
{
    u_char                              *host, *port, c;
    size_t                               host_len, port_len, size;
    uint32_t                             hash, base_hash, prev_hash;
    ngx_str_t                           *server;
    ngx_uint_t                           npoints, i, j;
    ngx_http_upstream_rr_peer_t         *peer;
    ngx_http_upstream_rr_peers_t        *peers;
    ngx_http_upstream_chash_points_t    *points;
    ngx_http_upstream_hash_srv_conf_t   *hcf;

    if (ngx_http_upstream_init_round_robin(cf, us) != NGX_OK) {
        return NGX_ERROR;
    }

    us->peer.init = ngx_http_upstream_init_chash_peer;

    peers = us->peer.data;
    npoints = peers->total_weight * 160;

    size = sizeof(ngx_http_upstream_chash_points_t)
           + sizeof(ngx_http_upstream_chash_point_t) * (npoints - 1);

    points = ngx_palloc(cf->pool, size);
    if (points == NULL) {
        return NGX_ERROR;
    }

    points->number = 0;

    for (i = 0; i < peers->number; i++) {
        peer = &peers->peer[i];
        server = &peer->server;

        /*
         * Hash expression is compatible with Cache::Memcached::Fast:
         * crc32(HOST \0 PORT PREV_HASH).
         */

        if (server->len >= 5
            && ngx_strncasecmp(server->data, (u_char *) "unix:", 5) == 0)
        {
            host = server->data + 5;
            host_len = server->len - 5;
            port = NULL;
            port_len = 0;
            goto done;
```

```
        }

        for (j = 0; j < server->len; j++) {
            c = server->data[server->len - j - 1];

            if (c == ':') {
                host = server->data;
                host_len = server->len - j - 1;
                port = server->data + server->len - j;
                port_len = j;
                goto done;
            }

            if (c < '0' || c > '9') {
                break;
            }
        }

        host = server->data;
        host_len = server->len;
        port = NULL;
        port_len = 0;

    done:

        ngx_crc32_init(base_hash);
        ngx_crc32_update(&base_hash, host, host_len);
        ngx_crc32_update(&base_hash, (u_char *) "", 1);
        ngx_crc32_update(&base_hash, port, port_len);

        prev_hash = 0;
        npoints = peer->weight * 160;

        for (j = 0; j < npoints; j++) {
            hash = base_hash;

            ngx_crc32_update(&hash, (u_char *) &prev_hash, sizeof(uint32_t));
            ngx_crc32_final(hash);

            ngx_http_upstream_add_chash_point(points, hash, &peer->server);

            prev_hash = hash;
        }
    }

    hcf = ngx_http_conf_upstream_srv_conf(us, ngx_http_upstream_hash_module);
    hcf->points = points;

    return NGX_OK;
}


static void
ngx_http_upstream_add_chash_point(ngx_http_upstream_chash_points_t *points,
    uint32_t hash, ngx_str_t *server)
{
    size_t                           size;
    ngx_uint_t                       i;
    ngx_http_upstream_chash_point_t  *point;

    i = ngx_http_upstream_find_chash_point(points, hash);
    point = &points->point[i];

    if (point->hash == hash) {
        return;
    }

    size = (points->number - i) * sizeof(ngx_http_upstream_chash_point_t);

    ngx_memmove(point + 1, point, size);

    points->number++;
    point->hash = hash;
    point->server = server;
}

```

```
400
401 static ngx_uint_t
402 ngx_http_upstream_find_chash_point(ngx_http_upstream_chash_points_t *points,
403     uint32_t hash)
404 {
405     ngx_uint_t                        i, j, k;
406     ngx_http_upstream_chash_point_t  *point;
407
408     /* find first point >= hash */
409
410     point = &points->point[0];
411
412     i = 0;
413     j = points->number;
414
415     while (i < j) {
416         k = (i + j) / 2;
417
418         if (hash > point[k].hash) {
419             i = k + 1;
420
421         } else if (hash < point[k].hash) {
422             j = k;
423
424         } else {
425             return k;
426         }
427     }
428
429     return i;
430 }
431
432
433 static ngx_int_t
434 ngx_http_upstream_init_chash_peer(ngx_http_request_t *r,
435     ngx_http_upstream_srv_conf_t *us)
436 {
437     uint32_t                             hash;
438     ngx_http_upstream_hash_srv_conf_t   *hcf;
439     ngx_http_upstream_hash_peer_data_t  *hp;
440
441     if (ngx_http_upstream_init_hash_peer(r, us) != NGX_OK) {
442         return NGX_ERROR;
443     }
444
445     r->upstream->peer.get = ngx_http_upstream_get_chash_peer;
446
447     hp = r->upstream->peer.data;
448     hcf = ngx_http_conf_upstream_srv_conf(us, ngx_http_upstream_hash_module);
449
450     hash = ngx_crc32_long(hp->key.data, hp->key.len);
451     hp->hash = ngx_http_upstream_find_chash_point(hcf->points, hash);
452
453     return NGX_OK;
454 }
455
456
457 static ngx_int_t
458 ngx_http_upstream_get_chash_peer(ngx_peer_connection_t *pc, void *data)
459 {
460     ngx_http_upstream_hash_peer_data_t  *hp = data;
461
462     time_t                               now;
463     intptr_t                             m;
464     ngx_str_t                           *server;
465     ngx_int_t                            total;
466     ngx_uint_t                           i, n;
467     ngx_http_upstream_rr_peer_t         *peer, *best;
468     ngx_http_upstream_chash_point_t     *point;
469     ngx_http_upstream_chash_points_t    *points;
470     ngx_http_upstream_hash_srv_conf_t   *hcf;
471
472     ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
473                    "get consistent hash peer, try: %ui", pc->tries);
474
475     pc->cached = 0;
```

```
476     pc->connection = NULL;
477
478     now = ngx_time();
479     hcf = hp->conf;
480
481     points = hcf->points;
482     point = &points->point[0];
483
484     for ( ;; ) {
485         server = point[hp->hash % points->number].server;
486
487         ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
488                        "consistent hash peer:%uD, server:\"%V\"",
489                        hp->hash, server);
490
491         best = NULL;
492         total = 0;
493
494         for (i = 0; i < hp->rrp.peers->number; i++) {
495
496             n = i / (8 * sizeof(uintptr_t));
497             m = (uintptr_t) 1 << i % (8 * sizeof(uintptr_t));
498
499             if (hp->rrp.tried[n] & m) {
500                 continue;
501             }
502
503             peer = &hp->rrp.peers->peer[i];
504
505             if (peer->down) {
506                 continue;
507             }
508
509             if (peer->server.len != server->len
510                 || ngx_strncmp(peer->server.data, server->data, server->len)
511                    != 0)
512             {
513                 continue;
514             }
515
516             if (peer->max_fails
517                 && peer->fails >= peer->max_fails
518                 && now - peer->checked <= peer->fail_timeout)
519             {
520                 continue;
521             }
522
523             peer->current_weight += peer->effective_weight;
524             total += peer->effective_weight;
525
526             if (peer->effective_weight < peer->weight) {
527                 peer->effective_weight++;
528             }
529
530             if (best == NULL || peer->current_weight > best->current_weight) {
531                 best = peer;
532             }
533         }
534
535         if (best) {
536             best->current_weight -= total;
537
538             i = best - &hp->rrp.peers->peer[0];
539
540             hp->rrp.current = i;
541
542             n = i / (8 * sizeof(uintptr_t));
543             m = (uintptr_t) 1 << i % (8 * sizeof(uintptr_t));
544
545             hp->rrp.tried[n] |= m;
546
547             if (now - best->checked > best->fail_timeout) {
548                 best->checked = now;
549             }
550
551             pc->sockaddr = best->sockaddr;
```

```
            pc->socklen = best->socklen;
            pc->name = &best->name;

            return NGX_OK;
        }

        hp->hash++;
        hp->tries++;

        if (hp->tries >= points->number) {
            return NGX_BUSY;
        }
    }
}


static void *
ngx_http_upstream_hash_create_conf(ngx_conf_t *cf)
{
    ngx_http_upstream_hash_srv_conf_t  *conf;

    conf = ngx_palloc(cf->pool, sizeof(ngx_http_upstream_hash_srv_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    conf->points = NULL;

    return conf;
}


static char *
ngx_http_upstream_hash(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_upstream_hash_srv_conf_t  *hcf = conf;

    ngx_str_t                          *value;
    ngx_http_upstream_srv_conf_t       *uscf;
    ngx_http_compile_complex_value_t    ccv;

    value = cf->args->elts;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &hcf->key;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    uscf = ngx_http_conf_get_module_srv_conf(cf, ngx_http_upstream_module);

    if (uscf->peer.init_upstream) {
        ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
                           "load balancing method redefined");
    }

    uscf->flags = NGX_HTTP_UPSTREAM_CREATE
                  |NGX_HTTP_UPSTREAM_WEIGHT
                  |NGX_HTTP_UPSTREAM_MAX_FAILS
                  |NGX_HTTP_UPSTREAM_FAIL_TIMEOUT
                  |NGX_HTTP_UPSTREAM_DOWN;

    if (cf->args->nelts == 2) {
        uscf->peer.init_upstream = ngx_http_upstream_init_hash;

    } else if (ngx_strcmp(value[2].data, "consistent") == 0) {
        uscf->peer.init_upstream = ngx_http_upstream_init_chash;

    } else {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "invalid parameter \"%V\"", &value[2]);
        return NGX_CONF_ERROR;
```

```
628     }
629
630     return NGX_CONF_OK;
631 }
```

One Level Up Top Level

src/http/modules/ngx_http_upstream_ip_hash_module.c - nginx-1.7.10

Global variables defined

- ngx_http_upstream_ip_hash_commands
- ngx_http_upstream_ip_hash_module
- ngx_http_upstream_ip_hash_module_ctx
- ngx_http_upstream_ip_hash_pseudo_addr

Data types defined

- ngx_http_upstream_ip_hash_peer_data_t

Functions defined

- ngx_http_upstream_get_ip_hash_peer
- ngx_http_upstream_init_ip_hash
- ngx_http_upstream_init_ip_hash_peer
- ngx_http_upstream_ip_hash

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      /* the round robin data must be first */
15      ngx_http_upstream_rr_peer_data_t   rrp;
16
17      ngx_uint_t                         hash;
18
19      u_char                             addrlen;
20      u_char                            *addr;
21
22      u_char                             tries;
23
24      ngx_event_get_peer_pt              get_rr_peer;
25  } ngx_http_upstream_ip_hash_peer_data_t;
26
27
28  static ngx_int_t ngx_http_upstream_init_ip_hash_peer(ngx_http_request_t *r,
29      ngx_http_upstream_srv_conf_t *us);
30  static ngx_int_t ngx_http_upstream_get_ip_hash_peer(ngx_peer_connection_t *pc,
31      void *data);
32  static char *ngx_http_upstream_ip_hash(ngx_conf_t *cf, ngx_command_t *cmd,
33      void *conf);
34
35
36  static ngx_command_t  ngx_http_upstream_ip_hash_commands[] = {
37
```

```
    { ngx_string("ip_hash"),
      NGX_HTTP_UPS_CONF|NGX_CONF_NOARGS,
      ngx_http_upstream_ip_hash,
      0,
      0,
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_upstream_ip_hash_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_upstream_ip_hash_module = {
    NGX_MODULE_V1,
    &ngx_http_upstream_ip_hash_module_ctx, /* module context */
    ngx_http_upstream_ip_hash_commands,    /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static u_char ngx_http_upstream_ip_hash_pseudo_addr[3];


static ngx_int_t
ngx_http_upstream_init_ip_hash(ngx_conf_t *cf, ngx_http_upstream_srv_conf_t *us)
{
    if (ngx_http_upstream_init_round_robin(cf, us) != NGX_OK) {
        return NGX_ERROR;
    }

    us->peer.init = ngx_http_upstream_init_ip_hash_peer;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_init_ip_hash_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us)
{
    struct sockaddr_in                  *sin;
#if (NGX_HAVE_INET6)
    struct sockaddr_in6                 *sin6;
#endif
    ngx_http_upstream_ip_hash_peer_data_t  *iphp;

    iphp = ngx_palloc(r->pool, sizeof(ngx_http_upstream_ip_hash_peer_data_t));
    if (iphp == NULL) {
        return NGX_ERROR;
    }

    r->upstream->peer.data = &iphp->rrp;

    if (ngx_http_upstream_init_round_robin_peer(r, us) != NGX_OK) {
```

```
        return NGX_ERROR;
    }

    r->upstream->peer.get = ngx_http_upstream_get_ip_hash_peer;

    switch (r->connection->sockaddr->sa_family) {

    case AF_INET:
        sin = (struct sockaddr_in *) r->connection->sockaddr;
        iphp->addr = (u_char *) &sin->sin_addr.s_addr;
        iphp->addrlen = 3;
        break;

#if (NGX_HAVE_INET6)
    case AF_INET6:
        sin6 = (struct sockaddr_in6 *) r->connection->sockaddr;
        iphp->addr = (u_char *) &sin6->sin6_addr.s6_addr;
        iphp->addrlen = 16;
        break;
#endif

    default:
        iphp->addr = ngx_http_upstream_ip_hash_pseudo_addr;
        iphp->addrlen = 3;
    }

    iphp->hash = 89;
    iphp->tries = 0;
    iphp->get_rr_peer = ngx_http_upstream_get_round_robin_peer;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_get_ip_hash_peer(ngx_peer_connection_t *pc, void *data)
{
    ngx_http_upstream_ip_hash_peer_data_t  *iphp = data;

    time_t                        now;
    ngx_int_t                     w;
    uintptr_t                     m;
    ngx_uint_t                    i, n, p, hash;
    ngx_http_upstream_rr_peer_t  *peer;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "get ip hash peer, try: %ui", pc->tries);

    /* TODO: cached */

    if (iphp->tries > 20 || iphp->rrp.peers->single) {
        return iphp->get_rr_peer(pc, &iphp->rrp);
    }

    now = ngx_time();

    pc->cached = 0;
    pc->connection = NULL;

    hash = iphp->hash;

    for ( ;; ) {

        for (i = 0; i < (ngx_uint_t) iphp->addrlen; i++) {
            hash = (hash * 113 + iphp->addr[i]) % 6271;
        }

        if (!iphp->rrp.peers->weighted) {
            p = hash % iphp->rrp.peers->number;

        } else {
            w = hash % iphp->rrp.peers->total_weight;

            for (i = 0; i < iphp->rrp.peers->number; i++) {
                w -= iphp->rrp.peers->peer[i].weight;
                if (w < 0) {
```

```
                    break;
                }
            }

            p = i;
        }

        n = p / (8 * sizeof(uintptr_t));
        m = (uintptr_t) 1 << p % (8 * sizeof(uintptr_t));

        if (iphp->rrp.tried[n] & m) {
            goto next;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                       "get ip hash peer, hash: %ui %04XA", p, m);

        peer = &iphp->rrp.peers->peer[p];

        /* ngx_lock_mutex(iphp->rrp.peers->mutex); */

        if (peer->down) {
            goto next_try;
        }

        if (peer->max_fails
            && peer->fails >= peer->max_fails
            && now - peer->checked <= peer->fail_timeout)
        {
            goto next_try;
        }

        break;

    next_try:

        iphp->rrp.tried[n] |= m;

        /* ngx_unlock_mutex(iphp->rrp.peers->mutex); */

        pc->tries--;

    next:

        if (++iphp->tries > 20) {
            return iphp->get_rr_peer(pc, &iphp->rrp);
        }
    }

    iphp->rrp.current = p;

    pc->sockaddr = peer->sockaddr;
    pc->socklen = peer->socklen;
    pc->name = &peer->name;

    if (now - peer->checked > peer->fail_timeout) {
        peer->checked = now;
    }

    /* ngx_unlock_mutex(iphp->rrp.peers->mutex); */

    iphp->rrp.tried[n] |= m;
    iphp->hash = hash;

    return NGX_OK;
}


static char *
ngx_http_upstream_ip_hash(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_upstream_srv_conf_t  *uscf;

    uscf = ngx_http_conf_get_module_srv_conf(cf, ngx_http_upstream_module);

    if (uscf->peer.init_upstream) {
```

```
266         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
267                            "load balancing method redefined");
268     }
269
270     uscf->peer.init_upstream = ngx_http_upstream_init_ip_hash;
271
272     uscf->flags = NGX_HTTP_UPSTREAM_CREATE
273                   |NGX_HTTP_UPSTREAM_WEIGHT
274                   |NGX_HTTP_UPSTREAM_MAX_FAILS
275                   |NGX_HTTP_UPSTREAM_FAIL_TIMEOUT
276                   |NGX_HTTP_UPSTREAM_DOWN;
277
278     return NGX_CONF_OK;
279 }
```

One Level Up Top Level

src/http/modules/ngx_http_upstream_keepalive_module.c - nginx-1.7.10

Global variables defined

- ngx_http_upstream_keepalive_commands
- ngx_http_upstream_keepalive_module
- ngx_http_upstream_keepalive_module_ctx

Data types defined

- ngx_http_upstream_keepalive_cache_t
- ngx_http_upstream_keepalive_peer_data_t
- ngx_http_upstream_keepalive_srv_conf_t

Functions defined

- ngx_http_upstream_free_keepalive_peer
- ngx_http_upstream_get_keepalive_peer
- ngx_http_upstream_init_keepalive
- ngx_http_upstream_init_keepalive_peer
- ngx_http_upstream_keepalive
- ngx_http_upstream_keepalive_close
- ngx_http_upstream_keepalive_close_handler
- ngx_http_upstream_keepalive_create_conf
- ngx_http_upstream_keepalive_dummy_handler
- ngx_http_upstream_keepalive_save_session
- ngx_http_upstream_keepalive_set_session

Source code

```
1
2  /*
3   * Copyright (C) Maxim Dounin
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_uint_t                      max_cached;
15
16      ngx_queue_t                     cache;
17      ngx_queue_t                     free;
18
19      ngx_http_upstream_init_pt       original_init_upstream;
```

```
    ngx_http_upstream_init_peer_pt     original_init_peer;

} ngx_http_upstream_keepalive_srv_conf_t;


typedef struct {
    ngx_http_upstream_keepalive_srv_conf_t  *conf;

    ngx_http_upstream_t               *upstream;

    void                              *data;

    ngx_event_get_peer_pt              original_get_peer;
    ngx_event_free_peer_pt             original_free_peer;

#if (NGX_HTTP_SSL)
    ngx_event_set_peer_session_pt      original_set_session;
    ngx_event_save_peer_session_pt     original_save_session;
#endif

} ngx_http_upstream_keepalive_peer_data_t;


typedef struct {
    ngx_http_upstream_keepalive_srv_conf_t  *conf;

    ngx_queue_t                        queue;
    ngx_connection_t                  *connection;

    socklen_t                          socklen;
    u_char                             sockaddr[NGX_SOCKADDRLEN];

} ngx_http_upstream_keepalive_cache_t;


static ngx_int_t ngx_http_upstream_init_keepalive_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us);
static ngx_int_t ngx_http_upstream_get_keepalive_peer(ngx_peer_connection_t *pc,
    void *data);
static void ngx_http_upstream_free_keepalive_peer(ngx_peer_connection_t *pc,
    void *data, ngx_uint_t state);

static void ngx_http_upstream_keepalive_dummy_handler(ngx_event_t *ev);
static void ngx_http_upstream_keepalive_close_handler(ngx_event_t *ev);
static void ngx_http_upstream_keepalive_close(ngx_connection_t *c);


#if (NGX_HTTP_SSL)
static ngx_int_t ngx_http_upstream_keepalive_set_session(
    ngx_peer_connection_t *pc, void *data);
static void ngx_http_upstream_keepalive_save_session(ngx_peer_connection_t *pc,
    void *data);
#endif

static void *ngx_http_upstream_keepalive_create_conf(ngx_conf_t *cf);
static char *ngx_http_upstream_keepalive(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


static ngx_command_t  ngx_http_upstream_keepalive_commands[] = {

    { ngx_string("keepalive"),
      NGX_HTTP_UPS_CONF|NGX_CONF_TAKE1,
      ngx_http_upstream_keepalive,
      NGX_HTTP_SRV_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_upstream_keepalive_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

```

```
    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_http_upstream_keepalive_create_conf, /* create server configuration */
     NULL,                                  /* merge server configuration */

     NULL,                                  /* create location configuration */
     NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_upstream_keepalive_module = {
    NGX_MODULE_V1,
    &ngx_http_upstream_keepalive_module_ctx, /* module context */
    ngx_http_upstream_keepalive_commands,    /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_upstream_init_keepalive(ngx_conf_t *cf,
    ngx_http_upstream_srv_conf_t *us)
{
    ngx_uint_t                          i;
    ngx_http_upstream_keepalive_srv_conf_t  *kcf;
    ngx_http_upstream_keepalive_cache_t     *cached;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, cf->log, 0,
                   "init keepalive");

    kcf = ngx_http_conf_upstream_srv_conf(us,
                                          ngx_http_upstream_keepalive_module);

    if (kcf->original_init_upstream(cf, us) != NGX_OK) {
        return NGX_ERROR;
    }

    kcf->original_init_peer = us->peer.init;

    us->peer.init = ngx_http_upstream_init_keepalive_peer;

    /* allocate cache items and add to free queue */

    cached = ngx_pcalloc(cf->pool,
                sizeof(ngx_http_upstream_keepalive_cache_t) * kcf->max_cached);
    if (cached == NULL) {
        return NGX_ERROR;
    }

    ngx_queue_init(&kcf->cache);
    ngx_queue_init(&kcf->free);

    for (i = 0; i < kcf->max_cached; i++) {
        ngx_queue_insert_head(&kcf->free, &cached[i].queue);
        cached[i].conf = kcf;
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_init_keepalive_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us)
{
    ngx_http_upstream_keepalive_peer_data_t  *kp;
    ngx_http_upstream_keepalive_srv_conf_t   *kcf;

```

```
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "init keepalive peer");

    kcf = ngx_http_conf_upstream_srv_conf(us,
                                          ngx_http_upstream_keepalive_module);

    kp = ngx_palloc(r->pool, sizeof(ngx_http_upstream_keepalive_peer_data_t));
    if (kp == NULL) {
        return NGX_ERROR;
    }

    if (kcf->original_init_peer(r, us) != NGX_OK) {
        return NGX_ERROR;
    }

    kp->conf = kcf;
    kp->upstream = r->upstream;
    kp->data = r->upstream->peer.data;
    kp->original_get_peer = r->upstream->peer.get;
    kp->original_free_peer = r->upstream->peer.free;

    r->upstream->peer.data = kp;
    r->upstream->peer.get = ngx_http_upstream_get_keepalive_peer;
    r->upstream->peer.free = ngx_http_upstream_free_keepalive_peer;

#if (NGX_HTTP_SSL)
    kp->original_set_session = r->upstream->peer.set_session;
    kp->original_save_session = r->upstream->peer.save_session;
    r->upstream->peer.set_session = ngx_http_upstream_keepalive_set_session;
    r->upstream->peer.save_session = ngx_http_upstream_keepalive_save_session;
#endif

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_get_keepalive_peer(ngx_peer_connection_t *pc, void *data)
{
    ngx_http_upstream_keepalive_peer_data_t  *kp = data;
    ngx_http_upstream_keepalive_cache_t      *item;

    ngx_int_t          rc;
    ngx_queue_t       *q, *cache;
    ngx_connection_t  *c;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "get keepalive peer");

    /* ask balancer */

    rc = kp->original_get_peer(pc, kp->data);

    if (rc != NGX_OK) {
        return rc;
    }

    /* search cache for suitable connection */

    cache = &kp->conf->cache;

    for (q = ngx_queue_head(cache);
         q != ngx_queue_sentinel(cache);
         q = ngx_queue_next(q))
    {
        item = ngx_queue_data(q, ngx_http_upstream_keepalive_cache_t, queue);
        c = item->connection;

        if (ngx_memn2cmp((u_char *) &item->sockaddr, (u_char *) pc->sockaddr,
                         item->socklen, pc->socklen)
            == 0)
        {
            ngx_queue_remove(q);
            ngx_queue_insert_head(&kp->conf->free, q);

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
```

```
                           "get keepalive peer: using connection %p", c);

            c->idle = 0;
            c->sent = 0;
            c->log = pc->log;
            c->read->log = pc->log;
            c->write->log = pc->log;
            c->pool->log = pc->log;

            pc->connection = c;
            pc->cached = 1;

            return NGX_DONE;
        }
    }

    return NGX_OK;
}


static void
ngx_http_upstream_free_keepalive_peer(ngx_peer_connection_t *pc, void *data,
    ngx_uint_t state)
{
    ngx_http_upstream_keepalive_peer_data_t  *kp = data;
    ngx_http_upstream_keepalive_cache_t      *item;

    ngx_queue_t          *q;
    ngx_connection_t     *c;
    ngx_http_upstream_t  *u;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "free keepalive peer");

    /* cache valid connections */

    u = kp->upstream;
    c = pc->connection;

    if (state & NGX_PEER_FAILED
        || c == NULL
        || c->read->eof
        || c->read->error
        || c->read->timedout
        || c->write->error
        || c->write->timedout)
    {
        goto invalid;
    }

    if (!u->keepalive) {
        goto invalid;
    }

    if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
        goto invalid;
    }

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "free keepalive peer: saving connection %p", c);

    if (ngx_queue_empty(&kp->conf->free)) {

        q = ngx_queue_last(&kp->conf->cache);
        ngx_queue_remove(q);

        item = ngx_queue_data(q, ngx_http_upstream_keepalive_cache_t, queue);

        ngx_http_upstream_keepalive_close(item->connection);

    } else {
        q = ngx_queue_head(&kp->conf->free);
        ngx_queue_remove(q);

        item = ngx_queue_data(q, ngx_http_upstream_keepalive_cache_t, queue);
    }
```

```
    item->connection = c;
    ngx_queue_insert_head(&kp->conf->cache, q);

    pc->connection = NULL;

    if (c->read->timer_set) {
        ngx_del_timer(c->read);
    }
    if (c->write->timer_set) {
        ngx_del_timer(c->write);
    }

    c->write->handler = ngx_http_upstream_keepalive_dummy_handler;
    c->read->handler = ngx_http_upstream_keepalive_close_handler;

    c->data = item;
    c->idle = 1;
    c->log = ngx_cycle->log;
    c->read->log = ngx_cycle->log;
    c->write->log = ngx_cycle->log;
    c->pool->log = ngx_cycle->log;

    item->socklen = pc->socklen;
    ngx_memcpy(&item->sockaddr, pc->sockaddr, pc->socklen);

    if (c->read->ready) {
        ngx_http_upstream_keepalive_close_handler(c->read);
    }

invalid:

    kp->original_free_peer(pc, kp->data, state);
}


static void
ngx_http_upstream_keepalive_dummy_handler(ngx_event_t *ev)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, ev->log, 0,
                   "keepalive dummy handler");
}


static void
ngx_http_upstream_keepalive_close_handler(ngx_event_t *ev)
{
    ngx_http_upstream_keepalive_srv_conf_t  *conf;
    ngx_http_upstream_keepalive_cache_t     *item;

    int                n;
    char               buf[1];
    ngx_connection_t  *c;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, ev->log, 0,
                   "keepalive close handler");

    c = ev->data;

    if (c->close) {
        goto close;
    }

    n = recv(c->fd, buf, 1, MSG_PEEK);

    if (n == -1 && ngx_socket_errno == NGX_EAGAIN) {
        /* stale event */

        if (ngx_handle_read_event(c->read, 0) != NGX_OK) {
            goto close;
        }

        return;
    }

close:
```

```
400
401     item = c->data;
402     conf = item->conf;
403
404     ngx_http_upstream_keepalive_close(c);
405
406     ngx_queue_remove(&item->queue);
407     ngx_queue_insert_head(&conf->free, &item->queue);
408 }
409
410
411 static void
412 ngx_http_upstream_keepalive_close(ngx_connection_t *c)
413 {
414
415 #if (NGX_HTTP_SSL)
416
417     if (c->ssl) {
418         c->ssl->no_wait_shutdown = 1;
419         c->ssl->no_send_shutdown = 1;
420
421         if (ngx_ssl_shutdown(c) == NGX_AGAIN) {
422             c->ssl->handler = ngx_http_upstream_keepalive_close;
423             return;
424         }
425     }
426
427 #endif
428
429     ngx_destroy_pool(c->pool);
430     ngx_close_connection(c);
431 }
432
433
434 #if (NGX_HTTP_SSL)
435
436 static ngx_int_t
437 ngx_http_upstream_keepalive_set_session(ngx_peer_connection_t *pc, void *data)
438 {
439     ngx_http_upstream_keepalive_peer_data_t  *kp = data;
440
441     return kp->original_set_session(pc, kp->data);
442 }
443
444
445 static void
446 ngx_http_upstream_keepalive_save_session(ngx_peer_connection_t *pc, void *data)
447 {
448     ngx_http_upstream_keepalive_peer_data_t  *kp = data;
449
450     kp->original_save_session(pc, kp->data);
451     return;
452 }
453
454 #endif
455
456
457 static void *
458 ngx_http_upstream_keepalive_create_conf(ngx_conf_t *cf)
459 {
460     ngx_http_upstream_keepalive_srv_conf_t  *conf;
461
462     conf = ngx_pcalloc(cf->pool,
463                        sizeof(ngx_http_upstream_keepalive_srv_conf_t));
464     if (conf == NULL) {
465         return NULL;
466     }
467
468     /*
469      * set by ngx_pcalloc():
470      *
471      *     conf->original_init_upstream = NULL;
472      *     conf->original_init_peer = NULL;
473      */
474
475     conf->max_cached = 1;
```

```
476
477     return conf;
478 }
479
480
481 static char *
482 ngx_http_upstream_keepalive(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
483 {
484     ngx_http_upstream_srv_conf_t            *uscf;
485     ngx_http_upstream_keepalive_srv_conf_t  *kcf = conf;
486
487     ngx_int_t    n;
488     ngx_str_t   *value;
489
490     uscf = ngx_http_conf_get_module_srv_conf(cf, ngx_http_upstream_module);
491
492     if (kcf->original_init_upstream) {
493         return "is duplicate";
494     }
495
496     kcf->original_init_upstream = uscf->peer.init_upstream
497                                   ? uscf->peer.init_upstream
498                                   : ngx_http_upstream_init_round_robin;
499
500     uscf->peer.init_upstream = ngx_http_upstream_init_keepalive;
501
502     /* read options */
503
504     value = cf->args->elts;
505
506     n = ngx_atoi(value[1].data, value[1].len);
507
508     if (n == NGX_ERROR || n == 0) {
509         ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
510                            "invalid value \"%V\" in \"%V\" directive",
511                            &value[1], &cmd->name);
512         return NGX_CONF_ERROR;
513     }
514
515     kcf->max_cached = n;
516
517     return NGX_CONF_OK;
518 }
```

One Level Up Top Level

src/http/modules/ngx_http_upstream_least_conn_module.c - nginx-1.7.10

Global variables defined

- ngx_http_upstream_least_conn_commands
- ngx_http_upstream_least_conn_module
- ngx_http_upstream_least_conn_module_ctx

Data types defined

- ngx_http_upstream_lc_peer_data_t
- ngx_http_upstream_least_conn_conf_t

Functions defined

- ngx_http_upstream_free_least_conn_peer
- ngx_http_upstream_get_least_conn_peer
- ngx_http_upstream_init_least_conn
- ngx_http_upstream_init_least_conn_peer
- ngx_http_upstream_least_conn
- ngx_http_upstream_least_conn_create_conf

Source code

```
1
2  /*
3  * Copyright (C) Maxim Dounin
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12
13  typedef struct {
14      ngx_uint_t                       *conns;
15  } ngx_http_upstream_least_conn_conf_t;
16
17
18  typedef struct {
19      /* the round robin data must be first */
20      ngx_http_upstream_rr_peer_data_t   rrp;
21
22      ngx_uint_t                       *conns;
23
24      ngx_event_get_peer_pt             get_rr_peer;
25      ngx_event_free_peer_pt            free_rr_peer;
26  } ngx_http_upstream_lc_peer_data_t;
27
28
29  static ngx_int_t ngx_http_upstream_init_least_conn_peer(ngx_http_request_t *r,
30      ngx_http_upstream_srv_conf_t *us);
31  static ngx_int_t ngx_http_upstream_get_least_conn_peer(
32      ngx_peer_connection_t *pc, void *data);
33  static void ngx_http_upstream_free_least_conn_peer(ngx_peer_connection_t *pc,
```

```
    void *data, ngx_uint_t state);
static void *ngx_http_upstream_least_conn_create_conf(ngx_conf_t *cf);
static char *ngx_http_upstream_least_conn(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);


static ngx_command_t  ngx_http_upstream_least_conn_commands[] = {

    { ngx_string("least_conn"),
      NGX_HTTP_UPS_CONF|NGX_CONF_NOARGS,
      ngx_http_upstream_least_conn,
      0,
      0,
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_upstream_least_conn_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    NULL,                                  /* create main configuration */
    NULL,                                  /* init main configuration */

    ngx_http_upstream_least_conn_create_conf, /* create server configuration */
    NULL,                                  /* merge server configuration */

    NULL,                                  /* create location configuration */
    NULL                                   /* merge location configuration */
};


ngx_module_t  ngx_http_upstream_least_conn_module = {
    NGX_MODULE_V1,
    &ngx_http_upstream_least_conn_module_ctx, /* module context */
    ngx_http_upstream_least_conn_commands, /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_int_t
ngx_http_upstream_init_least_conn(ngx_conf_t *cf,
    ngx_http_upstream_srv_conf_t *us)
{
    ngx_uint_t                          n;
    ngx_http_upstream_rr_peers_t       *peers;
    ngx_http_upstream_least_conn_conf_t  *lcf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, cf->log, 0,
                   "init least conn");

    if (ngx_http_upstream_init_round_robin(cf, us) != NGX_OK) {
        return NGX_ERROR;
    }

    peers = us->peer.data;

     n = peers->number;

     if (peers->next) {
         n += peers->next->number;
     }

     lcf = ngx_http_conf_upstream_srv_conf(us,
                                           ngx_http_upstream_least_conn_module);

```

```
    lcf->conns = ngx_pcalloc(cf->pool, sizeof(ngx_uint_t) * n);
    if (lcf->conns == NULL) {
        return NGX_ERROR;
    }

    us->peer.init = ngx_http_upstream_init_least_conn_peer;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_init_least_conn_peer(ngx_http_request_t *r,
    ngx_http_upstream_srv_conf_t *us)
{
    ngx_http_upstream_lc_peer_data_t     *lcp;
    ngx_http_upstream_least_conn_conf_t  *lcf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "init least conn peer");

    lcf = ngx_http_conf_upstream_srv_conf(us,
                                          ngx_http_upstream_least_conn_module);

    lcp = ngx_palloc(r->pool, sizeof(ngx_http_upstream_lc_peer_data_t));
    if (lcp == NULL) {
        return NGX_ERROR;
    }

    lcp->conns = lcf->conns;

    r->upstream->peer.data = &lcp->rrp;

    if (ngx_http_upstream_init_round_robin_peer(r, us) != NGX_OK) {
        return NGX_ERROR;
    }

    r->upstream->peer.get = ngx_http_upstream_get_least_conn_peer;
    r->upstream->peer.free = ngx_http_upstream_free_least_conn_peer;

    lcp->get_rr_peer = ngx_http_upstream_get_round_robin_peer;
    lcp->free_rr_peer = ngx_http_upstream_free_round_robin_peer;

    return NGX_OK;
}


static ngx_int_t
ngx_http_upstream_get_least_conn_peer(ngx_peer_connection_t *pc, void *data)
{
    ngx_http_upstream_lc_peer_data_t  *lcp = data;

    time_t                        now;
    uintptr_t                     m;
    ngx_int_t                     rc, total;
    ngx_uint_t                    i, n, p, many;
    ngx_http_upstream_rr_peer_t   *peer, *best;
    ngx_http_upstream_rr_peers_t  *peers;

    ngx_log_debug1(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                   "get least conn peer, try: %ui", pc->tries);

    if (lcp->rrp.peers->single) {
        return lcp->get_rr_peer(pc, &lcp->rrp);
    }

    pc->cached = 0;
    pc->connection = NULL;

    now = ngx_time();

    peers = lcp->rrp.peers;

    best = NULL;
    total = 0;

```

```
#if (NGX_SUPPRESS_WARN)
    many = 0;
    p = 0;
#endif

    for (i = 0; i < peers->number; i++) {

        n = i / (8 * sizeof(uintptr_t));
        m = (uintptr_t) 1 << i % (8 * sizeof(uintptr_t));

        if (lcp->rrp.tried[n] & m) {
            continue;
        }

        peer = &peers->peer[i];

        if (peer->down) {
            continue;
        }

        if (peer->max_fails
            && peer->fails >= peer->max_fails
            && now - peer->checked <= peer->fail_timeout)
        {
            continue;
        }

        /*
         * select peer with least number of connections; if there are
         * multiple peers with the same number of connections, select
         * based on round-robin
         */

        if (best == NULL
            || lcp->conns[i] * best->weight < lcp->conns[p] * peer->weight)
        {
            best = peer;
            many = 0;
            p = i;

        } else if (lcp->conns[i] * best->weight
                   == lcp->conns[p] * peer->weight)
        {
            many = 1;
        }
    }

    if (best == NULL) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                       "get least conn peer, no peer found");

        goto failed;
    }

    if (many) {
        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pc->log, 0,
                       "get least conn peer, many");

        for (i = p; i < peers->number; i++) {

            n = i / (8 * sizeof(uintptr_t));
            m = (uintptr_t) 1 << i % (8 * sizeof(uintptr_t));

            if (lcp->rrp.tried[n] & m) {
                continue;
            }

            peer = &peers->peer[i];

            if (peer->down) {
                continue;
            }

            if (lcp->conns[i] * best->weight != lcp->conns[p] * peer->weight) {
                continue;
            }
```

```
262
263             if (peer->max_fails
264                 && peer->fails >= peer->max_fails
265                 && now - peer->checked <= peer->fail_timeout)
266             {
267                 continue;
268             }
269
270             peer->current_weight += peer->effective_weight;
271             total += peer->effective_weight;
272
273             if (peer->effective_weight < peer->weight) {
274                 peer->effective_weight++;
275             }
276
277             if (peer->current_weight > best->current_weight) {
278                 best = peer;
279                 p = i;
280             }
281         }
282     }
283
284     best->current_weight -= total;
285
286     if (now - best->checked > best->fail_timeout) {
287         best->checked = now;
288     }
289
290     pc->sockaddr = best->sockaddr;
291     pc->socklen = best->socklen;
292     pc->name = &best->name;
293
294     lcp->rrp.current = p;
295
296     n = p / (8 * sizeof(uintptr_t));
297     m = (uintptr_t) 1 << p % (8 * sizeof(uintptr_t));
298
299     lcp->rrp.tried[n] |= m;
300     lcp->conns[p]++;
301
302     return NGX_OK;
303
304 failed:
305
306     if (peers->next) {
307         ngx_log_debug0(NGX_LOG_DEBUG_HTTP, pc->log, 0,
308                        "get least conn peer, backup servers");
309
310         lcp->conns += peers->number;
311
312         lcp->rrp.peers = peers->next;
313
314         n = (lcp->rrp.peers->number + (8 * sizeof(uintptr_t) - 1))
315                 / (8 * sizeof(uintptr_t));
316
317         for (i = 0; i < n; i++) {
318              lcp->rrp.tried[i] = 0;
319         }
320
321         rc = ngx_http_upstream_get_least_conn_peer(pc, lcp);
322
323         if (rc != NGX_BUSY) {
324             return rc;
325         }
326     }
327
328     /* all peers failed, mark them as live for quick recovery */
329
330     for (i = 0; i < peers->number; i++) {
331         peers->peer[i].fails = 0;
332     }
333
334     pc->name = peers->name;
335
336     return NGX_BUSY;
337 }
```

```
338
339
340 static void
341 ngx_http_upstream_free_least_conn_peer(ngx_peer_connection_t *pc,
342     void *data, ngx_uint_t state)
343 {
344     ngx_http_upstream_lc_peer_data_t  *lcp = data;
345
346     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, pc->log, 0,
347                    "free least conn peer %ui %ui", pc->tries, state);
348
349     if (lcp->rrp.peers->single) {
350         lcp->free_rr_peer(pc, &lcp->rrp, state);
351         return;
352     }
353
354     lcp->conns[lcp->rrp.current]--;
355
356     lcp->free_rr_peer(pc, &lcp->rrp, state);
357 }
358
359
360 static void *
361 ngx_http_upstream_least_conn_create_conf(ngx_conf_t *cf)
362 {
363     ngx_http_upstream_least_conn_conf_t  *conf;
364
365     conf = ngx_pcalloc(cf->pool,
366                        sizeof(ngx_http_upstream_least_conn_conf_t));
367     if (conf == NULL) {
368         return NULL;
369     }
370
371     /*
372      * set by ngx_pcalloc():
373      *
374      *     conf->conns = NULL;
375      */
376
377     return conf;
378 }
379
380
381 static char *
382 ngx_http_upstream_least_conn(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
383 {
384     ngx_http_upstream_srv_conf_t  *uscf;
385
386     uscf = ngx_http_conf_get_module_srv_conf(cf, ngx_http_upstream_module);
387
388     if (uscf->peer.init_upstream) {
389         ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
390                            "load balancing method redefined");
391     }
392
393     uscf->peer.init_upstream = ngx_http_upstream_init_least_conn;
394
395     uscf->flags = NGX_HTTP_UPSTREAM_CREATE
396                   |NGX_HTTP_UPSTREAM_WEIGHT
397                   |NGX_HTTP_UPSTREAM_MAX_FAILS
398                   |NGX_HTTP_UPSTREAM_FAIL_TIMEOUT
399                   |NGX_HTTP_UPSTREAM_DOWN
400                   |NGX_HTTP_UPSTREAM_BACKUP;
401
402     return NGX_CONF_OK;
403 }
```

One Level Up Top Level

src/http/modules/ngx_http_uwsgi_module.c - nginx-1.7.10

Global variables defined

- ngx_http_uwsgi_cache_headers
- ngx_http_uwsgi_commands
- ngx_http_uwsgi_hide_headers
- ngx_http_uwsgi_modifier_bounds
- ngx_http_uwsgi_module
- ngx_http_uwsgi_module
- ngx_http_uwsgi_module_ctx
- ngx_http_uwsgi_next_upstream_masks
- ngx_http_uwsgi_ssl_protocols
- ngx_http_uwsgi_temp_path

Data types defined

- ngx_http_uwsgi_loc_conf_t
- ngx_http_uwsgi_main_conf_t
- ngx_http_uwsgi_params_t

Functions defined

- ngx_http_uwsgi_abort_request
- ngx_http_uwsgi_cache
- ngx_http_uwsgi_cache_key
- ngx_http_uwsgi_create_key
- ngx_http_uwsgi_create_loc_conf
- ngx_http_uwsgi_create_main_conf
- ngx_http_uwsgi_create_request
- ngx_http_uwsgi_eval
- ngx_http_uwsgi_finalize_request
- ngx_http_uwsgi_handler
- ngx_http_uwsgi_init_params
- ngx_http_uwsgi_merge_loc_conf
- ngx_http_uwsgi_pass
- ngx_http_uwsgi_process_header

- ngx_http_uwsgi_process_status_line
- ngx_http_uwsgi_reinit_request
- ngx_http_uwsgi_set_ssl
- ngx_http_uwsgi_ssl_password_file
- ngx_http_uwsgi_store

Source code

```
1
2  /*
3  * Copyright (C) Unbit S.a.s. 2009-2010
4  * Copyright (C) 2008 Manlio Perillo (manlio.perillo@gmail.com)
5  * Copyright (C) Igor Sysoev
6  * Copyright (C) Nginx, Inc.
7  */
8
9
10 #include <ngx_config.h>
11 #include <ngx_core.h>
12 #include <ngx_http.h>
13
14
15 typedef struct {
16     ngx_array_t                caches;  /* ngx_http_file_cache_t * */
17 } ngx_http_uwsgi_main_conf_t;
18
19
20 typedef struct {
21     ngx_array_t               *flushes;
22     ngx_array_t               *lengths;
23     ngx_array_t               *values;
24     ngx_uint_t                 number;
25     ngx_hash_t                 hash;
26 } ngx_http_uwsgi_params_t;
27
28
29 typedef struct {
30     ngx_http_upstream_conf_t   upstream;
31
32     ngx_http_uwsgi_params_t    params;
33 #if (NGX_HTTP_CACHE)
34     ngx_http_uwsgi_params_t    params_cache;
35 #endif
36     ngx_array_t               *params_source;
37
38     ngx_array_t               *uwsgi_lengths;
39     ngx_array_t               *uwsgi_values;
40
41 #if (NGX_HTTP_CACHE)
42     ngx_http_complex_value_t   cache_key;
43 #endif
44
45     ngx_str_t                  uwsgi_string;
46
47     ngx_uint_t                 modifier1;
48     ngx_uint_t                 modifier2;
49
50 #if (NGX_HTTP_SSL)
51     ngx_uint_t                 ssl;
52     ngx_uint_t                 ssl_protocols;
53     ngx_str_t                  ssl_ciphers;
54     ngx_uint_t                 ssl_verify_depth;
55     ngx_str_t                  ssl_trusted_certificate;
56     ngx_str_t                  ssl_crl;
57     ngx_str_t                  ssl_certificate;
58     ngx_str_t                  ssl_certificate_key;
59     ngx_array_t               *ssl_passwords;
60 #endif
61 } ngx_http_uwsgi_loc_conf_t;
```

```
62
63
64  static ngx_int_t ngx_http_uwsgi_eval(ngx_http_request_t *r,
65      ngx_http_uwsgi_loc_conf_t *uwcf);
66  static ngx_int_t ngx_http_uwsgi_create_request(ngx_http_request_t *r);
67  static ngx_int_t ngx_http_uwsgi_reinit_request(ngx_http_request_t *r);
68  static ngx_int_t ngx_http_uwsgi_process_status_line(ngx_http_request_t *r);
69  static ngx_int_t ngx_http_uwsgi_process_header(ngx_http_request_t *r);
70  static void ngx_http_uwsgi_abort_request(ngx_http_request_t *r);
71  static void ngx_http_uwsgi_finalize_request(ngx_http_request_t *r,
72      ngx_int_t rc);
73
74  static void *ngx_http_uwsgi_create_main_conf(ngx_conf_t *cf);
75  static void *ngx_http_uwsgi_create_loc_conf(ngx_conf_t *cf);
76  static char *ngx_http_uwsgi_merge_loc_conf(ngx_conf_t *cf, void *parent,
77      void *child);
78  static ngx_int_t ngx_http_uwsgi_init_params(ngx_conf_t *cf,
79      ngx_http_uwsgi_loc_conf_t *conf, ngx_http_uwsgi_params_t *params,
80      ngx_keyval_t *default_params);
81
82  static char *ngx_http_uwsgi_pass(ngx_conf_t *cf, ngx_command_t *cmd,
83      void *conf);
84  static char *ngx_http_uwsgi_store(ngx_conf_t *cf, ngx_command_t *cmd,
85      void *conf);
86
87  #if (NGX_HTTP_CACHE)
88  static ngx_int_t ngx_http_uwsgi_create_key(ngx_http_request_t *r);
89  static char *ngx_http_uwsgi_cache(ngx_conf_t *cf, ngx_command_t *cmd,
90      void *conf);
91  static char *ngx_http_uwsgi_cache_key(ngx_conf_t *cf, ngx_command_t *cmd,
92      void *conf);
93  #endif
94
95  #if (NGX_HTTP_SSL)
96  static char *ngx_http_uwsgi_ssl_password_file(ngx_conf_t *cf,
97      ngx_command_t *cmd, void *conf);
98  static ngx_int_t ngx_http_uwsgi_set_ssl(ngx_conf_t *cf,
99      ngx_http_uwsgi_loc_conf_t *uwcf);
100 #endif
101
102
103 static ngx_conf_num_bounds_t  ngx_http_uwsgi_modifier_bounds = {
104     ngx_conf_check_num_bounds, 0, 255
105 };
106
107
108 static ngx_conf_bitmask_t ngx_http_uwsgi_next_upstream_masks[] = {
109     { ngx_string("error"), NGX_HTTP_UPSTREAM_FT_ERROR },
110     { ngx_string("timeout"), NGX_HTTP_UPSTREAM_FT_TIMEOUT },
111     { ngx_string("invalid_header"), NGX_HTTP_UPSTREAM_FT_INVALID_HEADER },
112     { ngx_string("http_500"), NGX_HTTP_UPSTREAM_FT_HTTP_500 },
113     { ngx_string("http_503"), NGX_HTTP_UPSTREAM_FT_HTTP_503 },
114     { ngx_string("http_403"), NGX_HTTP_UPSTREAM_FT_HTTP_403 },
115     { ngx_string("http_404"), NGX_HTTP_UPSTREAM_FT_HTTP_404 },
116     { ngx_string("updating"), NGX_HTTP_UPSTREAM_FT_UPDATING },
117     { ngx_string("off"), NGX_HTTP_UPSTREAM_FT_OFF },
118     { ngx_null_string, 0 }
119 };
120
121
122 #if (NGX_HTTP_SSL)
123
124 static ngx_conf_bitmask_t  ngx_http_uwsgi_ssl_protocols[] = {
125     { ngx_string("SSLv2"), NGX_SSL_SSLv2 },
126     { ngx_string("SSLv3"), NGX_SSL_SSLv3 },
127     { ngx_string("TLSv1"), NGX_SSL_TLSv1 },
128     { ngx_string("TLSv1.1"), NGX_SSL_TLSv1_1 },
129     { ngx_string("TLSv1.2"), NGX_SSL_TLSv1_2 },
130     { ngx_null_string, 0 }
131 };
132
133 #endif
134
135
136 ngx_module_t  ngx_http_uwsgi_module;
137
```

```
138
139 static ngx_command_t ngx_http_uwsgi_commands[] = {
140
141     { ngx_string("uwsgi_pass"),
142       NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
143       ngx_http_uwsgi_pass,
144       NGX_HTTP_LOC_CONF_OFFSET,
145       0,
146       NULL },
147
148     { ngx_string("uwsgi_modifier1"),
149       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
150       ngx_conf_set_num_slot,
151       NGX_HTTP_LOC_CONF_OFFSET,
152       offsetof(ngx_http_uwsgi_loc_conf_t, modifier1),
153       &ngx_http_uwsgi_modifier_bounds },
154
155     { ngx_string("uwsgi_modifier2"),
156       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
157       ngx_conf_set_num_slot,
158       NGX_HTTP_LOC_CONF_OFFSET,
159       offsetof(ngx_http_uwsgi_loc_conf_t, modifier2),
160       &ngx_http_uwsgi_modifier_bounds },
161
162     { ngx_string("uwsgi_store"),
163       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
164       ngx_http_uwsgi_store,
165       NGX_HTTP_LOC_CONF_OFFSET,
166       0,
167       NULL },
168
169     { ngx_string("uwsgi_store_access"),
170       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE123,
171       ngx_conf_set_access_slot,
172       NGX_HTTP_LOC_CONF_OFFSET,
173       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.store_access),
174       NULL },
175
176     { ngx_string("uwsgi_buffering"),
177       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
178       ngx_conf_set_flag_slot,
179       NGX_HTTP_LOC_CONF_OFFSET,
180       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.buffering),
181       NULL },
182
183     { ngx_string("uwsgi_ignore_client_abort"),
184       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
185       ngx_conf_set_flag_slot,
186       NGX_HTTP_LOC_CONF_OFFSET,
187       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.ignore_client_abort),
188       NULL },
189
190     { ngx_string("uwsgi_bind"),
191       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
192       ngx_http_upstream_bind_set_slot,
193       NGX_HTTP_LOC_CONF_OFFSET,
194       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.local),
195       NULL },
196
197     { ngx_string("uwsgi_connect_timeout"),
198       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
199       ngx_conf_set_msec_slot,
200       NGX_HTTP_LOC_CONF_OFFSET,
201       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.connect_timeout),
202       NULL },
203
204     { ngx_string("uwsgi_send_timeout"),
205       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
206       ngx_conf_set_msec_slot,
207       NGX_HTTP_LOC_CONF_OFFSET,
208       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.send_timeout),
209       NULL },
210
211     { ngx_string("uwsgi_buffer_size"),
212       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
213       ngx_conf_set_size_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.buffer_size),
      NULL },

    { ngx_string("uwsgi_pass_request_headers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.pass_request_headers),
      NULL },

    { ngx_string("uwsgi_pass_request_body"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.pass_request_body),
      NULL },

    { ngx_string("uwsgi_intercept_errors"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.intercept_errors),
      NULL },

    { ngx_string("uwsgi_read_timeout"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.read_timeout),
      NULL },

    { ngx_string("uwsgi_buffers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_conf_set_bufs_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.bufs),
      NULL },

    { ngx_string("uwsgi_busy_buffers_size"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.busy_buffers_size_conf),
      NULL },

    { ngx_string("uwsgi_force_ranges"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.force_ranges),
      NULL },

    { ngx_string("uwsgi_limit_rate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_size_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.limit_rate),
      NULL },

#if (NGX_HTTP_CACHE)

    { ngx_string("uwsgi_cache"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_uwsgi_cache,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("uwsgi_cache_key"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_uwsgi_cache_key,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

```

```
290     { ngx_string("uwsgi_cache_path"),
291       NGX_HTTP_MAIN_CONF|NGX_CONF_2MORE,
292       ngx_http_file_cache_set_slot,
293       NGX_HTTP_MAIN_CONF_OFFSET,
294       offsetof(ngx_http_uwsgi_main_conf_t, caches),
295       &ngx_http_uwsgi_module },
296
297     { ngx_string("uwsgi_cache_bypass"),
298       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
299       ngx_http_set_predicate_slot,
300       NGX_HTTP_LOC_CONF_OFFSET,
301       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_bypass),
302       NULL },
303
304     { ngx_string("uwsgi_no_cache"),
305       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
306       ngx_http_set_predicate_slot,
307       NGX_HTTP_LOC_CONF_OFFSET,
308       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.no_cache),
309       NULL },
310
311     { ngx_string("uwsgi_cache_valid"),
312       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
313       ngx_http_file_cache_valid_set_slot,
314       NGX_HTTP_LOC_CONF_OFFSET,
315       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_valid),
316       NULL },
317
318     { ngx_string("uwsgi_cache_min_uses"),
319       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
320       ngx_conf_set_num_slot,
321       NGX_HTTP_LOC_CONF_OFFSET,
322       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_min_uses),
323       NULL },
324
325     { ngx_string("uwsgi_cache_use_stale"),
326       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
327       ngx_conf_set_bitmask_slot,
328       NGX_HTTP_LOC_CONF_OFFSET,
329       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_use_stale),
330       &ngx_http_uwsgi_next_upstream_masks },
331
332     { ngx_string("uwsgi_cache_methods"),
333       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
334       ngx_conf_set_bitmask_slot,
335       NGX_HTTP_LOC_CONF_OFFSET,
336       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_methods),
337       &ngx_http_upstream_cache_method_mask },
338
339     { ngx_string("uwsgi_cache_lock"),
340       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
341       ngx_conf_set_flag_slot,
342       NGX_HTTP_LOC_CONF_OFFSET,
343       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_lock),
344       NULL },
345
346     { ngx_string("uwsgi_cache_lock_timeout"),
347       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
348       ngx_conf_set_msec_slot,
349       NGX_HTTP_LOC_CONF_OFFSET,
350       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_lock_timeout),
351       NULL },
352
353     { ngx_string("uwsgi_cache_lock_age"),
354       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
355       ngx_conf_set_msec_slot,
356       NGX_HTTP_LOC_CONF_OFFSET,
357       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_lock_age),
358       NULL },
359
360     { ngx_string("uwsgi_cache_revalidate"),
361       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
362       ngx_conf_set_flag_slot,
363       NGX_HTTP_LOC_CONF_OFFSET,
364       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.cache_revalidate),
365       NULL },
```

```
366
367 #endif
368
369     { ngx_string("uwsgi_temp_path"),
370       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1234,
371       ngx_conf_set_path_slot,
372       NGX_HTTP_LOC_CONF_OFFSET,
373       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.temp_path),
374       NULL },
375
376     { ngx_string("uwsgi_max_temp_file_size"),
377       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
378       ngx_conf_set_size_slot,
379       NGX_HTTP_LOC_CONF_OFFSET,
380       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.max_temp_file_size_conf),
381       NULL },
382
383     { ngx_string("uwsgi_temp_file_write_size"),
384       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
385       ngx_conf_set_size_slot,
386       NGX_HTTP_LOC_CONF_OFFSET,
387       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.temp_file_write_size_conf),
388       NULL },
389
390     { ngx_string("uwsgi_next_upstream"),
391       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
392       ngx_conf_set_bitmask_slot,
393       NGX_HTTP_LOC_CONF_OFFSET,
394       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.next_upstream),
395       &ngx_http_uwsgi_next_upstream_masks },
396
397     { ngx_string("uwsgi_next_upstream_tries"),
398       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
399       ngx_conf_set_num_slot,
400       NGX_HTTP_LOC_CONF_OFFSET,
401       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.next_upstream_tries),
402       NULL },
403
404     { ngx_string("uwsgi_next_upstream_timeout"),
405       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
406       ngx_conf_set_msec_slot,
407       NGX_HTTP_LOC_CONF_OFFSET,
408       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.next_upstream_timeout),
409       NULL },
410
411     { ngx_string("uwsgi_param"),
412       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE23,
413       ngx_http_upstream_param_set_slot,
414       NGX_HTTP_LOC_CONF_OFFSET,
415       offsetof(ngx_http_uwsgi_loc_conf_t, params_source),
416       NULL },
417
418     { ngx_string("uwsgi_string"),
419       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
420       ngx_conf_set_str_slot,
421       NGX_HTTP_LOC_CONF_OFFSET,
422       offsetof(ngx_http_uwsgi_loc_conf_t, uwsgi_string),
423       NULL },
424
425     { ngx_string("uwsgi_pass_header"),
426       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
427       ngx_conf_set_str_array_slot,
428       NGX_HTTP_LOC_CONF_OFFSET,
429       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.pass_headers),
430       NULL },
431
432     { ngx_string("uwsgi_hide_header"),
433       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
434       ngx_conf_set_str_array_slot,
435       NGX_HTTP_LOC_CONF_OFFSET,
436       offsetof(ngx_http_uwsgi_loc_conf_t, upstream.hide_headers),
437       NULL },
438
439     { ngx_string("uwsgi_ignore_headers"),
440       NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
441       ngx_conf_set_bitmask_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.ignore_headers),
      &ngx_http_upstream_ignore_headers_masks },

#if (NGX_HTTP_SSL)

    { ngx_string("uwsgi_ssl_session_reuse"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.ssl_session_reuse),
      NULL },

    { ngx_string("uwsgi_ssl_protocols"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_protocols),
      &ngx_http_uwsgi_ssl_protocols },

    { ngx_string("uwsgi_ssl_ciphers"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_ciphers),
      NULL },

    { ngx_string("uwsgi_ssl_name"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_set_complex_value_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.ssl_name),
      NULL },

    { ngx_string("uwsgi_ssl_server_name"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.ssl_server_name),
      NULL },

    { ngx_string("uwsgi_ssl_verify"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, upstream.ssl_verify),
      NULL },

    { ngx_string("uwsgi_ssl_verify_depth"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_num_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_verify_depth),
      NULL },

    { ngx_string("uwsgi_ssl_trusted_certificate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_trusted_certificate),
      NULL },

    { ngx_string("uwsgi_ssl_crl"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_crl),
      NULL },

    { ngx_string("uwsgi_ssl_certificate"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_certificate),
      NULL },
```

```
    { ngx_string("uwsgi_ssl_certificate_key"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_uwsgi_loc_conf_t, ssl_certificate_key),
      NULL },

    { ngx_string("uwsgi_ssl_password_file"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_uwsgi_ssl_password_file,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

#endif

      ngx_null_command
};


static ngx_http_module_t ngx_http_uwsgi_module_ctx = {
    NULL,                                  /* preconfiguration */
    NULL,                                  /* postconfiguration */

    ngx_http_uwsgi_create_main_conf,       /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_uwsgi_create_loc_conf,        /* create location configuration */
    ngx_http_uwsgi_merge_loc_conf          /* merge location configuration */
};


ngx_module_t ngx_http_uwsgi_module = {
    NGX_MODULE_V1,
    &ngx_http_uwsgi_module_ctx,            /* module context */
    ngx_http_uwsgi_commands,               /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_str_t ngx_http_uwsgi_hide_headers[] = {
    ngx_string("X-Accel-Expires"),
    ngx_string("X-Accel-Redirect"),
    ngx_string("X-Accel-Limit-Rate"),
    ngx_string("X-Accel-Buffering"),
    ngx_string("X-Accel-Charset"),
    ngx_null_string
};


#if (NGX_HTTP_CACHE)

static ngx_keyval_t  ngx_http_uwsgi_cache_headers[] = {
    { ngx_string("HTTP_IF_MODIFIED_SINCE"),
      ngx_string("$upstream_cache_last_modified") },
    { ngx_string("HTTP_IF_UNMODIFIED_SINCE"), ngx_string("") },
    { ngx_string("HTTP_IF_NONE_MATCH"), ngx_string("$upstream_cache_etag") },
    { ngx_string("HTTP_IF_MATCH"), ngx_string("") },
    { ngx_string("HTTP_RANGE"), ngx_string("") },
    { ngx_string("HTTP_IF_RANGE"), ngx_string("") },
    { ngx_null_string, ngx_null_string }
};

#endif
```

```
594
595 static ngx_path_init_t ngx_http_uwsgi_temp_path = {
596     ngx_string(NGX_HTTP_UWSGI_TEMP_PATH), { 1, 2, 0 }
597 };
598
599
600 static ngx_int_t
601 ngx_http_uwsgi_handler(ngx_http_request_t *r)
602 {
603     ngx_int_t                    rc;
604     ngx_http_status_t           *status;
605     ngx_http_upstream_t         *u;
606     ngx_http_uwsgi_loc_conf_t   *uwcf;
607 #if (NGX_HTTP_CACHE)
608     ngx_http_uwsgi_main_conf_t  *uwmcf;
609 #endif
610
611     if (ngx_http_upstream_create(r) != NGX_OK) {
612         return NGX_HTTP_INTERNAL_SERVER_ERROR;
613     }
614
615     status = ngx_pcalloc(r->pool, sizeof(ngx_http_status_t));
616     if (status == NULL) {
617         return NGX_HTTP_INTERNAL_SERVER_ERROR;
618     }
619
620     ngx_http_set_ctx(r, status, ngx_http_uwsgi_module);
621
622     uwcf = ngx_http_get_module_loc_conf(r, ngx_http_uwsgi_module);
623
624     u = r->upstream;
625
626     if (uwcf->uwsgi_lengths == NULL) {
627
628 #if (NGX_HTTP_SSL)
629         u->ssl = (uwcf->upstream.ssl != NULL);
630
631         if (u->ssl) {
632             ngx_str_set(&u->schema, "suwsgi://");
633
634         } else {
635             ngx_str_set(&u->schema, "uwsgi://");
636         }
637 #else
638         ngx_str_set(&u->schema, "uwsgi://");
639 #endif
640
641     } else {
642         if (ngx_http_uwsgi_eval(r, uwcf) != NGX_OK) {
643             return NGX_HTTP_INTERNAL_SERVER_ERROR;
644         }
645     }
646
647     u->output.tag = (ngx_buf_tag_t) &ngx_http_uwsgi_module;
648
649     u->conf = &uwcf->upstream;
650
651 #if (NGX_HTTP_CACHE)
652     uwmcf = ngx_http_get_module_main_conf(r, ngx_http_uwsgi_module);
653
654     u->caches = &uwmcf->caches;
655     u->create_key = ngx_http_uwsgi_create_key;
656 #endif
657
658     u->create_request = ngx_http_uwsgi_create_request;
659     u->reinit_request = ngx_http_uwsgi_reinit_request;
660     u->process_header = ngx_http_uwsgi_process_status_line;
661     u->abort_request = ngx_http_uwsgi_abort_request;
662     u->finalize_request = ngx_http_uwsgi_finalize_request;
663     r->state = 0;
664
665     u->buffering = uwcf->upstream.buffering;
666
667     u->pipe = ngx_pcalloc(r->pool, sizeof(ngx_event_pipe_t));
668     if (u->pipe == NULL) {
669         return NGX_HTTP_INTERNAL_SERVER_ERROR;
```

```
    }

    u->pipe->input_filter = ngx_event_pipe_copy_input_filter;
    u->pipe->input_ctx = r;

    rc = ngx_http_read_client_request_body(r, ngx_http_upstream_init);

    if (rc >= NGX_HTTP_SPECIAL_RESPONSE) {
        return rc;
    }

    return NGX_DONE;
}


static ngx_int_t
ngx_http_uwsgi_eval(ngx_http_request_t *r, ngx_http_uwsgi_loc_conf_t * uwcf)
{
    size_t                add;
    ngx_url_t             url;
    ngx_http_upstream_t  *u;

    ngx_memzero(&url, sizeof(ngx_url_t));

    if (ngx_http_script_run(r, &url.url, uwcf->uwsgi_lengths->elts, 0,
                            uwcf->uwsgi_values->elts)
        == NULL)
    {
        return NGX_ERROR;
    }

    if (url.url.len > 8
        && ngx_strncasecmp(url.url.data, (u_char *) "uwsgi://", 8) == 0)
    {
        add = 8;

    } else if (url.url.len > 9
               && ngx_strncasecmp(url.url.data, (u_char *) "suwsgi://", 9) == 0)
    {

#if (NGX_HTTP_SSL)
        add = 9;
        r->upstream->ssl = 1;
#else
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "suwsgi protocol requires SSL support");
        return NGX_ERROR;
#endif

    } else {
        add = 0;
    }

    u = r->upstream;

    if (add) {
        u->schema.len = add;
        u->schema.data = url.url.data;

        url.url.data += add;
        url.url.len -= add;

    } else {
        ngx_str_set(&u->schema, "uwsgi://");
    }

    url.no_resolve = 1;

    if (ngx_parse_url(r->pool, &url) != NGX_OK) {
        if (url.err) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "%s in upstream \"%V\"", url.err, &url.url);
        }

        return NGX_ERROR;
    }
```

```
    u->resolved = ngx_pcalloc(r->pool, sizeof(ngx_http_upstream_resolved_t));
    if (u->resolved == NULL) {
        return NGX_ERROR;
    }

    if (url.addrs && url.addrs[0].sockaddr) {
        u->resolved->sockaddr = url.addrs[0].sockaddr;
        u->resolved->socklen = url.addrs[0].socklen;
        u->resolved->naddrs = 1;
        u->resolved->host = url.addrs[0].name;

    } else {
        u->resolved->host = url.host;
        u->resolved->port = url.port;
        u->resolved->no_port = url.no_port;
    }

    return NGX_OK;
}


#if (NGX_HTTP_CACHE)

static ngx_int_t
ngx_http_uwsgi_create_key(ngx_http_request_t *r)
{
    ngx_str_t                  *key;
    ngx_http_uwsgi_loc_conf_t  *uwcf;

    key = ngx_array_push(&r->cache->keys);
    if (key == NULL) {
        return NGX_ERROR;
    }

    uwcf = ngx_http_get_module_loc_conf(r, ngx_http_uwsgi_module);

    if (ngx_http_complex_value(r, &uwcf->cache_key, key) != NGX_OK) {
        return NGX_ERROR;
    }

    return NGX_OK;
}

#endif


static ngx_int_t
ngx_http_uwsgi_create_request(ngx_http_request_t *r)
{
    u_char                        ch, *lowcase_key;
    size_t                        key_len, val_len, len, allocated;
    ngx_uint_t                    i, n, hash, skip_empty, header_params;
    ngx_buf_t                    *b;
    ngx_chain_t                  *cl, *body;
    ngx_list_part_t              *part;
    ngx_table_elt_t              *header, **ignored;
    ngx_http_uwsgi_params_t      *params;
    ngx_http_script_code_pt       code;
    ngx_http_script_engine_t      e, le;
    ngx_http_uwsgi_loc_conf_t    *uwcf;
    ngx_http_script_len_code_pt   lcode;

    len = 0;
    header_params = 0;
    ignored = NULL;

    uwcf = ngx_http_get_module_loc_conf(r, ngx_http_uwsgi_module);

#if (NGX_HTTP_CACHE)
    params = r->upstream->cacheable ? &uwcf->params_cache : &uwcf->params;
#else
    params = &uwcf->params;
#endif

    if (params->lengths) {
```

```
        ngx_memzero(&le, sizeof(ngx_http_script_engine_t));

        ngx_http_script_flush_no_cacheable_variables(r, params->flushes);
        le.flushed = 1;

        le.ip = params->lengths->elts;
        le.request = r;

        while (*(uintptr_t *) le.ip) {

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            key_len = lcode(&le);

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            skip_empty = lcode(&le);

            for (val_len = 0; *(uintptr_t *) le.ip; val_len += lcode (&le)) {
                lcode = *(ngx_http_script_len_code_pt *) le.ip;
            }
            le.ip += sizeof(uintptr_t);

            if (skip_empty && val_len == 0) {
                continue;
            }

            len += 2 + key_len + 2 + val_len;
        }
    }

    if (uwcf->upstream.pass_request_headers) {

        allocated = 0;
        lowcase_key = NULL;

        if (params->number) {
            n = 0;
            part = &r->headers_in.headers.part;

            while (part) {
                n += part->nelts;
                part = part->next;
            }

            ignored = ngx_palloc(r->pool, n * sizeof(void *));
            if (ignored == NULL) {
                return NGX_ERROR;
            }
        }

        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */ ; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            if (params->number) {
                if (allocated < header[i].key.len) {
                    allocated = header[i].key.len + 16;
                    lowcase_key = ngx_pnalloc(r->pool, allocated);
                    if (lowcase_key == NULL) {
                        return NGX_ERROR;
                    }
                }

                hash = 0;

                for (n = 0; n < header[i].key.len; n++) {
```

```
                    ch = header[i].key.data[n];

                    if (ch >= 'A' && ch <= 'Z') {
                        ch |= 0x20;

                    } else if (ch == '-') {
                        ch = '_';
                    }

                    hash = ngx_hash(hash, ch);
                    lowcase_key[n] = ch;
                }

                if (ngx_hash_find(&params->hash, hash, lowcase_key, n)) {
                    ignored[header_params++] = &header[i];
                    continue;
                }
            }

            len += 2 + sizeof("HTTP_") - 1 + header[i].key.len
                 + 2 + header[i].value.len;
        }
    }

    len += uwcf->uwsgi_string.len;

#if 0
    /* allow custom uwsgi packet */
    if (len > 0 && len < 2) {
        ngx_log_error (NGX_LOG_ALERT, r->connection->log, 0,
                       "uwsgi request is too little: %uz", len);
        return NGX_ERROR;
    }
#endif

    b = ngx_create_temp_buf(r->pool, len + 4);
    if (b == NULL) {
        return NGX_ERROR;
    }

    cl = ngx_alloc_chain_link(r->pool);
    if (cl == NULL) {
        return NGX_ERROR;
    }

    cl->buf = b;

    *b->last++ = (u_char) uwcf->modifier1;
    *b->last++ = (u_char) (len & 0xff);
    *b->last++ = (u_char) ((len >> 8) & 0xff);
    *b->last++ = (u_char) uwcf->modifier2;

    if (params->lengths) {
        ngx_memzero(&e, sizeof(ngx_http_script_engine_t));

        e.ip = params->values->elts;
        e.pos = b->last;
        e.request = r;
        e.flushed = 1;

        le.ip = params->lengths->elts;

        while (*(uintptr_t *) le.ip) {

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            key_len = (u_char) lcode (&le);

            lcode = *(ngx_http_script_len_code_pt *) le.ip;
            skip_empty = lcode(&le);

            for (val_len = 0; *(uintptr_t *) le.ip; val_len += lcode(&le)) {
                lcode = *(ngx_http_script_len_code_pt *) le.ip;
            }
            le.ip += sizeof(uintptr_t);

            if (skip_empty && val_len == 0) {
```

```
                e.skip = 1;

                while (*(uintptr_t *) e.ip) {
                    code = *(ngx_http_script_code_pt *) e.ip;
                    code((ngx_http_script_engine_t *) &e);
                }
                e.ip += sizeof(uintptr_t);

                e.skip = 0;

                continue;
            }

            *e.pos++ = (u_char) (key_len & 0xff);
            *e.pos++ = (u_char) ((key_len >> 8) & 0xff);

            code = *(ngx_http_script_code_pt *) e.ip;
            code((ngx_http_script_engine_t *) & e);

            *e.pos++ = (u_char) (val_len & 0xff);
            *e.pos++ = (u_char) ((val_len >> 8) & 0xff);

            while (*(uintptr_t *) e.ip) {
                code = *(ngx_http_script_code_pt *) e.ip;
                code((ngx_http_script_engine_t *) & e);
            }

             e.ip += sizeof(uintptr_t);

             ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                            "uwsgi param: \"%*s: %*s\"",
                            key_len, e.pos - (key_len + 2 + val_len),
                            val_len, e.pos - val_len);
        }

        b->last = e.pos;
    }

    if (uwcf->upstream.pass_request_headers) {

        part = &r->headers_in.headers.part;
        header = part->elts;

        for (i = 0; /* void */ ; i++) {

            if (i >= part->nelts) {
                if (part->next == NULL) {
                    break;
                }

                part = part->next;
                header = part->elts;
                i = 0;
            }

            for (n = 0; n < header_params; n++) {
                if (&header[i] == ignored[n]) {
                    goto next;
                }
            }

            key_len = sizeof("HTTP_") - 1 + header[i].key.len;
            *b->last++ = (u_char) (key_len & 0xff);
            *b->last++ = (u_char) ((key_len >> 8) & 0xff);

            b->last = ngx_cpymem(b->last, "HTTP_", sizeof("HTTP_") - 1);
            for (n = 0; n < header[i].key.len; n++) {
                ch = header[i].key.data[n];

                if (ch >= 'a' && ch <= 'z') {
                    ch &= ~0x20;

                } else if (ch == '-') {
                    ch = '_';
                }

```

```
                *b->last++ = ch;
            }

            val_len = header[i].value.len;
            *b->last++ = (u_char) (val_len & 0xff);
            *b->last++ = (u_char) ((val_len >> 8) & 0xff);
            b->last = ngx_copy(b->last, header[i].value.data, val_len);

            ngx_log_debug4(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "uwsgi param: \"%*s: %*s\"",
                           key_len, b->last - (key_len + 2 + val_len),
                           val_len, b->last - val_len);
        next:

            continue;
        }
    }

    b->last = ngx_copy(b->last, uwcf->uwsgi_string.data,
                       uwcf->uwsgi_string.len);

    if (uwcf->upstream.pass_request_body) {
        body = r->upstream->request_bufs;
        r->upstream->request_bufs = cl;

        while (body) {
            b = ngx_alloc_buf(r->pool);
            if (b == NULL) {
                return NGX_ERROR;
            }

            ngx_memcpy(b, body->buf, sizeof(ngx_buf_t));

            cl->next = ngx_alloc_chain_link(r->pool);
            if (cl->next == NULL) {
                return NGX_ERROR;
            }

            cl = cl->next;
            cl->buf = b;

            body = body->next;
        }

    } else {
        r->upstream->request_bufs = cl;
    }

    cl->next = NULL;

    return NGX_OK;
}


static ngx_int_t
ngx_http_uwsgi_reinit_request(ngx_http_request_t *r)
{
    ngx_http_status_t  *status;

    status = ngx_http_get_module_ctx(r, ngx_http_uwsgi_module);

    if (status == NULL) {
        return NGX_OK;
    }

    status->code = 0;
    status->count = 0;
    status->start = NULL;
    status->end = NULL;

    r->upstream->process_header = ngx_http_uwsgi_process_status_line;
    r->state = 0;

    return NGX_OK;
}

```

```
1126
1127 static ngx_int_t
1128 ngx_http_uwsgi_process_status_line(ngx_http_request_t *r)
1129 {
1130     size_t                len;
1131     ngx_int_t             rc;
1132     ngx_http_status_t     *status;
1133     ngx_http_upstream_t   *u;
1134
1135     status = ngx_http_get_module_ctx(r, ngx_http_uwsgi_module);
1136
1137     if (status == NULL) {
1138         return NGX_ERROR;
1139     }
1140
1141     u = r->upstream;
1142
1143     rc = ngx_http_parse_status_line(r, &u->buffer, status);
1144
1145     if (rc == NGX_AGAIN) {
1146         return rc;
1147     }
1148
1149     if (rc == NGX_ERROR) {
1150         u->process_header = ngx_http_uwsgi_process_header;
1151         return ngx_http_uwsgi_process_header(r);
1152     }
1153
1154     if (u->state && u->state->status == 0) {
1155         u->state->status = status->code;
1156     }
1157
1158     u->headers_in.status_n = status->code;
1159
1160     len = status->end - status->start;
1161     u->headers_in.status_line.len = len;
1162
1163     u->headers_in.status_line.data = ngx_pnalloc(r->pool, len);
1164     if (u->headers_in.status_line.data == NULL) {
1165         return NGX_ERROR;
1166     }
1167
1168     ngx_memcpy(u->headers_in.status_line.data, status->start, len);
1169
1170     ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
1171                    "http uwsgi status %ui \"%V\"",
1172                    u->headers_in.status_n, &u->headers_in.status_line);
1173
1174     u->process_header = ngx_http_uwsgi_process_header;
1175
1176     return ngx_http_uwsgi_process_header(r);
1177 }
1178
1179
1180 static ngx_int_t
1181 ngx_http_uwsgi_process_header(ngx_http_request_t *r)
1182 {
1183     ngx_str_t                      *status_line;
1184     ngx_int_t                       rc, status;
1185     ngx_table_elt_t                *h;
1186     ngx_http_upstream_t            *u;
1187     ngx_http_upstream_header_t     *hh;
1188     ngx_http_upstream_main_conf_t  *umcf;
1189
1190     umcf = ngx_http_get_module_main_conf(r, ngx_http_upstream_module);
1191
1192     for ( ;; ) {
1193
1194         rc = ngx_http_parse_header_line(r, &r->upstream->buffer, 1);
1195
1196         if (rc == NGX_OK) {
1197
1198             /* a header line has been parsed successfully */
1199
1200             h = ngx_list_push(&r->upstream->headers_in.headers);
1201             if (h == NULL) {
```

```
            return NGX_ERROR;
        }

        h->hash = r->header_hash;

        h->key.len = r->header_name_end - r->header_name_start;
        h->value.len = r->header_end - r->header_start;

        h->key.data = ngx_pnalloc(r->pool,
                                  h->key.len + 1 + h->value.len + 1
                                  + h->key.len);
        if (h->key.data == NULL) {
            return NGX_ERROR;
        }

        h->value.data = h->key.data + h->key.len + 1;
        h->lowcase_key = h->key.data + h->key.len + 1 + h->value.len + 1;

        ngx_memcpy(h->key.data, r->header_name_start, h->key.len);
        h->key.data[h->key.len] = '\0';
        ngx_memcpy(h->value.data, r->header_start, h->value.len);
        h->value.data[h->value.len] = '\0';

        if (h->key.len == r->lowcase_index) {
            ngx_memcpy(h->lowcase_key, r->lowcase_header, h->key.len);

        } else {
            ngx_strlow(h->lowcase_key, h->key.data, h->key.len);
        }

        hh = ngx_hash_find(&umcf->headers_in_hash, h->hash,
                           h->lowcase_key, h->key.len);

        if (hh && hh->handler(r, h, hh->offset) != NGX_OK) {
            return NGX_ERROR;
        }

        ngx_log_debug2(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http uwsgi header: \"%V: %V\"", &h->key, &h->value);

        continue;
    }

    if (rc == NGX_HTTP_PARSE_HEADER_DONE) {

        /* a whole header has been parsed successfully */

        ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                       "http uwsgi header done");

        u = r->upstream;

        if (u->headers_in.status_n) {
            goto done;
        }

        if (u->headers_in.status) {
            status_line = &u->headers_in.status->value;

            status = ngx_atoi(status_line->data, 3);
            if (status == NGX_ERROR) {
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "upstream sent invalid status \"%V\"",
                              status_line);
                return NGX_HTTP_UPSTREAM_INVALID_HEADER;
            }

            u->headers_in.status_n = status;
            u->headers_in.status_line = *status_line;

        } else if (u->headers_in.location) {
            u->headers_in.status_n = 302;
            ngx_str_set(&u->headers_in.status_line,
                        "302 Moved Temporarily");

        } else {
```

```
                u->headers_in.status_n = 200;
                ngx_str_set(&u->headers_in.status_line, "200 OK");
            }

            if (u->state && u->state->status == 0) {
                u->state->status = u->headers_in.status_n;
            }

        done:

            if (u->headers_in.status_n == NGX_HTTP_SWITCHING_PROTOCOLS
                && r->headers_in.upgrade)
            {
                u->upgrade = 1;
            }

            return NGX_OK;
        }

        if (rc == NGX_AGAIN) {
            return NGX_AGAIN;
        }

        /* there was error while a header line parsing */

        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "upstream sent invalid header");

        return NGX_HTTP_UPSTREAM_INVALID_HEADER;
    }
}


static void
ngx_http_uwsgi_abort_request(ngx_http_request_t *r)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "abort http uwsgi request");

    return;
}


static void
ngx_http_uwsgi_finalize_request(ngx_http_request_t *r, ngx_int_t rc)
{
    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "finalize http uwsgi request");

    return;
}


static void *
ngx_http_uwsgi_create_main_conf(ngx_conf_t *cf)
{
    ngx_http_uwsgi_main_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_uwsgi_main_conf_t));
    if (conf == NULL) {
        return NULL;
    }

#if (NGX_HTTP_CACHE)
    if (ngx_array_init(&conf->caches, cf->pool, 4,
                       sizeof(ngx_http_file_cache_t *))
        != NGX_OK)
    {
        return NULL;
    }
#endif

    return conf;
}


```

```
1354 static void *
1355 ngx_http_uwsgi_create_loc_conf(ngx_conf_t *cf)
1356 {
1357     ngx_http_uwsgi_loc_conf_t  *conf;
1358
1359     conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_uwsgi_loc_conf_t));
1360     if (conf == NULL) {
1361         return NULL;
1362     }
1363
1364     conf->modifier1 = NGX_CONF_UNSET_UINT;
1365     conf->modifier2 = NGX_CONF_UNSET_UINT;
1366
1367     conf->upstream.store = NGX_CONF_UNSET;
1368     conf->upstream.store_access = NGX_CONF_UNSET_UINT;
1369     conf->upstream.next_upstream_tries = NGX_CONF_UNSET_UINT;
1370     conf->upstream.buffering = NGX_CONF_UNSET;
1371     conf->upstream.ignore_client_abort = NGX_CONF_UNSET;
1372     conf->upstream.force_ranges = NGX_CONF_UNSET;
1373
1374     conf->upstream.local = NGX_CONF_UNSET_PTR;
1375
1376     conf->upstream.connect_timeout = NGX_CONF_UNSET_MSEC;
1377     conf->upstream.send_timeout = NGX_CONF_UNSET_MSEC;
1378     conf->upstream.read_timeout = NGX_CONF_UNSET_MSEC;
1379     conf->upstream.next_upstream_timeout = NGX_CONF_UNSET_MSEC;
1380
1381     conf->upstream.send_lowat = NGX_CONF_UNSET_SIZE;
1382     conf->upstream.buffer_size = NGX_CONF_UNSET_SIZE;
1383     conf->upstream.limit_rate = NGX_CONF_UNSET_SIZE;
1384
1385     conf->upstream.busy_buffers_size_conf = NGX_CONF_UNSET_SIZE;
1386     conf->upstream.max_temp_file_size_conf = NGX_CONF_UNSET_SIZE;
1387     conf->upstream.temp_file_write_size_conf = NGX_CONF_UNSET_SIZE;
1388
1389     conf->upstream.pass_request_headers = NGX_CONF_UNSET;
1390     conf->upstream.pass_request_body = NGX_CONF_UNSET;
1391
1392 #if (NGX_HTTP_CACHE)
1393     conf->upstream.cache = NGX_CONF_UNSET;
1394     conf->upstream.cache_min_uses = NGX_CONF_UNSET_UINT;
1395     conf->upstream.cache_bypass = NGX_CONF_UNSET_PTR;
1396     conf->upstream.no_cache = NGX_CONF_UNSET_PTR;
1397     conf->upstream.cache_valid = NGX_CONF_UNSET_PTR;
1398     conf->upstream.cache_lock = NGX_CONF_UNSET;
1399     conf->upstream.cache_lock_timeout = NGX_CONF_UNSET_MSEC;
1400     conf->upstream.cache_lock_age = NGX_CONF_UNSET_MSEC;
1401     conf->upstream.cache_revalidate = NGX_CONF_UNSET;
1402 #endif
1403
1404     conf->upstream.hide_headers = NGX_CONF_UNSET_PTR;
1405     conf->upstream.pass_headers = NGX_CONF_UNSET_PTR;
1406
1407     conf->upstream.intercept_errors = NGX_CONF_UNSET;
1408
1409 #if (NGX_HTTP_SSL)
1410     conf->upstream.ssl_session_reuse = NGX_CONF_UNSET;
1411     conf->upstream.ssl_server_name = NGX_CONF_UNSET;
1412     conf->upstream.ssl_verify = NGX_CONF_UNSET;
1413     conf->ssl_verify_depth = NGX_CONF_UNSET_UINT;
1414     conf->ssl_passwords = NGX_CONF_UNSET_PTR;
1415 #endif
1416
1417     /* "uwsgi_cyclic_temp_file" is disabled */
1418     conf->upstream.cyclic_temp_file = 0;
1419
1420     conf->upstream.change_buffering = 1;
1421
1422     ngx_str_set(&conf->upstream.module, "uwsgi");
1423
1424     return conf;
1425 }
1426
1427
1428 static char *
1429 ngx_http_uwsgi_merge_loc_conf(ngx_conf_t *cf, void *parent, void *child)
```

```
{
    ngx_http_uwsgi_loc_conf_t *prev = parent;
    ngx_http_uwsgi_loc_conf_t *conf = child;

    size_t                     size;
    ngx_int_t                  rc;
    ngx_hash_init_t            hash;
    ngx_http_core_loc_conf_t  *clcf;

#if (NGX_HTTP_CACHE)

    if (conf->upstream.store > 0) {
        conf->upstream.cache = 0;
    }

    if (conf->upstream.cache > 0) {
        conf->upstream.store = 0;
    }

#endif

    if (conf->upstream.store == NGX_CONF_UNSET) {
        ngx_conf_merge_value(conf->upstream.store, prev->upstream.store, 0);

        conf->upstream.store_lengths = prev->upstream.store_lengths;
        conf->upstream.store_values = prev->upstream.store_values;
    }

    ngx_conf_merge_uint_value(conf->upstream.store_access,
                              prev->upstream.store_access, 0600);

    ngx_conf_merge_uint_value(conf->upstream.next_upstream_tries,
                              prev->upstream.next_upstream_tries, 0);

    ngx_conf_merge_value(conf->upstream.buffering,
                              prev->upstream.buffering, 1);

    ngx_conf_merge_value(conf->upstream.ignore_client_abort,
                              prev->upstream.ignore_client_abort, 0);

    ngx_conf_merge_value(conf->upstream.force_ranges,
                              prev->upstream.force_ranges, 0);

    ngx_conf_merge_ptr_value(conf->upstream.local,
                              prev->upstream.local, NULL);

    ngx_conf_merge_msec_value(conf->upstream.connect_timeout,
                              prev->upstream.connect_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.send_timeout,
                              prev->upstream.send_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.read_timeout,
                              prev->upstream.read_timeout, 60000);

    ngx_conf_merge_msec_value(conf->upstream.next_upstream_timeout,
                              prev->upstream.next_upstream_timeout, 0);

    ngx_conf_merge_size_value(conf->upstream.send_lowat,
                              prev->upstream.send_lowat, 0);

    ngx_conf_merge_size_value(conf->upstream.buffer_size,
                              prev->upstream.buffer_size,
                              (size_t) ngx_pagesize);

    ngx_conf_merge_size_value(conf->upstream.limit_rate,
                              prev->upstream.limit_rate, 0);


    ngx_conf_merge_bufs_value(conf->upstream.bufs, prev->upstream.bufs,
                              8, ngx_pagesize);

    if (conf->upstream.bufs.num < 2) {
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "there must be at least 2 \"uwsgi_buffers\"");
        return NGX_CONF_ERROR;
```

```
1506      }
1507
1508
1509      size = conf->upstream.buffer_size;
1510      if (size < conf->upstream.bufs.size) {
1511          size = conf->upstream.bufs.size;
1512      }
1513
1514
1515      ngx_conf_merge_size_value(conf->upstream.busy_buffers_size_conf,
1516                                prev->upstream.busy_buffers_size_conf,
1517                                NGX_CONF_UNSET_SIZE);
1518
1519      if (conf->upstream.busy_buffers_size_conf == NGX_CONF_UNSET_SIZE) {
1520          conf->upstream.busy_buffers_size = 2 * size;
1521      } else {
1522          conf->upstream.busy_buffers_size =
1523              conf->upstream.busy_buffers_size_conf;
1524      }
1525
1526      if (conf->upstream.busy_buffers_size < size) {
1527          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1528              "\"uwsgi_busy_buffers_size\" must be equal to or greater "
1529              "than the maximum of the value of \"uwsgi_buffer_size\" and "
1530              "one of the \"uwsgi_buffers\"");
1531
1532          return NGX_CONF_ERROR;
1533      }
1534
1535      if (conf->upstream.busy_buffers_size
1536          > (conf->upstream.bufs.num - 1) * conf->upstream.bufs.size)
1537      {
1538          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1539              "\"uwsgi_busy_buffers_size\" must be less than "
1540              "the size of all \"uwsgi_buffers\" minus one buffer");
1541
1542          return NGX_CONF_ERROR;
1543      }
1544
1545
1546      ngx_conf_merge_size_value(conf->upstream.temp_file_write_size_conf,
1547                                prev->upstream.temp_file_write_size_conf,
1548                                NGX_CONF_UNSET_SIZE);
1549
1550      if (conf->upstream.temp_file_write_size_conf == NGX_CONF_UNSET_SIZE) {
1551          conf->upstream.temp_file_write_size = 2 * size;
1552      } else {
1553          conf->upstream.temp_file_write_size =
1554              conf->upstream.temp_file_write_size_conf;
1555      }
1556
1557      if (conf->upstream.temp_file_write_size < size) {
1558          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
1559              "\"uwsgi_temp_file_write_size\" must be equal to or greater than "
1560              "the maximum of the value of \"uwsgi_buffer_size\" and "
1561              "one of the \"uwsgi_buffers\"");
1562
1563          return NGX_CONF_ERROR;
1564      }
1565
1566
1567      ngx_conf_merge_size_value(conf->upstream.max_temp_file_size_conf,
1568                                prev->upstream.max_temp_file_size_conf,
1569                                NGX_CONF_UNSET_SIZE);
1570
1571      if (conf->upstream.max_temp_file_size_conf == NGX_CONF_UNSET_SIZE) {
1572          conf->upstream.max_temp_file_size = 1024 * 1024 * 1024;
1573      } else {
1574          conf->upstream.max_temp_file_size =
1575              conf->upstream.max_temp_file_size_conf;
1576      }
1577
1578      if (conf->upstream.max_temp_file_size != 0
1579          && conf->upstream.max_temp_file_size < size)
1580      {
1581          ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
```

```
            "\"uwsgi_max_temp_file_size\" must be equal to zero to disable "
            "temporary files usage or must be equal to or greater than "
            "the maximum of the value of \"uwsgi_buffer_size\" and "
            "one of the \"uwsgi_buffers\"");

        return NGX_CONF_ERROR;
    }


    ngx_conf_merge_bitmask_value(conf->upstream.ignore_headers,
                                 prev->upstream.ignore_headers,
                                 NGX_CONF_BITMASK_SET);


    ngx_conf_merge_bitmask_value(conf->upstream.next_upstream,
                                 prev->upstream.next_upstream,
                                 (NGX_CONF_BITMASK_SET
                                  |NGX_HTTP_UPSTREAM_FT_ERROR
                                  |NGX_HTTP_UPSTREAM_FT_TIMEOUT));

    if (conf->upstream.next_upstream & NGX_HTTP_UPSTREAM_FT_OFF) {
        conf->upstream.next_upstream = NGX_CONF_BITMASK_SET
                                       |NGX_HTTP_UPSTREAM_FT_OFF;
    }

    if (ngx_conf_merge_path_value(cf, &conf->upstream.temp_path,
                                  prev->upstream.temp_path,
                                  &ngx_http_uwsgi_temp_path)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

#if (NGX_HTTP_CACHE)

    if (conf->upstream.cache == NGX_CONF_UNSET) {
        ngx_conf_merge_value(conf->upstream.cache,
                             prev->upstream.cache, 0);

        conf->upstream.cache_zone = prev->upstream.cache_zone;
        conf->upstream.cache_value = prev->upstream.cache_value;
    }

    if (conf->upstream.cache_zone && conf->upstream.cache_zone->data == NULL) {
        ngx_shm_zone_t  *shm_zone;

        shm_zone = conf->upstream.cache_zone;

        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "\"uwsgi_cache\" zone \"%V\" is unknown",
                           &shm_zone->shm.name);

        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_uint_value(conf->upstream.cache_min_uses,
                              prev->upstream.cache_min_uses, 1);

    ngx_conf_merge_bitmask_value(conf->upstream.cache_use_stale,
                              prev->upstream.cache_use_stale,
                              (NGX_CONF_BITMASK_SET
                               |NGX_HTTP_UPSTREAM_FT_OFF));

    if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_OFF) {
        conf->upstream.cache_use_stale = NGX_CONF_BITMASK_SET
                                         |NGX_HTTP_UPSTREAM_FT_OFF;
    }

    if (conf->upstream.cache_use_stale & NGX_HTTP_UPSTREAM_FT_ERROR) {
        conf->upstream.cache_use_stale |= NGX_HTTP_UPSTREAM_FT_NOLIVE;
    }

    if (conf->upstream.cache_methods == 0) {
        conf->upstream.cache_methods = prev->upstream.cache_methods;
    }

```

```
1658      conf->upstream.cache_methods |= NGX_HTTP_GET|NGX_HTTP_HEAD;
1659
1660      ngx_conf_merge_ptr_value(conf->upstream.cache_bypass,
1661                               prev->upstream.cache_bypass, NULL);
1662
1663      ngx_conf_merge_ptr_value(conf->upstream.no_cache,
1664                               prev->upstream.no_cache, NULL);
1665
1666      ngx_conf_merge_ptr_value(conf->upstream.cache_valid,
1667                               prev->upstream.cache_valid, NULL);
1668
1669      if (conf->cache_key.value.data == NULL) {
1670          conf->cache_key = prev->cache_key;
1671      }
1672
1673      if (conf->upstream.cache && conf->cache_key.value.data == NULL) {
1674          ngx_conf_log_error(NGX_LOG_WARN, cf, 0,
1675                             "no \"uwsgi_cache_key\" for \"uwsgi_cache\"");
1676      }
1677
1678      ngx_conf_merge_value(conf->upstream.cache_lock,
1679                               prev->upstream.cache_lock, 0);
1680
1681      ngx_conf_merge_msec_value(conf->upstream.cache_lock_timeout,
1682                               prev->upstream.cache_lock_timeout, 5000);
1683
1684      ngx_conf_merge_msec_value(conf->upstream.cache_lock_age,
1685                               prev->upstream.cache_lock_age, 5000);
1686
1687      ngx_conf_merge_value(conf->upstream.cache_revalidate,
1688                               prev->upstream.cache_revalidate, 0);
1689
1690  #endif
1691
1692      ngx_conf_merge_value(conf->upstream.pass_request_headers,
1693                           prev->upstream.pass_request_headers, 1);
1694      ngx_conf_merge_value(conf->upstream.pass_request_body,
1695                           prev->upstream.pass_request_body, 1);
1696
1697      ngx_conf_merge_value(conf->upstream.intercept_errors,
1698                           prev->upstream.intercept_errors, 0);
1699
1700  #if (NGX_HTTP_SSL)
1701
1702      ngx_conf_merge_value(conf->upstream.ssl_session_reuse,
1703                               prev->upstream.ssl_session_reuse, 1);
1704
1705      ngx_conf_merge_bitmask_value(conf->ssl_protocols, prev->ssl_protocols,
1706                                   (NGX_CONF_BITMASK_SET|NGX_SSL_SSLv3
1707                                    |NGX_SSL_TLSv1|NGX_SSL_TLSv1_1
1708                                    |NGX_SSL_TLSv1_2));
1709
1710      ngx_conf_merge_str_value(conf->ssl_ciphers, prev->ssl_ciphers,
1711                               "DEFAULT");
1712
1713      if (conf->upstream.ssl_name == NULL) {
1714          conf->upstream.ssl_name = prev->upstream.ssl_name;
1715      }
1716
1717      ngx_conf_merge_value(conf->upstream.ssl_server_name,
1718                               prev->upstream.ssl_server_name, 0);
1719      ngx_conf_merge_value(conf->upstream.ssl_verify,
1720                               prev->upstream.ssl_verify, 0);
1721      ngx_conf_merge_uint_value(conf->ssl_verify_depth,
1722                               prev->ssl_verify_depth, 1);
1723      ngx_conf_merge_str_value(conf->ssl_trusted_certificate,
1724                               prev->ssl_trusted_certificate, "");
1725      ngx_conf_merge_str_value(conf->ssl_crl, prev->ssl_crl, "");
1726
1727      ngx_conf_merge_str_value(conf->ssl_certificate,
1728                               prev->ssl_certificate, "");
1729      ngx_conf_merge_str_value(conf->ssl_certificate_key,
1730                               prev->ssl_certificate_key, "");
1731      ngx_conf_merge_ptr_value(conf->ssl_passwords, prev->ssl_passwords, NULL);
1732
1733      if (conf->ssl && ngx_http_uwsgi_set_ssl(cf, conf) != NGX_OK) {
```

```
1734         return NGX_CONF_ERROR;
1735     }
1736
1737 #endif
1738
1739     ngx_conf_merge_str_value(conf->uwsgi_string, prev->uwsgi_string, "");
1740
1741     hash.max_size = 512;
1742     hash.bucket_size = ngx_align(64, ngx_cacheline_size);
1743     hash.name = "uwsgi_hide_headers_hash";
1744
1745     if (ngx_http_upstream_hide_headers_hash(cf, &conf->upstream,
1746             &prev->upstream, ngx_http_uwsgi_hide_headers, &hash)
1747         != NGX_OK)
1748     {
1749         return NGX_CONF_ERROR;
1750     }
1751
1752     clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
1753
1754     if (clcf->noname
1755         && conf->upstream.upstream == NULL && conf->uwsgi_lengths == NULL)
1756     {
1757         conf->upstream.upstream = prev->upstream.upstream;
1758
1759         conf->uwsgi_lengths = prev->uwsgi_lengths;
1760         conf->uwsgi_values = prev->uwsgi_values;
1761
1762 #if (NGX_HTTP_SSL)
1763         conf->upstream.ssl = prev->upstream.ssl;
1764 #endif
1765     }
1766
1767     if (clcf->lmt_excpt && clcf->handler == NULL
1768         && (conf->upstream.upstream || conf->uwsgi_lengths))
1769     {
1770         clcf->handler = ngx_http_uwsgi_handler;
1771     }
1772
1773     ngx_conf_merge_uint_value(conf->modifier1, prev->modifier1, 0);
1774     ngx_conf_merge_uint_value(conf->modifier2, prev->modifier2, 0);
1775
1776     if (conf->params_source == NULL) {
1777         conf->params = prev->params;
1778 #if (NGX_HTTP_CACHE)
1779         conf->params_cache = prev->params_cache;
1780 #endif
1781         conf->params_source = prev->params_source;
1782     }
1783
1784     rc = ngx_http_uwsgi_init_params(cf, conf, &conf->params, NULL);
1785     if (rc != NGX_OK) {
1786         return NGX_CONF_ERROR;
1787     }
1788
1789 #if (NGX_HTTP_CACHE)
1790
1791     if (conf->upstream.cache) {
1792         rc = ngx_http_uwsgi_init_params(cf, conf, &conf->params_cache,
1793                                         ngx_http_uwsgi_cache_headers);
1794         if (rc != NGX_OK) {
1795             return NGX_CONF_ERROR;
1796         }
1797     }
1798
1799 #endif
1800
1801     return NGX_CONF_OK;
1802 }
1803
1804
1805 static ngx_int_t
1806 ngx_http_uwsgi_init_params(ngx_conf_t *cf, ngx_http_uwsgi_loc_conf_t *conf,
1807     ngx_http_uwsgi_params_t *params, ngx_keyval_t *default_params)
1808 {
1809     u_char                       *p;
```

```
    size_t                      size;
    uintptr_t                   *code;
    ngx_uint_t                  i, nsrc;
    ngx_array_t                 headers_names, params_merged;
    ngx_keyval_t                *h;
    ngx_hash_key_t              *hk;
    ngx_hash_init_t             hash;
    ngx_http_upstream_param_t   *src, *s;
    ngx_http_script_compile_t   sc;
    ngx_http_script_copy_code_t  *copy;

    if (params->hash.buckets) {
        return NGX_OK;
    }

    if (conf->params_source == NULL && default_params == NULL) {
        params->hash.buckets = (void *) 1;
        return NGX_OK;
    }

    params->lengths = ngx_array_create(cf->pool, 64, 1);
    if (params->lengths == NULL) {
        return NGX_ERROR;
    }

    params->values = ngx_array_create(cf->pool, 512, 1);
    if (params->values == NULL) {
        return NGX_ERROR;
    }

    if (ngx_array_init(&headers_names, cf->temp_pool, 4, sizeof(ngx_hash_key_t))
        != NGX_OK)
    {
        return NGX_ERROR;
    }

    if (conf->params_source) {
        src = conf->params_source->elts;
        nsrc = conf->params_source->nelts;

    } else {
        src = NULL;
        nsrc = 0;
    }

    if (default_params) {
        if (ngx_array_init(&params_merged, cf->temp_pool, 4,
                           sizeof(ngx_http_upstream_param_t))
            != NGX_OK)
        {
            return NGX_ERROR;
        }

        for (i = 0; i < nsrc; i++) {

            s = ngx_array_push(&params_merged);
            if (s == NULL) {
                return NGX_ERROR;
            }

            *s = src[i];
        }

        h = default_params;

        while (h->key.len) {

            src = params_merged.elts;
            nsrc = params_merged.nelts;

            for (i = 0; i < nsrc; i++) {
                if (ngx_strcasecmp(h->key.data, src[i].key.data) == 0) {
                    goto next;
                }
            }

```

```
            s = ngx_array_push(&params_merged);
            if (s == NULL) {
                return NGX_ERROR;
            }

            s->key = h->key;
            s->value = h->value;
            s->skip_empty = 1;

        next:

            h++;
        }

        src = params_merged.elts;
        nsrc = params_merged.nelts;
    }

    for (i = 0; i < nsrc; i++) {

        if (src[i].key.len > sizeof("HTTP_") - 1
            && ngx_strncmp(src[i].key.data, "HTTP_", sizeof("HTTP_") - 1) == 0)
        {
            hk = ngx_array_push(&headers_names);
            if (hk == NULL) {
                return NGX_ERROR;
            }

            hk->key.len = src[i].key.len - 5;
            hk->key.data = src[i].key.data + 5;
            hk->key_hash = ngx_hash_key_lc(hk->key.data, hk->key.len);
            hk->value = (void *) 1;

            if (src[i].value.len == 0) {
                continue;
            }
        }

        copy = ngx_array_push_n(params->lengths,
                                sizeof(ngx_http_script_copy_code_t));
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
        copy->len = src[i].key.len;

        copy = ngx_array_push_n(params->lengths,
                                sizeof(ngx_http_script_copy_code_t));
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = (ngx_http_script_code_pt) ngx_http_script_copy_len_code;
        copy->len = src[i].skip_empty;


        size = (sizeof(ngx_http_script_copy_code_t)
                + src[i].key.len + sizeof(uintptr_t) - 1)
               & ~(sizeof(uintptr_t) - 1);

        copy = ngx_array_push_n(params->values, size);
        if (copy == NULL) {
            return NGX_ERROR;
        }

        copy->code = ngx_http_script_copy_code;
        copy->len = src[i].key.len;

        p = (u_char *) copy + sizeof(ngx_http_script_copy_code_t);
        ngx_memcpy(p, src[i].key.data, src[i].key.len);


        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

        sc.cf = cf;
```

```
        sc.source = &src[i].value;
        sc.flushes = &params->flushes;
        sc.lengths = &params->lengths;
        sc.values = &params->values;

        if (ngx_http_script_compile(&sc) != NGX_OK) {
            return NGX_ERROR;
        }

        code = ngx_array_push_n(params->lengths, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;


        code = ngx_array_push_n(params->values, sizeof(uintptr_t));
        if (code == NULL) {
            return NGX_ERROR;
        }

        *code = (uintptr_t) NULL;
    }

    code = ngx_array_push_n(params->lengths, sizeof(uintptr_t));
    if (code == NULL) {
        return NGX_ERROR;
    }

    *code = (uintptr_t) NULL;

    params->number = headers_names.nelts;

    hash.hash = &params->hash;
    hash.key = ngx_hash_key_lc;
    hash.max_size = 512;
    hash.bucket_size = 64;
    hash.name = "uwsgi_params_hash";
    hash.pool = cf->pool;
    hash.temp_pool = NULL;

    return ngx_hash_init(&hash, headers_names.elts, headers_names.nelts);
}


static char *
ngx_http_uwsgi_pass(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_uwsgi_loc_conf_t *uwcf = conf;

    size_t                     add;
    ngx_url_t                  u;
    ngx_str_t                 *value, *url;
    ngx_uint_t                 n;
    ngx_http_core_loc_conf_t  *clcf;
    ngx_http_script_compile_t  sc;

    if (uwcf->upstream.upstream || uwcf->uwsgi_lengths) {
        return "is duplicate";
    }

    clcf = ngx_http_conf_get_module_loc_conf(cf, ngx_http_core_module);
    clcf->handler = ngx_http_uwsgi_handler;

    value = cf->args->elts;

    url = &value[1];

    n = ngx_http_script_variables_count(url);

    if (n) {

        ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

        sc.cf = cf;
```

```
        sc.source = url;
        sc.lengths = &uwcf->uwsgi_lengths;
        sc.values = &uwcf->uwsgi_values;
        sc.variables = n;
        sc.complete_lengths = 1;
        sc.complete_values = 1;

        if (ngx_http_script_compile(&sc) != NGX_OK) {
            return NGX_CONF_ERROR;
        }

#if (NGX_HTTP_SSL)
        uwcf->ssl = 1;
#endif

        return NGX_CONF_OK;
    }

    if (ngx_strncasecmp(url->data, (u_char *) "uwsgi://", 8) == 0) {
        add = 8;

    } else if (ngx_strncasecmp(url->data, (u_char *) "suwsgi://", 9) == 0) {

#if (NGX_HTTP_SSL)
        add = 9;
        uwcf->ssl = 1;
#else
        ngx_conf_log_error(NGX_LOG_EMERG, cf, 0,
                           "suwsgi protocol requires SSL support");
        return NGX_CONF_ERROR;
#endif

    } else {
        add = 0;
    }

    ngx_memzero(&u, sizeof(ngx_url_t));

    u.url.len = url->len - add;
    u.url.data = url->data + add;
    u.no_resolve = 1;

    uwcf->upstream.upstream = ngx_http_upstream_add(cf, &u, 0);
    if (uwcf->upstream.upstream == NULL) {
        return NGX_CONF_ERROR;
    }

    if (clcf->name.data[clcf->name.len - 1] == '/') {
        clcf->auto_redirect = 1;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_uwsgi_store(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_uwsgi_loc_conf_t *uwcf = conf;

    ngx_str_t                 *value;
    ngx_http_script_compile_t  sc;

    if (uwcf->upstream.store != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    value = cf->args->elts;

    if (ngx_strcmp(value[1].data, "off") == 0) {
        uwcf->upstream.store = 0;
        return NGX_CONF_OK;
    }

#if (NGX_HTTP_CACHE)

```

```
    if (uwcf->upstream.cache > 0) {
        return "is incompatible with \"uwsgi_cache\"";
    }

#endif

    uwcf->upstream.store = 1;

    if (ngx_strcmp(value[1].data, "on") == 0) {
        return NGX_CONF_OK;
    }

    /* include the terminating '\0' into script */
    value[1].len++;

    ngx_memzero(&sc, sizeof(ngx_http_script_compile_t));

    sc.cf = cf;
    sc.source = &value[1];
    sc.lengths = &uwcf->upstream.store_lengths;
    sc.values = &uwcf->upstream.store_values;
    sc.variables = ngx_http_script_variables_count(&value[1]);
    sc.complete_lengths = 1;
    sc.complete_values = 1;

    if (ngx_http_script_compile(&sc) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


#if (NGX_HTTP_CACHE)

static char *
ngx_http_uwsgi_cache(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_uwsgi_loc_conf_t *uwcf = conf;

    ngx_str_t                         *value;
    ngx_http_complex_value_t           cv;
    ngx_http_compile_complex_value_t   ccv;

    value = cf->args->elts;

    if (uwcf->upstream.cache != NGX_CONF_UNSET) {
        return "is duplicate";
    }

    if (ngx_strcmp(value[1].data, "off") == 0) {
        uwcf->upstream.cache = 0;
        return NGX_CONF_OK;
    }

    if (uwcf->upstream.store > 0) {
        return "is incompatible with \"uwsgi_store\"";
    }

    uwcf->upstream.cache = 1;

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &cv;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    if (cv.lengths != NULL) {

        uwcf->upstream.cache_value = ngx_palloc(cf->pool,
                                                sizeof(ngx_http_complex_value_t));
        if (uwcf->upstream.cache_value == NULL) {
```

```
            return NGX_CONF_ERROR;
        }

        *uwcf->upstream.cache_value = cv;

        return NGX_CONF_OK;
    }

    uwcf->upstream.cache_zone = ngx_shared_memory_add(cf, &value[1], 0,
                                                      &ngx_http_uwsgi_module);
    if (uwcf->upstream.cache_zone == NULL) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static char *
ngx_http_uwsgi_cache_key(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_uwsgi_loc_conf_t *uwcf = conf;

    ngx_str_t                         *value;
    ngx_http_compile_complex_value_t   ccv;

    value = cf->args->elts;

    if (uwcf->cache_key.value.data) {
        return "is duplicate";
    }

    ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));

    ccv.cf = cf;
    ccv.value = &value[1];
    ccv.complex_value = &uwcf->cache_key;

    if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}

#endif


#if (NGX_HTTP_SSL)

static char *
ngx_http_uwsgi_ssl_password_file(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_uwsgi_loc_conf_t *uwcf = conf;

    ngx_str_t  *value;

    if (uwcf->ssl_passwords != NGX_CONF_UNSET_PTR) {
        return "is duplicate";
    }

    value = cf->args->elts;

    uwcf->ssl_passwords = ngx_ssl_read_password_file(cf, &value[1]);

    if (uwcf->ssl_passwords == NULL) {
        return NGX_CONF_ERROR;
    }

    return NGX_CONF_OK;
}


static ngx_int_t
ngx_http_uwsgi_set_ssl(ngx_conf_t *cf, ngx_http_uwsgi_loc_conf_t *uwcf)
{
```

```
2266     ngx_pool_cleanup_t  *cln;
2267
2268     uwcf->upstream.ssl = ngx_pcalloc(cf->pool, sizeof(ngx_ssl_t));
2269     if (uwcf->upstream.ssl == NULL) {
2270         return NGX_ERROR;
2271     }
2272
2273     uwcf->upstream.ssl->log = cf->log;
2274
2275     if (ngx_ssl_create(uwcf->upstream.ssl, uwcf->ssl_protocols, NULL)
2276         != NGX_OK)
2277     {
2278         return NGX_ERROR;
2279     }
2280
2281     cln = ngx_pool_cleanup_add(cf->pool, 0);
2282     if (cln == NULL) {
2283         return NGX_ERROR;
2284     }
2285
2286     cln->handler = ngx_ssl_cleanup_ctx;
2287     cln->data = uwcf->upstream.ssl;
2288
2289     if (uwcf->ssl_certificate.len) {
2290
2291         if (uwcf->ssl_certificate_key.len == 0) {
2292             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
2293                           "no \"uwsgi_ssl_certificate_key\" is defined "
2294                           "for certificate \"%V\"", &uwcf->ssl_certificate);
2295             return NGX_ERROR;
2296         }
2297
2298         if (ngx_ssl_certificate(cf, uwcf->upstream.ssl, &uwcf->ssl_certificate,
2299                                 &uwcf->ssl_certificate_key, uwcf->ssl_passwords)
2300             != NGX_OK)
2301         {
2302             return NGX_ERROR;
2303         }
2304     }
2305
2306     if (SSL_CTX_set_cipher_list(uwcf->upstream.ssl->ctx,
2307                                 (const char *) uwcf->ssl_ciphers.data)
2308         == 0)
2309     {
2310         ngx_ssl_error(NGX_LOG_EMERG, cf->log, 0,
2311                       "SSL_CTX_set_cipher_list(\"%V\") failed",
2312                       &uwcf->ssl_ciphers);
2313         return NGX_ERROR;
2314     }
2315
2316     if (uwcf->upstream.ssl_verify) {
2317         if (uwcf->ssl_trusted_certificate.len == 0) {
2318             ngx_log_error(NGX_LOG_EMERG, cf->log, 0,
2319                       "no uwsgi_ssl_trusted_certificate for uwsgi_ssl_verify");
2320             return NGX_ERROR;
2321         }
2322
2323         if (ngx_ssl_trusted_certificate(cf, uwcf->upstream.ssl,
2324                                         &uwcf->ssl_trusted_certificate,
2325                                         uwcf->ssl_verify_depth)
2326             != NGX_OK)
2327         {
2328             return NGX_ERROR;
2329         }
2330
2331         if (ngx_ssl_crl(cf, uwcf->upstream.ssl, &uwcf->ssl_crl) != NGX_OK) {
2332             return NGX_ERROR;
2333         }
2334     }
2335
2336     return NGX_OK;
2337 }
2338
2339 #endif
```

One Level Up Top Level

src/http/modules/ngx_http_xslt_filter_module.c - nginx-1.7.10

Global variables defined

- ngx_http_next_body_filter
- ngx_http_next_header_filter
- ngx_http_xslt_default_types
- ngx_http_xslt_filter_commands
- ngx_http_xslt_filter_module
- ngx_http_xslt_filter_module_ctx

Data types defined

- ngx_http_xslt_file_t
- ngx_http_xslt_filter_ctx_t
- ngx_http_xslt_filter_loc_conf_t
- ngx_http_xslt_filter_main_conf_t
- ngx_http_xslt_param_t
- ngx_http_xslt_sheet_t

Functions defined

- ngx_http_xslt_add_chunk
- ngx_http_xslt_apply_stylesheet
- ngx_http_xslt_body_filter
- ngx_http_xslt_cleanup
- ngx_http_xslt_cleanup_dtd
- ngx_http_xslt_cleanup_stylesheet
- ngx_http_xslt_content_type
- ngx_http_xslt_encoding
- ngx_http_xslt_entities
- ngx_http_xslt_filter_create_conf
- ngx_http_xslt_filter_create_main_conf
- ngx_http_xslt_filter_exit
- ngx_http_xslt_filter_init
- ngx_http_xslt_filter_merge_conf
- ngx_http_xslt_filter_preconfiguration

- ngx_http_xslt_header_filter
- ngx_http_xslt_param
- ngx_http_xslt_params
- ngx_http_xslt_sax_error
- ngx_http_xslt_sax_external_subset
- ngx_http_xslt_send
- ngx_http_xslt_stylesheet

Macros defined

- NGX_HTTP_XSLT_REUSE_DTD

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10  #include <ngx_http.h>
11
12  #include <libxml/parser.h>
13  #include <libxml/tree.h>
14  #include <libxslt/xslt.h>
15  #include <libxslt/xsltInternals.h>
16  #include <libxslt/transform.h>
17  #include <libxslt/variables.h>
18  #include <libxslt/xsltutils.h>
19
20  #if (NGX_HAVE_EXSLT)
21  #include <libexslt/exslt.h>
22  #endif
23
24
25  #ifndef NGX_HTTP_XSLT_REUSE_DTD
26  #define NGX_HTTP_XSLT_REUSE_DTD  1
27  #endif
28
29
30  typedef struct {
31      u_char                   *name;
32      void                     *data;
33  } ngx_http_xslt_file_t;
34
35
36  typedef struct {
37      ngx_array_t               dtd_files;    /* ngx_http_xslt_file_t */
38      ngx_array_t               sheet_files;  /* ngx_http_xslt_file_t */
39  } ngx_http_xslt_filter_main_conf_t;
40
41
42  typedef struct {
43      u_char                   *name;
44      ngx_http_complex_value_t  value;
45      ngx_uint_t                quote;        /* unsigned  quote:1; */
46  } ngx_http_xslt_param_t;
47
48
49  typedef struct {
50      xsltStylesheetPtr         stylesheet;
51      ngx_array_t               params;       /* ngx_http_xslt_param_t */
```

```
} ngx_http_xslt_sheet_t;


typedef struct {
    xmlDtdPtr                  dtd;
    ngx_array_t                sheets;       /* ngx_http_xslt_sheet_t */
    ngx_hash_t                 types;
    ngx_array_t               *types_keys;
    ngx_array_t               *params;       /* ngx_http_xslt_param_t */
    ngx_flag_t                 last_modified;
} ngx_http_xslt_filter_loc_conf_t;


typedef struct {
    xmlDocPtr                  doc;
    xmlParserCtxtPtr           ctxt;
    xsltTransformContextPtr    transform;
    ngx_http_request_t        *request;
    ngx_array_t                params;

    ngx_uint_t                 done;         /* unsigned  done:1; */
} ngx_http_xslt_filter_ctx_t;


static ngx_int_t ngx_http_xslt_send(ngx_http_request_t *r,
    ngx_http_xslt_filter_ctx_t *ctx, ngx_buf_t *b);
static ngx_int_t ngx_http_xslt_add_chunk(ngx_http_request_t *r,
    ngx_http_xslt_filter_ctx_t *ctx, ngx_buf_t *b);


static void ngx_http_xslt_sax_external_subset(void *data, const xmlChar *name,
    const xmlChar *externalId, const xmlChar *systemId);
static void ngx_cdecl ngx_http_xslt_sax_error(void *data, const char *msg, ...);


static ngx_buf_t *ngx_http_xslt_apply_stylesheet(ngx_http_request_t *r,
    ngx_http_xslt_filter_ctx_t *ctx);
static ngx_int_t ngx_http_xslt_params(ngx_http_request_t *r,
    ngx_http_xslt_filter_ctx_t *ctx, ngx_array_t *params, ngx_uint_t final);
static u_char *ngx_http_xslt_content_type(xsltStylesheetPtr s);
static u_char *ngx_http_xslt_encoding(xsltStylesheetPtr s);
static void ngx_http_xslt_cleanup(void *data);

static char *ngx_http_xslt_entities(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_xslt_stylesheet(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static char *ngx_http_xslt_param(ngx_conf_t *cf, ngx_command_t *cmd,
    void *conf);
static void ngx_http_xslt_cleanup_dtd(void *data);
static void ngx_http_xslt_cleanup_stylesheet(void *data);
static void *ngx_http_xslt_filter_create_main_conf(ngx_conf_t *cf);
static void *ngx_http_xslt_filter_create_conf(ngx_conf_t *cf);
static char *ngx_http_xslt_filter_merge_conf(ngx_conf_t *cf, void *parent,
    void *child);
static ngx_int_t ngx_http_xslt_filter_preconfiguration(ngx_conf_t *cf);
static ngx_int_t ngx_http_xslt_filter_init(ngx_conf_t *cf);
static void ngx_http_xslt_filter_exit(ngx_cycle_t *cycle);


ngx_str_t  ngx_http_xslt_default_types[] = {
    ngx_string("text/xml"),
    ngx_null_string
};


static ngx_command_t  ngx_http_xslt_filter_commands[] = {

    { ngx_string("xml_entities"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_xslt_entities,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("xslt_stylesheet"),
```

```
      NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_xslt_stylesheet,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("xslt_param"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_http_xslt_param,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("xslt_string_param"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE2,
      ngx_http_xslt_param,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      (void *) 1 },

    { ngx_string("xslt_types"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_types_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_xslt_filter_loc_conf_t, types_keys),
      &ngx_http_xslt_default_types[0] },

    { ngx_string("xslt_last_modified"),
      NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_xslt_filter_loc_conf_t, last_modified),
      NULL },

      ngx_null_command
};


static ngx_http_module_t  ngx_http_xslt_filter_module_ctx = {
    ngx_http_xslt_filter_preconfiguration, /* preconfiguration */
    ngx_http_xslt_filter_init,             /* postconfiguration */

    ngx_http_xslt_filter_create_main_conf, /* create main configuration */
    NULL,                                  /* init main configuration */

    NULL,                                  /* create server configuration */
    NULL,                                  /* merge server configuration */

    ngx_http_xslt_filter_create_conf,      /* create location configuration */
    ngx_http_xslt_filter_merge_conf        /* merge location configuration */
};


ngx_module_t  ngx_http_xslt_filter_module = {
    NGX_MODULE_V1,
    &ngx_http_xslt_filter_module_ctx,      /* module context */
    ngx_http_xslt_filter_commands,         /* module directives */
    NGX_HTTP_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    NULL,                                  /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    ngx_http_xslt_filter_exit,             /* exit process */
    ngx_http_xslt_filter_exit,             /* exit master */
    NGX_MODULE_V1_PADDING
};


static ngx_http_output_header_filter_pt  ngx_http_next_header_filter;
static ngx_http_output_body_filter_pt    ngx_http_next_body_filter;


static ngx_int_t
ngx_http_xslt_header_filter(ngx_http_request_t *r)
{
```

```
    ngx_http_xslt_filter_ctx_t       *ctx;
    ngx_http_xslt_filter_loc_conf_t  *conf;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "xslt filter header");

    if (r->headers_out.status == NGX_HTTP_NOT_MODIFIED) {
        return ngx_http_next_header_filter(r);
    }

    conf = ngx_http_get_module_loc_conf(r, ngx_http_xslt_filter_module);

    if (conf->sheets.nelts == 0
        || ngx_http_test_content_type(r, &conf->types) == NULL)
    {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_http_get_module_ctx(r, ngx_http_xslt_filter_module);

    if (ctx) {
        return ngx_http_next_header_filter(r);
    }

    ctx = ngx_pcalloc(r->pool, sizeof(ngx_http_xslt_filter_ctx_t));
    if (ctx == NULL) {
        return NGX_ERROR;
    }

    ngx_http_set_ctx(r, ctx, ngx_http_xslt_filter_module);

    r->main_filter_need_in_memory = 1;

    return NGX_OK;
}


static ngx_int_t
ngx_http_xslt_body_filter(ngx_http_request_t *r, ngx_chain_t *in)
{
    int                          wellFormed;
    ngx_chain_t                 *cl;
    ngx_http_xslt_filter_ctx_t  *ctx;

    ngx_log_debug0(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "xslt filter body");

    if (in == NULL) {
        return ngx_http_next_body_filter(r, in);
    }

    ctx = ngx_http_get_module_ctx(r, ngx_http_xslt_filter_module);

    if (ctx == NULL || ctx->done) {
        return ngx_http_next_body_filter(r, in);
    }

    for (cl = in; cl; cl = cl->next) {

        if (ngx_http_xslt_add_chunk(r, ctx, cl->buf) != NGX_OK) {

            if (ctx->ctxt->myDoc) {

#if (NGX_HTTP_XSLT_REUSE_DTD)
                ctx->ctxt->myDoc->extSubset = NULL;
#endif
                xmlFreeDoc(ctx->ctxt->myDoc);
            }

            xmlFreeParserCtxt(ctx->ctxt);

            return ngx_http_xslt_send(r, ctx, NULL);
        }

        if (cl->buf->last_buf || cl->buf->last_in_chain) {

```

```
            ctx->doc = ctx->ctxt->myDoc;

#if (NGX_HTTP_XSLT_REUSE_DTD)
            ctx->doc->extSubset = NULL;
#endif

            wellFormed = ctx->ctxt->wellFormed;

            xmlFreeParserCtxt(ctx->ctxt);

            if (wellFormed) {
                return ngx_http_xslt_send(r, ctx,
                                          ngx_http_xslt_apply_stylesheet(r, ctx));
            }

            xmlFreeDoc(ctx->doc);

            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "not well formed XML document");

            return ngx_http_xslt_send(r, ctx, NULL);
        }
    }

    return NGX_OK;
}


static ngx_int_t
ngx_http_xslt_send(ngx_http_request_t *r, ngx_http_xslt_filter_ctx_t *ctx,
    ngx_buf_t *b)
{
    ngx_int_t                       rc;
    ngx_chain_t                     out;
    ngx_pool_cleanup_t             *cln;
    ngx_http_xslt_filter_loc_conf_t  *conf;

    ctx->done = 1;

    if (b == NULL) {
        return ngx_http_filter_finalize_request(r, &ngx_http_xslt_filter_module,
                                                NGX_HTTP_INTERNAL_SERVER_ERROR);
    }

    cln = ngx_pool_cleanup_add(r->pool, 0);

    if (cln == NULL) {
        ngx_free(b->pos);
        return ngx_http_filter_finalize_request(r, &ngx_http_xslt_filter_module,
                                                NGX_HTTP_INTERNAL_SERVER_ERROR);
    }

    if (r == r->main) {
        r->headers_out.content_length_n = b->last - b->pos;

        if (r->headers_out.content_length) {
            r->headers_out.content_length->hash = 0;
            r->headers_out.content_length = NULL;
        }

        conf = ngx_http_get_module_loc_conf(r, ngx_http_xslt_filter_module);

        if (!conf->last_modified) {
            ngx_http_clear_last_modified(r);
            ngx_http_clear_etag(r);

        } else {
            ngx_http_weak_etag(r);
        }
    }

    rc = ngx_http_next_header_filter(r);

    if (rc == NGX_ERROR || rc > NGX_OK || r->header_only) {
        ngx_free(b->pos);
        return rc;
```

```
    }

    cln->handler = ngx_http_xslt_cleanup;
    cln->data = b->pos;

    out.buf = b;
    out.next = NULL;

    return ngx_http_next_body_filter(r, &out);
}


static ngx_int_t
ngx_http_xslt_add_chunk(ngx_http_request_t *r, ngx_http_xslt_filter_ctx_t *ctx,
    ngx_buf_t *b)
{
    int               err;
    xmlParserCtxtPtr  ctxt;

    if (ctx->ctxt == NULL) {

        ctxt = xmlCreatePushParserCtxt(NULL, NULL, NULL, 0, NULL);
        if (ctxt == NULL) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "xmlCreatePushParserCtxt() failed");
            return NGX_ERROR;
        }
        xmlCtxtUseOptions(ctxt, XML_PARSE_NOENT|XML_PARSE_DTDLOAD
                                               |XML_PARSE_NOWARNING);

        ctxt->sax->externalSubset = ngx_http_xslt_sax_external_subset;
        ctxt->sax->setDocumentLocator = NULL;
        ctxt->sax->error = ngx_http_xslt_sax_error;
        ctxt->sax->fatalError = ngx_http_xslt_sax_error;
        ctxt->sax->_private = ctx;

        ctx->ctxt = ctxt;
        ctx->request = r;
    }

    err = xmlParseChunk(ctx->ctxt, (char *) b->pos, (int) (b->last - b->pos),
                        (b->last_buf) || (b->last_in_chain));

    if (err == 0) {
        b->pos = b->last;
        return NGX_OK;
    }

    ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                  "xmlParseChunk() failed, error:%d", err);

    return NGX_ERROR;
}


static void
ngx_http_xslt_sax_external_subset(void *data, const xmlChar *name,
    const xmlChar *externalId, const xmlChar *systemId)
{
    xmlParserCtxtPtr ctxt = data;

    xmlDocPtr                        doc;
    xmlDtdPtr                        dtd;
    ngx_http_request_t              *r;
    ngx_http_xslt_filter_ctx_t      *ctx;
    ngx_http_xslt_filter_loc_conf_t  *conf;

    ctx = ctxt->sax->_private;
    r = ctx->request;

    conf = ngx_http_get_module_loc_conf(r, ngx_http_xslt_filter_module);

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "xslt filter extSubset: \"%s\" \"%s\" \"%s\"",
                   name ? name : (xmlChar *) "",
                   externalId ? externalId : (xmlChar *) "",
```

```
                  systemId ? systemId : (xmlChar *) "");

    doc = ctxt->myDoc;

#if (NGX_HTTP_XSLT_REUSE_DTD)

    dtd = conf->dtd;

#else

    dtd = xmlCopyDtd(conf->dtd);
    if (dtd == NULL) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "xmlCopyDtd() failed");
        return;
    }

    if (doc->children == NULL) {
        xmlAddChild((xmlNodePtr) doc, (xmlNodePtr) dtd);

    } else {
        xmlAddPrevSibling(doc->children, (xmlNodePtr) dtd);
    }

#endif

    doc->extSubset = dtd;
}


static void ngx_cdecl
ngx_http_xslt_sax_error(void *data, const char *msg, ...)
{
    xmlParserCtxtPtr ctxt = data;

    size_t                      n;
    va_list                     args;
    ngx_http_xslt_filter_ctx_t  *ctx;
    u_char                      buf[NGX_MAX_ERROR_STR];

    ctx = ctxt->sax->_private;

    buf[0] = '\0';

    va_start(args, msg);
    n = (size_t) vsnprintf((char *) buf, NGX_MAX_ERROR_STR, msg, args);
    va_end(args);

    while (--n && (buf[n] == CR || buf[n] == LF)) { /* void */ }

    ngx_log_error(NGX_LOG_ERR, ctx->request->connection->log, 0,
                  "libxml2 error: \"%*s\"", n + 1, buf);
}


static ngx_buf_t *
ngx_http_xslt_apply_stylesheet(ngx_http_request_t *r,
    ngx_http_xslt_filter_ctx_t *ctx)
{
    int                           len, rc, doc_type;
    u_char                       *type, *encoding;
    ngx_buf_t                    *b;
    ngx_uint_t                    i;
    xmlChar                      *buf;
    xmlDocPtr                     doc, res;
    ngx_http_xslt_sheet_t        *sheet;
    ngx_http_xslt_filter_loc_conf_t  *conf;

    conf = ngx_http_get_module_loc_conf(r, ngx_http_xslt_filter_module);
    sheet = conf->sheets.elts;
    doc = ctx->doc;

    /* preallocate array for 4 params */

    if (ngx_array_init(&ctx->params, r->pool, 4 * 2 + 1, sizeof(char *))
        != NGX_OK)
```

```
    {
        xmlFreeDoc(doc);
        return NULL;
    }

    for (i = 0; i < conf->sheets.nelts; i++) {

        ctx->transform = xsltNewTransformContext(sheet[i].stylesheet, doc);
        if (ctx->transform == NULL) {
            xmlFreeDoc(doc);
            return NULL;
        }

        if (conf->params
            && ngx_http_xslt_params(r, ctx, conf->params, 0) != NGX_OK)
        {
            xsltFreeTransformContext(ctx->transform);
            xmlFreeDoc(doc);
            return NULL;
        }

        if (ngx_http_xslt_params(r, ctx, &sheet[i].params, 1) != NGX_OK) {
            xsltFreeTransformContext(ctx->transform);
            xmlFreeDoc(doc);
            return NULL;
        }

        res = xsltApplyStylesheetUser(sheet[i].stylesheet, doc,
                                      ctx->params.elts, NULL, NULL,
                                      ctx->transform);

        xsltFreeTransformContext(ctx->transform);
        xmlFreeDoc(doc);

        if (res == NULL) {
            ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                          "xsltApplyStylesheet() failed");
            return NULL;
        }

        doc = res;

        /* reset array elements */
        ctx->params.nelts = 0;
    }

    /* there must be at least one stylesheet */

    if (r == r->main) {
        type = ngx_http_xslt_content_type(sheet[i - 1].stylesheet);

    } else {
        type = NULL;
    }

    encoding = ngx_http_xslt_encoding(sheet[i - 1].stylesheet);
    doc_type = doc->type;

    ngx_log_debug3(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                   "xslt filter type: %d t:%s e:%s",
                   doc_type, type ? type : (u_char *) "(null)",
                   encoding ? encoding : (u_char *) "(null)");

    rc = xsltSaveResultToString(&buf, &len, doc, sheet[i - 1].stylesheet);

    xmlFreeDoc(doc);

    if (rc != 0) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "xsltSaveResultToString() failed");
        return NULL;
    }

    if (len == 0) {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "xsltSaveResultToString() returned zero-length result");
```

```
584         return NULL;
585     }
586 
587     b = ngx_pcalloc(r->pool, sizeof(ngx_buf_t));
588     if (b == NULL) {
589         ngx_free(buf);
590         return NULL;
591     }
592 
593     b->pos = buf;
594     b->last = buf + len;
595     b->memory = 1;
596 
597     if (encoding) {
598         r->headers_out.charset.len = ngx_strlen(encoding);
599         r->headers_out.charset.data = encoding;
600     }
601 
602     if (r != r->main) {
603         return b;
604     }
605 
606     b->last_buf = 1;
607 
608     if (type) {
609         len = ngx_strlen(type);
610 
611         r->headers_out.content_type_len = len;
612         r->headers_out.content_type.len = len;
613         r->headers_out.content_type.data = type;
614 
615     } else if (doc_type == XML_HTML_DOCUMENT_NODE) {
616 
617         r->headers_out.content_type_len = sizeof("text/html") - 1;
618         ngx_str_set(&r->headers_out.content_type, "text/html");
619     }
620 
621     r->headers_out.content_type_lowcase = NULL;
622 
623     return b;
624 }
625 
626 
627 static ngx_int_t
628 ngx_http_xslt_params(ngx_http_request_t *r, ngx_http_xslt_filter_ctx_t *ctx,
629     ngx_array_t *params, ngx_uint_t final)
630 {
631     u_char                 *p, *last, *value, *dst, *src, **s;
632     size_t                  len;
633     ngx_uint_t              i;
634     ngx_str_t               string;
635     ngx_http_xslt_param_t  *param;
636 
637     param = params->elts;
638 
639     for (i = 0; i < params->nelts; i++) {
640 
641         if (ngx_http_complex_value(r, &param[i].value, &string) != NGX_OK) {
642             return NGX_ERROR;
643         }
644 
645         ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
646                        "xslt filter param: \"%s\"", string.data);
647 
648         if (param[i].name) {
649 
650             ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
651                            "xslt filter param name: \"%s\"", param[i].name);
652 
653             if (param[i].quote) {
654                 if (xsltQuoteOneUserParam(ctx->transform, param[i].name,
655                                           string.data)
656                     != 0)
657                 {
658                     ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
659                                   "xsltQuoteOneUserParam(\"%s\", \"%s\") failed",
```

```
                                param[i].name, string.data);
                    return NGX_ERROR;
                }

                continue;
            }

            s = ngx_array_push(&ctx->params);
            if (s == NULL) {
                return NGX_ERROR;
            }

            *s = param[i].name;

            s = ngx_array_push(&ctx->params);
            if (s == NULL) {
                return NGX_ERROR;
            }

            *s = string.data;

            continue;
        }

        /*
         * parse param1=value1:param2=value2 syntax as used by parameters
         * specified in xslt_stylesheet directives
         */

        p = string.data;
        last = string.data + string.len;

        while (p && *p) {

            value = p;
            p = (u_char *) ngx_strchr(p, '=');
            if (p == NULL) {
                ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                              "invalid libxslt parameter \"%s\"", value);
                return NGX_ERROR;
            }
            *p++ = '\0';

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "xslt filter param name: \"%s\"", value);

            s = ngx_array_push(&ctx->params);
            if (s == NULL) {
                return NGX_ERROR;
            }

            *s = value;

            value = p;
            p = (u_char *) ngx_strchr(p, ':');

            if (p) {
                len = p - value;
                *p++ = '\0';

            } else {
                len = last - value;
            }

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "xslt filter param value: \"%s\"", value);

            dst = value;
            src = value;

            ngx_unescape_uri(&dst, &src, len, 0);

            *dst = '\0';

            ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
                           "xslt filter param unescaped: \"%s\"", value);
```

```
            s = ngx_array_push(&ctx->params);
            if (s == NULL) {
                return NGX_ERROR;
            }

            *s = value;
        }
    }

    if (final) {
        s = ngx_array_push(&ctx->params);
        if (s == NULL) {
            return NGX_ERROR;
        }

        *s = NULL;
    }

    return NGX_OK;
}


static u_char *
ngx_http_xslt_content_type(xsltStylesheetPtr s)
{
    u_char  *type;

    if (s->mediaType) {
        return s->mediaType;
    }

    for (s = s->imports; s; s = s->next) {

        type = ngx_http_xslt_content_type(s);

        if (type) {
            return type;
        }
    }

    return NULL;
}


static u_char *
ngx_http_xslt_encoding(xsltStylesheetPtr s)
{
    u_char  *encoding;

    if (s->encoding) {
        return s->encoding;
    }

    for (s = s->imports; s; s = s->next) {

        encoding = ngx_http_xslt_encoding(s);

        if (encoding) {
            return encoding;
        }
    }

    return NULL;
}


static void
ngx_http_xslt_cleanup(void *data)
{
    ngx_free(data);
}


static char *
ngx_http_xslt_entities(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
```

```
{
    ngx_http_xslt_filter_loc_conf_t *xlcf = conf;

    ngx_str_t                       *value;
    ngx_uint_t                       i;
    ngx_pool_cleanup_t              *cln;
    ngx_http_xslt_file_t            *file;
    ngx_http_xslt_filter_main_conf_t  *xmcf;

    if (xlcf->dtd) {
        return "is duplicate";
    }

    value = cf->args->elts;

    xmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_xslt_filter_module);

    file = xmcf->dtd_files.elts;
    for (i = 0; i < xmcf->dtd_files.nelts; i++) {
        if (ngx_strcmp(file[i].name, value[1].data) == 0) {
            xlcf->dtd = file[i].data;
            return NGX_CONF_OK;
        }
    }

    cln = ngx_pool_cleanup_add(cf->pool, 0);
    if (cln == NULL) {
        return NGX_CONF_ERROR;
    }

    xlcf->dtd = xmlParseDTD(NULL, (xmlChar *) value[1].data);

    if (xlcf->dtd == NULL) {
        ngx_conf_log_error(NGX_LOG_ERR, cf, 0, "xmlParseDTD() failed");
        return NGX_CONF_ERROR;
    }

    cln->handler = ngx_http_xslt_cleanup_dtd;
    cln->data = xlcf->dtd;

    file = ngx_array_push(&xmcf->dtd_files);
    if (file == NULL) {
        return NGX_CONF_ERROR;
    }

    file->name = value[1].data;
    file->data = xlcf->dtd;

    return NGX_CONF_OK;
}



static char *
ngx_http_xslt_stylesheet(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_http_xslt_filter_loc_conf_t *xlcf = conf;

    ngx_str_t                       *value;
    ngx_uint_t                       i, n;
    ngx_pool_cleanup_t              *cln;
    ngx_http_xslt_file_t            *file;
    ngx_http_xslt_sheet_t           *sheet;
    ngx_http_xslt_param_t           *param;
    ngx_http_compile_complex_value_t  ccv;
    ngx_http_xslt_filter_main_conf_t  *xmcf;

    value = cf->args->elts;

    if (xlcf->sheets.elts == NULL) {
        if (ngx_array_init(&xlcf->sheets, cf->pool, 1,
                           sizeof(ngx_http_xslt_sheet_t))
            != NGX_OK)
        {
            return NGX_CONF_ERROR;
        }
```

```
888     }
889
890     sheet = ngx_array_push(&xlcf->sheets);
891     if (sheet == NULL) {
892         return NGX_CONF_ERROR;
893     }
894
895     ngx_memzero(sheet, sizeof(ngx_http_xslt_sheet_t));
896
897     if (ngx_conf_full_name(cf->cycle, &value[1], 0) != NGX_OK) {
898         return NGX_CONF_ERROR;
899     }
900
901     xmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_xslt_filter_module);
902
903     file = xmcf->sheet_files.elts;
904     for (i = 0; i < xmcf->sheet_files.nelts; i++) {
905         if (ngx_strcmp(file[i].name, value[1].data) == 0) {
906             sheet->stylesheet = file[i].data;
907             goto found;
908         }
909     }
910
911     cln = ngx_pool_cleanup_add(cf->pool, 0);
912     if (cln == NULL) {
913         return NGX_CONF_ERROR;
914     }
915
916     sheet->stylesheet = xsltParseStylesheetFile(value[1].data);
917     if (sheet->stylesheet == NULL) {
918         ngx_conf_log_error(NGX_LOG_ERR, cf, 0,
919                            "xsltParseStylesheetFile(\"%s\") failed",
920                            value[1].data);
921         return NGX_CONF_ERROR;
922     }
923
924     cln->handler = ngx_http_xslt_cleanup_stylesheet;
925     cln->data = sheet->stylesheet;
926
927     file = ngx_array_push(&xmcf->sheet_files);
928     if (file == NULL) {
929         return NGX_CONF_ERROR;
930     }
931
932     file->name = value[1].data;
933     file->data = sheet->stylesheet;
934
935 found:
936
937     n = cf->args->nelts;
938
939     if (n == 2) {
940         return NGX_CONF_OK;
941     }
942
943     if (ngx_array_init(&sheet->params, cf->pool, n - 2,
944                        sizeof(ngx_http_xslt_param_t))
945         != NGX_OK)
946     {
947         return NGX_CONF_ERROR;
948     }
949
950     for (i = 2; i < n; i++) {
951
952         param = ngx_array_push(&sheet->params);
953         if (param == NULL) {
954             return NGX_CONF_ERROR;
955         }
956
957         ngx_memzero(param, sizeof(ngx_http_xslt_param_t));
958         ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
959
960         ccv.cf = cf;
961         ccv.value = &value[i];
962         ccv.complex_value = &param->value;
963         ccv.zero = 1;
```

```
964
965         if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
966             return NGX_CONF_ERROR;
967         }
968     }
969
970     return NGX_CONF_OK;
971 }
972
973
974 static char *
975 ngx_http_xslt_param(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
976 {
977     ngx_http_xslt_filter_loc_conf_t  *xlcf = conf;
978
979     ngx_http_xslt_param_t            *param;
980     ngx_http_compile_complex_value_t  ccv;
981     ngx_str_t                        *value;
982
983     value = cf->args->elts;
984
985     if (xlcf->params == NULL) {
986         xlcf->params = ngx_array_create(cf->pool, 2,
987                                         sizeof(ngx_http_xslt_param_t));
988         if (xlcf->params == NULL) {
989             return NGX_CONF_ERROR;
990         }
991     }
992
993     param = ngx_array_push(xlcf->params);
994     if (param == NULL) {
995         return NGX_CONF_ERROR;
996     }
997
998     param->name = value[1].data;
999     param->quote = (cmd->post == NULL) ? 0 : 1;
1000
1001     ngx_memzero(&ccv, sizeof(ngx_http_compile_complex_value_t));
1002
1003     ccv.cf = cf;
1004     ccv.value = &value[2];
1005     ccv.complex_value = &param->value;
1006     ccv.zero = 1;
1007
1008     if (ngx_http_compile_complex_value(&ccv) != NGX_OK) {
1009         return NGX_CONF_ERROR;
1010     }
1011
1012     return NGX_CONF_OK;
1013 }
1014
1015
1016 static void
1017 ngx_http_xslt_cleanup_dtd(void *data)
1018 {
1019     xmlFreeDtd(data);
1020 }
1021
1022
1023 static void
1024 ngx_http_xslt_cleanup_stylesheet(void *data)
1025 {
1026     xsltFreeStylesheet(data);
1027 }
1028
1029
1030 static void *
1031 ngx_http_xslt_filter_create_main_conf(ngx_conf_t *cf)
1032 {
1033     ngx_http_xslt_filter_main_conf_t  *conf;
1034
1035     conf = ngx_palloc(cf->pool, sizeof(ngx_http_xslt_filter_main_conf_t));
1036     if (conf == NULL) {
1037         return NULL;
1038     }
1039
```

```
    if (ngx_array_init(&conf->dtd_files, cf->pool, 1,
                       sizeof(ngx_http_xslt_file_t))
        != NGX_OK)
    {
        return NULL;
    }

    if (ngx_array_init(&conf->sheet_files, cf->pool, 1,
                       sizeof(ngx_http_xslt_file_t))
        != NGX_OK)
    {
        return NULL;
    }

    return conf;
}


static void *
ngx_http_xslt_filter_create_conf(ngx_conf_t *cf)
{
    ngx_http_xslt_filter_loc_conf_t  *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_xslt_filter_loc_conf_t));
    if (conf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *     conf->dtd = NULL;
     *     conf->sheets = { NULL };
     *     conf->types = { NULL };
     *     conf->types_keys = NULL;
     *     conf->params = NULL;
     */

    conf->last_modified = NGX_CONF_UNSET;

    return conf;
}


static char *
ngx_http_xslt_filter_merge_conf(ngx_conf_t *cf, void *parent, void *child)
{
    ngx_http_xslt_filter_loc_conf_t *prev = parent;
    ngx_http_xslt_filter_loc_conf_t *conf = child;

    if (conf->dtd == NULL) {
        conf->dtd = prev->dtd;
    }

    if (conf->sheets.nelts == 0) {
        conf->sheets = prev->sheets;
    }

    if (conf->params == NULL) {
        conf->params = prev->params;
    }

    if (ngx_http_merge_types(cf, &conf->types_keys, &conf->types,
                             &prev->types_keys, &prev->types,
                             ngx_http_xslt_default_types)
        != NGX_OK)
    {
        return NGX_CONF_ERROR;
    }

    ngx_conf_merge_value(conf->last_modified, prev->last_modified, 0);

    return NGX_CONF_OK;
}


```

```
1116 static ngx_int_t
1117 ngx_http_xslt_filter_preconfiguration(ngx_conf_t *cf)
1118 {
1119     xmlInitParser();
1120
1121 #if (NGX_HAVE_EXSLT)
1122     exsltRegisterAll();
1123 #endif
1124
1125     return NGX_OK;
1126 }
1127
1128
1129 static ngx_int_t
1130 ngx_http_xslt_filter_init(ngx_conf_t *cf)
1131 {
1132     ngx_http_next_header_filter = ngx_http_top_header_filter;
1133     ngx_http_top_header_filter = ngx_http_xslt_header_filter;
1134
1135     ngx_http_next_body_filter = ngx_http_top_body_filter;
1136     ngx_http_top_body_filter = ngx_http_xslt_body_filter;
1137
1138     return NGX_OK;
1139 }
1140
1141
1142 static void
1143 ngx_http_xslt_filter_exit(ngx_cycle_t *cycle)
1144 {
1145     xsltCleanupGlobals();
1146     xmlCleanupParser();
1147 }
```

One Level Up Top Level

src/http/modules/perl/nginx.pm - nginx-1.7.10

Data types defined

- rflush

Source code

```
1  package nginx;
2
3  use 5.006001;
4  use strict;
5  use warnings;
6
7  require Exporter;
8
9  our @ISA = qw(Exporter);
10
11 our @EXPORT = qw(
12     OK
13     DECLINED
14
15     HTTP_OK
16     HTTP_CREATED
17     HTTP_ACCEPTED
18     HTTP_NO_CONTENT
19     HTTP_PARTIAL_CONTENT
20
21     HTTP_MOVED_PERMANENTLY
22     HTTP_MOVED_TEMPORARILY
23     HTTP_REDIRECT
24     HTTP_SEE_OTHER
25     HTTP_NOT_MODIFIED
26     HTTP_TEMPORARY_REDIRECT
27
28     HTTP_BAD_REQUEST
29     HTTP_UNAUTHORIZED
30     HTTP_PAYMENT_REQUIRED
31     HTTP_FORBIDDEN
32     HTTP_NOT_FOUND
33     HTTP_NOT_ALLOWED
34     HTTP_NOT_ACCEPTABLE
35     HTTP_REQUEST_TIME_OUT
36     HTTP_CONFLICT
37     HTTP_GONE
38     HTTP_LENGTH_REQUIRED
39     HTTP_REQUEST_ENTITY_TOO_LARGE
40     HTTP_REQUEST_URI_TOO_LARGE
41     HTTP_UNSUPPORTED_MEDIA_TYPE
42     HTTP_RANGE_NOT_SATISFIABLE
43
44     HTTP_INTERNAL_SERVER_ERROR
45     HTTP_SERVER_ERROR
46     HTTP_NOT_IMPLEMENTED
47     HTTP_BAD_GATEWAY
48     HTTP_SERVICE_UNAVAILABLE
49     HTTP_GATEWAY_TIME_OUT
50     HTTP_INSUFFICIENT_STORAGE
51 );
52
53 our $VERSION = '%%VERSION%%';
54
55 require XSLoader;
56 XSLoader::load('nginx', $VERSION);
57
58 # Preloaded methods go here.
59
60 use constant OK                             => 0;
61 use constant DECLINED                       => -5;
62
```

```
use constant HTTP_OK                        => 200;
use constant HTTP_CREATED                   => 201;
use constant HTTP_ACCEPTED                  => 202;
use constant HTTP_NO_CONTENT                => 204;
use constant HTTP_PARTIAL_CONTENT           => 206;

use constant HTTP_MOVED_PERMANENTLY         => 301;
use constant HTTP_MOVED_TEMPORARILY         => 302;
use constant HTTP_REDIRECT                  => 302;
use constant HTTP_SEE_OTHER                 => 303;
use constant HTTP_NOT_MODIFIED              => 304;
use constant HTTP_TEMPORARY_REDIRECT        => 307;

use constant HTTP_BAD_REQUEST               => 400;
use constant HTTP_UNAUTHORIZED              => 401;
use constant HTTP_PAYMENT_REQUIRED          => 402;
use constant HTTP_FORBIDDEN                 => 403;
use constant HTTP_NOT_FOUND                 => 404;
use constant HTTP_NOT_ALLOWED               => 405;
use constant HTTP_NOT_ACCEPTABLE            => 406;
use constant HTTP_REQUEST_TIME_OUT          => 408;
use constant HTTP_CONFLICT                  => 409;
use constant HTTP_GONE                      => 410;
use constant HTTP_LENGTH_REQUIRED           => 411;
use constant HTTP_REQUEST_ENTITY_TOO_LARGE  => 413;
use constant HTTP_REQUEST_URI_TOO_LARGE     => 414;
use constant HTTP_UNSUPPORTED_MEDIA_TYPE    => 415;
use constant HTTP_RANGE_NOT_SATISFIABLE     => 416;

use constant HTTP_INTERNAL_SERVER_ERROR     => 500;
use constant HTTP_SERVER_ERROR              => 500;
use constant HTTP_NOT_IMPLEMENTED           => 501;
use constant HTTP_BAD_GATEWAY               => 502;
use constant HTTP_SERVICE_UNAVAILABLE       => 503;
use constant HTTP_GATEWAY_TIME_OUT          => 504;
use constant HTTP_INSUFFICIENT_STORAGE      => 507;


sub rflush {
    my $r = shift;

    $r->flush;
}


1;
__END__

=head1 NAME

nginx - Perl interface to the nginx HTTP server API

=head1 SYNOPSIS

  use nginx;

=head1 DESCRIPTION

This module provides a Perl interface to the nginx HTTP server API.


=head1 SEE ALSO

http://nginx.org/en/docs/http/ngx_http_perl_module.html

=head1 AUTHOR

Igor Sysoev

=head1 COPYRIGHT AND LICENSE

Copyright (C) Igor Sysoev
Copyright (C) Nginx, Inc.


=cut
```

One Level Up Top Level

src/core/ngx_crypt.h - nginx-1.7.10

Macros defined

- _NGX_CRYPT_H_INCLUDED_

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #ifndef _NGX_CRYPT_H_INCLUDED_
9  #define _NGX_CRYPT_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  ngx_int_t ngx_crypt(ngx_pool_t *pool, u_char *key, u_char *salt,
17      u_char **encrypted);
18
19
20  #endif /* _NGX_CRYPT_H_INCLUDED_ */
```

One Level Up Top Level

One Level Up Top Level

src/core/ngx_murmurhash.h - nginx-1.7.10

Macros defined

- _NGX_MURMURHASH_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_MURMURHASH_H_INCLUDED_
9  #define _NGX_MURMURHASH_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  uint32_t ngx_murmur_hash2(u_char *data, size_t len);
17
18
19  #endif /* _NGX_MURMURHASH_H_INCLUDED_ */
```

One Level Up Top Level

src/core/ngx_parse.h - nginx-1.7.10

Macros defined

- _NGX_PARSE_H_INCLUDED_

Source code

```
1
2  /*
3   * Copyright (C) Igor Sysoev
4   * Copyright (C) Nginx, Inc.
5   */
6
7
8  #ifndef _NGX_PARSE_H_INCLUDED_
9  #define _NGX_PARSE_H_INCLUDED_
10
11
12  #include <ngx_config.h>
13  #include <ngx_core.h>
14
15
16  ssize_t ngx_parse_size(ngx_str_t *line);
17  off_t ngx_parse_offset(ngx_str_t *line);
18  ngx_int_t ngx_parse_time(ngx_str_t *line, ngx_uint_t is_sec);
19
20
21  #endif /* _NGX_PARSE_H_INCLUDED_ */
```

src/misc/ - nginx-1.7.10

- ngx_cpp_test_module.cpp
- ngx_google_perftools_module.c

src/misc/ngx_cpp_test_module.cpp - nginx-1.7.10

Functions defined

- ngx_cpp_test_handler

Source code

```
1
2  // stub module to test header files' C++ compatibility
3
4  extern "C" {
5    #include <ngx_config.h>
6    #include <ngx_core.h>
7    #include <ngx_event.h>
8    #include <ngx_event_connect.h>
9    #include <ngx_event_pipe.h>
10
11   #include <ngx_http.h>
12
13   #include <ngx_mail.h>
14   #include <ngx_mail_pop3_module.h>
15   #include <ngx_mail_imap_module.h>
16   #include <ngx_mail_smtp_module.h>
17 }
18
19 // nginx header files should go before other, because they define 64-bit off_t
20 // #include <string>
21
22
23 void ngx_cpp_test_handler(void *data);
24
25 void
26 ngx_cpp_test_handler(void *data)
27 {
28     return;
29 }
```

src/misc/ngx_google_perftools_module.c - nginx-1.7.10

Global variables defined

- ngx_google_perftools_commands
- ngx_google_perftools_module
- ngx_google_perftools_module_ctx

Data types defined

- ngx_google_perftools_conf_t

Functions defined

- ngx_google_perftools_create_conf
- ngx_google_perftools_worker

Source code

```
1
2  /*
3  * Copyright (C) Igor Sysoev
4  * Copyright (C) Nginx, Inc.
5  */
6
7
8  #include <ngx_config.h>
9  #include <ngx_core.h>
10
11  /*
12  * declare Profiler interface here because
13  * <google/profiler.h> is C++ header file
14  */
15
16  int ProfilerStart(u_char* fname);
17  void ProfilerStop(void);
18  void ProfilerRegisterThread(void);
19
20
21  static void *ngx_google_perftools_create_conf(ngx_cycle_t *cycle);
22  static ngx_int_t ngx_google_perftools_worker(ngx_cycle_t *cycle);
23
24
25  typedef struct {
26      ngx_str_t  profiles;
27  } ngx_google_perftools_conf_t;
28
29
30  static ngx_command_t  ngx_google_perftools_commands[] = {
31
32      { ngx_string("google_perftools_profiles"),
33        NGX_MAIN_CONF|NGX_DIRECT_CONF|NGX_CONF_TAKE1,
34        ngx_conf_set_str_slot,
35        0,
36        offsetof(ngx_google_perftools_conf_t, profiles),
37        NULL },
38
39      ngx_null_command
40  };
41
42
43  static ngx_core_module_t  ngx_google_perftools_module_ctx = {
44      ngx_string("google_perftools"),
```

```
    ngx_google_perftools_create_conf,
    NULL
};


ngx_module_t  ngx_google_perftools_module = {
    NGX_MODULE_V1,
    &ngx_google_perftools_module_ctx,      /* module context */
    ngx_google_perftools_commands,         /* module directives */
    NGX_CORE_MODULE,                       /* module type */
    NULL,                                  /* init master */
    NULL,                                  /* init module */
    ngx_google_perftools_worker,           /* init process */
    NULL,                                  /* init thread */
    NULL,                                  /* exit thread */
    NULL,                                  /* exit process */
    NULL,                                  /* exit master */
    NGX_MODULE_V1_PADDING
};


static void *
ngx_google_perftools_create_conf(ngx_cycle_t *cycle)
{
    ngx_google_perftools_conf_t  *gptcf;

    gptcf = ngx_pcalloc(cycle->pool, sizeof(ngx_google_perftools_conf_t));
    if (gptcf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc()
     *
     *     gptcf->profiles = { 0, NULL };
     */

    return gptcf;
}


static ngx_int_t
ngx_google_perftools_worker(ngx_cycle_t *cycle)
{
    u_char                      *profile;
    ngx_google_perftools_conf_t  *gptcf;

    gptcf = (ngx_google_perftools_conf_t *)
                ngx_get_conf(cycle->conf_ctx, ngx_google_perftools_module);

    if (gptcf->profiles.len == 0) {
        return NGX_OK;
    }

    profile = ngx_alloc(gptcf->profiles.len + NGX_INT_T_LEN + 2, cycle->log);
     if (profile == NULL) {
         return NGX_OK;
     }

     if (getenv("CPUPROFILE")) {
         /* disable inherited Profiler enabled in master process */
         ProfilerStop();
     }

     ngx_sprintf(profile, "%V.%d%Z", &gptcf->profiles, ngx_pid);

     if (ProfilerStart(profile)) {
         /* start ITIMER_PROF timer */
         ProfilerRegisterThread();

     } else {
         ngx_log_error(NGX_LOG_CRIT, cycle->log, ngx_errno,
                       "ProfilerStart(%s) failed", profile);
     }

     ngx_free(profile);
```

```
121
122     return NGX_OK;
123 }
124
125
126 /* ProfilerStop() is called on Profiler destruction */
```

One Level Up Top Level