ngx_openresty: an Nginx ecosystem glued by Lua

ngx_openresty:
an Nginx ecosystem glued by Lua

由 Lua 粘合的 Nginx 生态环境

☺*agentzh@gmail.com*☺
章亦春 (agentzh)

2012.02

♡ I've been *hacking* in the Fuzhou city in the last 7 months

过去 7 个月中我一直在福州写码。。。


@agentzh
weibo.com/agentzh

♡ The trend in *AJAX-ization* and *service-ization*
makes everything speak the HTTP protocol

AJAX 化和 Service 化的趋势让所有东西开始讲 HTTP 协议

♡ Nginx is *fast*, because of I/O multiplexing

Nginx 很快，因为 I/O 多路复用

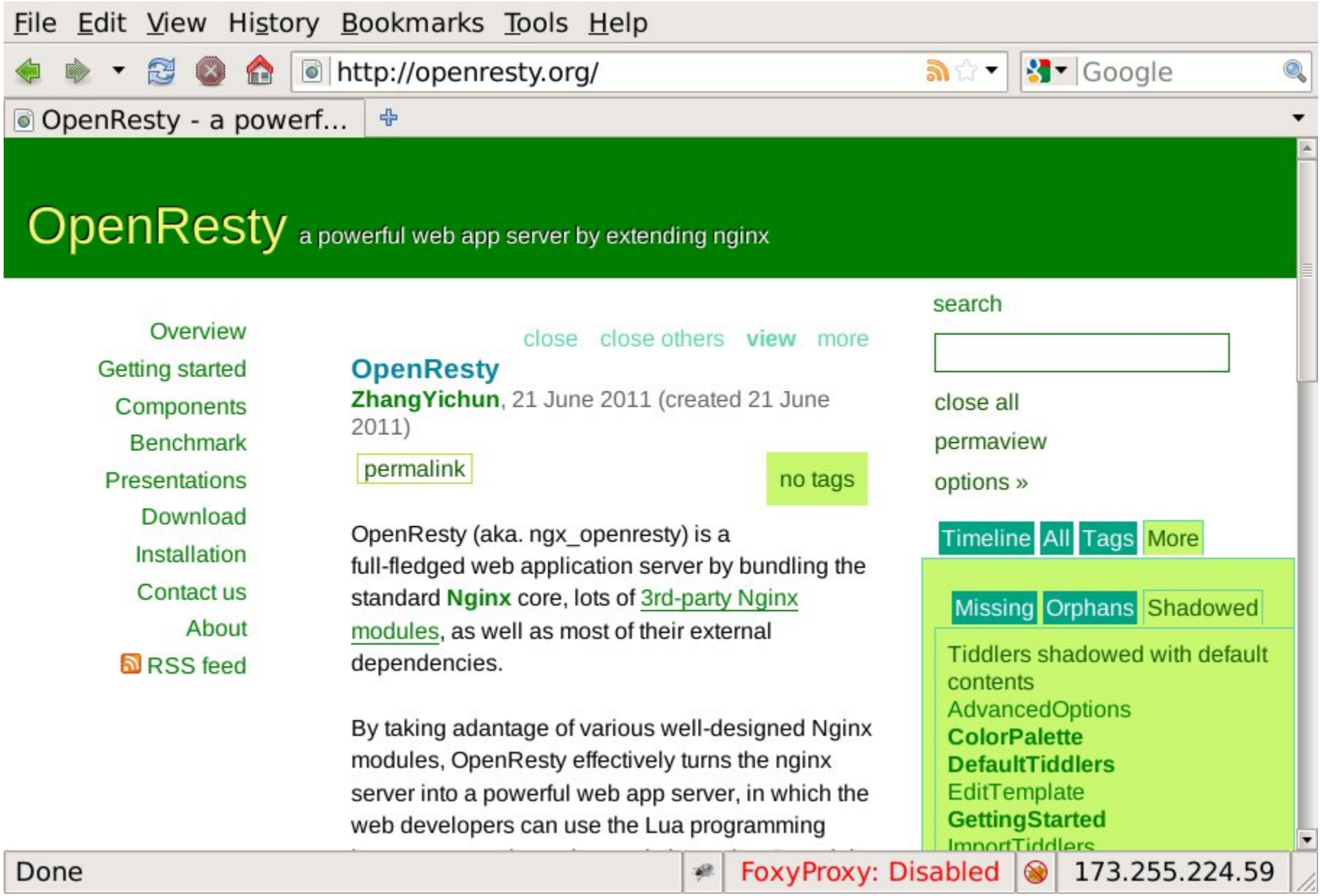

request 1
R/W
request 2
R/W
request 3
R/W
epoll_wait

♡ ngx_openresty is a *bundle* for Nginx,
lots of useful Nginx modules,
and lots of useful Lua libraries.

ngx_openresty 是 Nginx，许多有用的 Nginx 模块，
以及有用的 Lua 库的软件集合。

♡ Our *homepage*: http://openresty.org

我们的主页


File Edit View History Bookmarks Tools Help
http://openresty.org/
Google
OpenResty - a powerf...
OpenResty a powerful web app server by extending nginx
Overview
Getting started
Components
Benchmark
Presentations
Download
Installation
Contact us
About
RSS feed
close close others view more
OpenResty
ZhangYichun, 21 June 2011 (created 21 June 2011)
permalink
no tags
OpenResty (aka. ngx_openresty) is a full-fledged web application server by bundling the standard Nginx core, lots of 3rd-party Nginx modules, as well as most of their external dependencies.
By taking adantage of various well-designed Nginx modules, OpenResty effectively turns the nginx server into a powerful web app server, in which the web developers can use the Lua programming
search
close all
permaview
options »
Timeline All Tags More
Missing Orphans Shadowed
Tiddlers shadowed with default contents
AdvancedOptions
ColorPalette
DefaultTiddlers
EditTemplate
GettingStarted
ImportTiddlers
Done
FoxyProxy: Disabled
173.255.224.59

♡ The Nginx *configure file* notation
is a small language

Nginx 的配置文件记法就是一种小语言

```
location = '/hello' {
    set_unescape_uri $person $arg_person;
    set_if_empty $person 'anonymous';
    echo "hello, $person!";
}
```

```
$ curl 'http://localhost/hello?person=%E7%AB%A0%E4%BA%A6%E6%98%A5'
hello, 章亦春
```

```
$ curl 'http://localhost/hello'
hello, anonymous
```

♡ Various Nginx modules are *enriching* its vicabulary

众多 Nginx 模块正丰富着它的词汇表

♡ **ngx_memc**

➥ **an Nginx upstream module for *Memcached***

针对 **Memcached** 服务器的 **Nginx** 上游模块

```
# (not quite) REST interface to our memcached server
#  at 127.0.0.1:11211
location = /memc {
    set $memc_cmd $arg_cmd;
    set $memc_key $arg_key;
    set $memc_value $arg_val;
    set $memc_exptime $arg_exptime;

    memc_pass 127.0.0.1:11211;
}
```

```
$ curl 'http://localhost/memc?cmd=flush_all';
OK

$ curl 'http://localhost/memc?cmd=replace&key=foo&val=FOO';
NOT_STORED
```

♡ **ngx_drizzle**

➥ **an Nginx upstream module for *MySQL* and *Drizzle***

针对 **MySQL** 和 **Drizzle** 数据库的 **Nginx** 上游模块

```
upstream my_mysql_backend {
    drizzle_server 127.0.0.1:3306 dbname=test
                password=some_pass user=monty
                protocol=mysql;

    # a connection pool that can cache up to
    #   200 mysql TCP connections
    drizzle_keepalive max=200 overflow=reject;
}
```

```
location ~ '^/cat/(.*)' {
    set $name $1;
    set_quote_sql_str $quoted_name $name;
    drizzle_query "select *
        from cats
        where name=$quoted_name";

    drizzle_pass my_mysql_backend;

    rds_json on;
}
```

```
$ curl 'http://localhost/cat/Jerry'
[{"name":"Jerry","age":1}]
```

♡ **ngx_postgres**

➥ **an Nginx upstream module for *PostgreSQL***

针对 **PostgreSQL** 数据库的 **Nginx** 上游模块

```
upstream my_pg_backend {
    postgres_server 10.62.136.3:5432 dbname=test
            user=someone password=123456;

    postgres_keepalive max=50 mode=single overflow=ignore;
}
```

```
location ~ '^/cat/(.*)' {
    set $name $1;
    set_quote_pgsql_str $quoted_name $name;
    postgres_query "select *
        from cats
        where name=$quoted_name";

    postgres_pass my_pg_backend;

    rds_json on;
}
```

```
$ curl 'http://localhost/cat/Jerry'
[{"name":"Jerry","age":1}]
```

♡ **ngx_redis2**

➥ **an Nginx upstream module for *Redis***

针对 **Redis** 服务器的 **Nginx** 上游模块

```
upstream my_redis_node {
    server 127.0.0.1:6379;
    keepalive 1024 single;
}
```

```
# multiple pipelined queries
location /foo {
    set $value 'first';
    redis2_query set one $value;
    redis2_query get one;

    redis2_pass my_redis_node;
}
```

♡ **ngx_srcache**

➥ **General location response *cache***
based on Nginx subrequests

基于 **Nginx** 子请求的通用 **location** 响应缓存

```
location /api {
    set $key "$uri?$args";
    srcache_fetch GET /memc key=$key;
    srcache_store PUT /memc key=$key&exptime=3600;

    # proxy_pass/drizzle_pass/postgres_pass/etc
}
```

```
location /memc {
    internal;

    set_unescape_uri $memc_key $arg_key;
    set $memc_exptime $arg_exptime;

    set_hashed_upstream $backend my_memc_cluster $memc_key;

    memc_pass $backend;
}
```

```
upstream memc1 {
    server 10.32.126.3:11211;
}

upstream memc2 {
    server 10.32.126.4:11211;
}

upstream_list my_memc_cluster memc1 memc2;
```

♡ **ngx_iconv**

➥ ***Character set*** **converter based on libiconv**

基于 **libiconv** 的字符编码转换器

```
location /api {
    # drizzle_pass/postgres_pass/etc

    iconv_filter from=UTF-8 to=GBK;
}
```

♡ Add some *sugar* of Lua

添加一点儿 Lua 糖果...

```
# nginx.conf
location = /hello {
    content_by_lua '
        ngx.say("Hello World")
    ';
}
```

```
$ curl 'http://localhost/hello'
Hello World
```

♡ or use an *external* Lua file
to keep things clean

或者使用外部的 Lua 文件让代码保持整洁

```
# nginx.conf
location = /hello {
    content_by_lua_file conf/hello.lua;
}
```

```
-- hello.lua
ngx.say("Hello World")
```

♡ *Reuse* existing *Nginx modules* in *Lua*
by means of Nginx subrequests

通过 Nginx 子请求实现在 Lua 中复用现有的 Nginx 模块

```
location = /memc {
    internal;
    memc_pass ...;
}

location = /api {
    content_by_lua '
        local resp = ngx.location.capture("/memc")
        if resp.status ~= 200 then
            ngx.exit(500)
        end
        ngx.say(resp.body)
    ';
}
```

♡ Multiple *concurrent* subrequests in Lua

Lua 中发起多个并发子请求

```
location = /api {
    content_by_lua '
        local res1, res2, res3 =
            ngx.location.capture_multi{
                {"/memc"}, {"/mysql"}, {"/postgres"}
            }
        ngx.say(res1.body, res2.body, res3.body)
    ';
}
```

♡ *Shared-memory* dictionary API in Lua

Lua 中的共享内存字典 API

```
lua_shared_dict dogs 10m;

server {
    location = /set {
        content_by_lua '
            local dogs = ngx.shared.dogs
            dogs:set("Tom", ngx.var.arg_n)
            ngx.say("OK")
        '
    }

    location = /get {
        content_by_lua '
            local dogs = ngx.shared.dogs
            ngx.say("Tom: ", dogs.get("Tom"))
        ';
    }
}
```

```
$ curl 'localhost/set?n=58'
OK

$ curl 'localhost/get'
Tom: 58
```

♡ *Non-buffered* response body output in Lua

在 Lua 中不带缓存的数据输出

```
-- api.lua

-- asynchronous emit data as a response body part
ngx.say("big data chunk")

-- won't return until all the data flushed out
ngx.flush(true)

-- ditto
ngx.say("another big data chunk")
ngx.flush(true)
```

♡ TCP *socket* API
for accessing upstream services in Lua

用于在 Lua 中访问上游服务的 TCP 套接字 API

```
local sock = ngx.socket.tcp()

sock:settimeout(1000)   -- one second

local ok, err = sock:connect("127.0.0.1", 11211)
if not ok then
    ngx.say("failed to connect: ", err)
    return
end
```

```
local bytes, err = sock:send("flush_all\r\n")
if not bytes then
    ngx.say("failed to send query: ", err)
    return
end

local line, err = sock:receive()
if not line then
    ngx.say("failed to receive a line: ", err)
    return
end

ngx.say("result: ", line)
```

```
local ok, err = sock:setkeepalive(60000, 500)
if not ok then
    ngx.say("failed to put the connection into pool "
        .. "with pool capacity 500 "
        .. "and maximal idle time 60 sec")
    return
end
```

♡ *Unix Domain Socket* is also supported

Unix 域套接字也是支持的

```
local sock = ngx.socket.tcp()
local ok, err = sock:connect("/tmp/some.sock")
if not ok then
    ngx.say("failed to connect to /tmp/some.sock: ", err)
    return
end
```

♡ The socket API is implemented atop *Lua coroutines*
and is synchronous and non-blocking

这些套接字 API 都是在 Lua 协程的基础上实现的，
是同步和非阻塞的。

♡ We call this socket API "*cosocket*"

我们把这组套接字 API 称为“cosocket”.

♡ *cosocket API* can also be used to read huge request body data

cosocket API 还可以用于读取巨大的请求体数据

```
local sock, err = ngx.req.socket()
if not sock then
    ngx.say("failed to get request socket: ", err)
    return
end

sock:settimeout(10000)  -- 10 sec timeout
```

```
while true do
    local chunk, err = sock:receive(4096)
    if not chunk then
        if err == "closed" then
            break
        end
        ngx.say("faile to read: ", err)
        return
    end
    process_chunk(chunk)
end
```

♡ High-level Lua libraries based on the *cosocket API*

基于 cosocket API 构建的高层次的 Lua 库

☺ lua-resty-mysql: pure Lua MySQL driver based on cosocket

https://github.com/agentzh/lua-resty-mysql

lua-resty-mysql: 基于 cosocket 的纯 Lua 实现的 MySQL 驱动

☺ lua-resty-memcached: pure Lua Memcached driver

https://github.com/agentzh/lua-resty-memcached

lua-resty-memcached: 基于 cosocket 的纯 Lua 实现的 Memcached 驱动

☺ lua-resty-redis: pure Lua Redis driver

https://github.com/agentzh/lua-resty-redis

lua-resty-redis: 基于 cosocket 的纯 Lua 实现的 Redis 驱动

☺ lua-resty-upload: support for *big* file uploading
(multipart/form-data)

https://github.com/agentzh/lua-resty-upload

lua-resty-upload: 大文件上传支持

♡ I've been hacking on *GitHub*!

http://github.com/agentzh

我在 GitHub 上玩开源！

♡ *Follow* me on Sina Weibo!

http://weibo.com/agentzh/

在新浪微博上关注我！

☺ *Any questions*? ☺