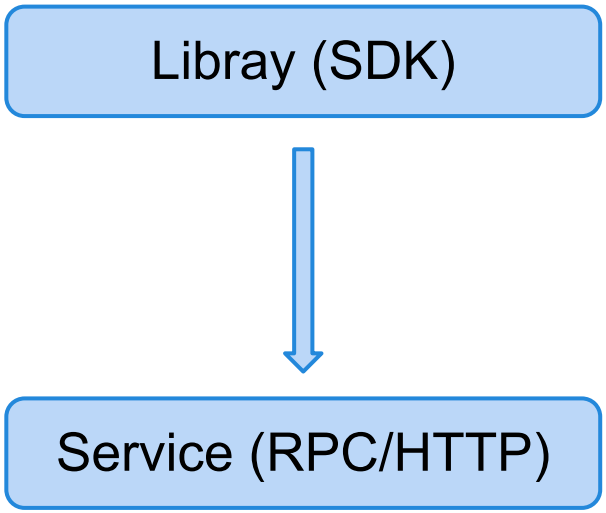
ECUG 2012

陈小玉 <smallfish>
@aliyun.com

Application Dev



Service

- cache
- database
- config
- route
- others

Problems

- permission
- rate limit
- filter
- rewrite
- route
- hot upgrade

Problems

- repeatability
- heavy
- test

Refactor

- backend
  - heavy language
- frontend
  - webserver module

Nginx

- fast
- hot-upgrade
- 3rd modules

HttpEchoModule

```
location /hello {
    echo "hello, world!";
}
```

```
$ curl http://localhost/hello
hello, world
```

HttpEchoModule

```
location /main {
    echo_location_async /sub1;
    echo_location_async /sub2;
}
location /sub1 {
    echo hello;
}
location /sub2 {
    echo world;
}
```

```
$ curl http://localhost/main
hello
world
```

HttpMemcModule

- keepalive
- upstream

HttpMemcModule

```
# GET /bar?cmd=get&key=cat
#
# POST /bar?cmd=set&key=dog
# My value for the "dog" key...
#
# DELETE /bar?cmd=delete&key=dog
# GET /bar?cmd=delete&key=dog
location /bar {
    set $memc_cmd $arg_cmd;
    set $memc_key $arg_key;
    set $memc_flags $arg_flags; # defaults to 0
    set $memc_exptime $arg_exptime; # defaults to 0
    # memc_cmds_allowed get set add delete flush_all;
    memc_pass 127.0.0.1:11211;
}
```

HttpDrizzleModule

- non-blocking
- support MySQL/Drizzle
- keepalive
- cluster upstream (simple round-robin)

HttpDrizzleModule

```
upstream cluster {
    # simple round-robin
    drizzle_server 127.0.0.1:3306 dbname=xx password=xx user=xx protocol=mysql;
    drizzle_server 127.0.0.1:1234 dbname=xx password=xx user=xx protocol=drizzle;
    drizzle_keepalive max=100 mode=single overflow=reject;
}

location /mysql {
    drizzle_query "SELECT * FROM cats";
    drizzle_pass cluster;
    rds_json on;
}
```

$ curl http://localhost/mysql

```
[{"id":1,"val":3.1415926}]
```

HttpDrizzleModule

```
location /mysql-pool-status {
    drizzle_status; # deny 127.0.0.1
}
```

```
$ curl http://localhost/mysql-pool/status
worker process: 15231
upstream backend
    active connections: 0
    connection pool capacity: 10
    overflow: reject
    ...
    servers: 1
    peers: 1
```

Not Enough

- expression (if)
- c module complex

Nginx Lua

- small
- fast (LuaJIT)
- power

$ wrk -c50 -r10000 -t4 -p1 http://127.0.0.1:8080/t

```
Making 10000 requests (HTTP/1.1) to http://127.0.0.1:8080/t
  4 threads and 50 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     4.22ms    2.01ms   8.45ms   75.00%
    Req/Sec     2.90k   307.79     3.00k    90.00%
  10008 requests in 763.57ms, 1.78MB read
Requests/sec:  13106.90
Transfer/sec:      2.34MB
```

Branch

- OpenResty openresty.org
- Tengine tengine.taobao.org
- Nginx www.nginx.org

HttpLuaModule

- non-blocking
- subrequest
- http-phase (rewrite/access/content/log)
- cosocket
- shared.DICT
- coroutine

HttpLuaModule

```
location /hello {
    default_type 'text/plain';
    content_by_lua "ngx.say('Hello,world!')";
}

# /nginx_var?a=hello,world
location /nginx_var {
    default_type 'text/plain';
    content_by_lua "ngx.print(ngx.var['arg_a'], '\\n')";
}
```

HttpLuaModule - subrequest

- not new request
- internals

HttpLuaModule - subrequest

```
location /lua {
    content_by_lua '
        local res = ngx.location.capture("/some_other_location")
        if res.status == 200 then
            ngx.print(res.body)
        end
    ';
}
```

HttpLuaModule - subrequest

```
local reqs = {}
table.insert(reqs, { "/mysql" })
table.insert(reqs, { "/postgres" })
table.insert(reqs, { "/redis" })
table.insert(reqs, { "/memcached" })

-- issue all the requests at once and wait until they all return
local resps = { ngx.location.capture_multi(reqs) }

-- loop over the responses table
for i, resp in ipairs(resps) do
    -- process the response table "resp"
end
```

HttpLuaModule - phase

- init_by_lua (init_by_lua_file)
- set_by_lua
- rewrite_by_lua
- access_by_lua
- content_by_lua
- log_by_lua

HttpLuaModule - phase

```
location /inline_concat {
    set $a "hello";
    set $b "world";
    set_by_lua $res "return ngx.arg[1] .. ngx.arg[2]" $a $b;
     # $ngx_prefix/conf/concat.lua contents:
     #    return ngx.arg[1]..ngx.arg[2]
     # set_by_lua_file $res conf/concat.lua $a $b;
    echo $res;
}
```

HttpLuaModule - phase

```
location /blah {
    access_by_lua '
        # if ngx.var.remote_addr == "132.5.72.3" then
        #    ngx.exit(ngx.HTTP_FORBIDDEN)
        # end
        local res = ngx.location.capture("/auth")
        if res.status == ngx.HTTP_OK then
            return
        end
        if res.status == ngx.HTTP_FORBIDDEN then
            ngx.exit(res.status)
        end
        ngx.exit(ngx.HTTP_INTERNAL_SERVER_ERROR)
    ';
    # proxy_pass/fastcgi_pass/postgres_pass/...
}
```

HttpLuaModule - cosocket

- coroutine
- tcp/udp/unix domain socket
- pool

HttpLuaModule - cosocket

```
location /test {
    resolver 8.8.8.8;

    content_by_lua '
        local sock = ngx.socket.tcp()
        local ok, err = sock:connect("www.google.com", 80)
        if not ok then
            ngx.say("failed to connect to google: ", err)
            return
        end
        ngx.say("successfully connected to google!")
        sock:close()
    ';
}
```

HttpLuaModule - shared.DICT

- shared with nginx worker
- like memcached
- LRU

HttpLuaModule - shared.DICT

```
lua_shared_dict dogs 10m;
server {
    location /set {
        content_by_lua '
            local dogs = ngx.shared.dogs
            dogs:set("Jim", 8)
            ngx.say("STORED")
        ';
    }
    location /get {
        content_by_lua '
            local dogs = ngx.shared.dogs
            ngx.say(dogs:get("Jim"))
        ';
    }
}
```

HttpLuaModule - coroutine

```
local function query_mysql()
    ngx.say("mysql done: ", cjson.encode(res))
end

local function query_memcached()
    ngx.say("memcached done: ", res)
end

local function query_http()
    local res = ngx.location.capture("/my-http-proxy")
    ngx.say("http done: ", res.body)
end

ngx.thread.spawn(query_mysql)          -- create thread 1
ngx.thread.spawn(query_memcached)  -- create thread 2
ngx.thread.spawn(query_http)           -- create thread 3
```

lua-resty-*

- base cosocket
- keepalive/pool

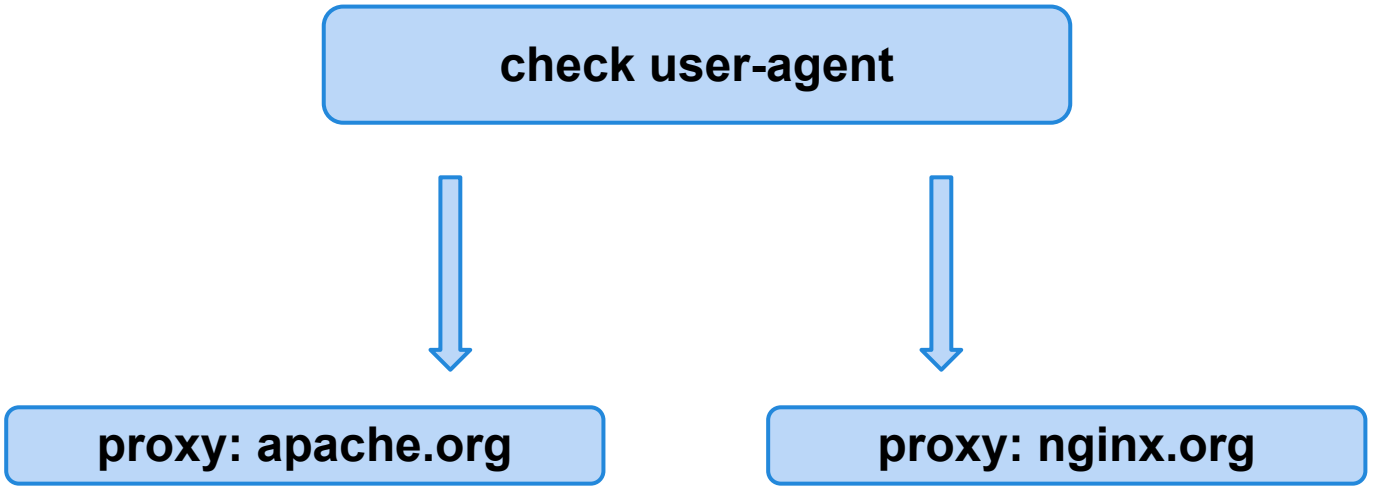
modules:

- memcache
- redis
- http
- beanstalkd
- gearman
- etc...

lua-resty-beanstalkd

```
local beanstalkd = require 'resty.beanstalkd'
local bean, err    = beanstalkd:new()
local ok, err      = bean:connect()
if not ok then
    ngx.say("failed to connect beanstalkd:", err)
    return
end
local id, err = bean:put("hello")
if not id then
    ngx.say("failed to put hello to tube, error:", err)
end
ngx.say("put hello to tube, id:", id)
bean:set_keepalive(0, 100)
-- bean:close()
```

Example - Dynamic Route



Example - Dynamic Route

```
upstream apache.org {
    server apache.org;
}

upstream nginx.org {
    server nginx.org;
}

location = /redis {
    set_unescape_uri $key $arg_key;
    redis2_query get $key;
    redis2_pass 127.0.0.1:6379;
}
```

Example - Dynamic Route

```
location / {
    set $target '';
    access_by_lua '
        local key = ngx.var.http_user_agent
        local res = ngx.location.capture( "/redis", { args = { key = key } })
        if res.status ~= 200 then
            ngx.exit(res.status)
        end
        local parser = require "redis.parser"
        local server, typ = parser.parse_reply(res.body)
        if typ ~= parser.BULK_REPLY or not server then
            ngx.exit(500)
        end
        ngx.var.target = server
    ';
    proxy_pass http://$target;
}
```

Example - Dynamic Route

$ ./redis-cli
redis> set foo apache.org
OK
redis> set bar nginx.org
OK

$ curl --user-agent foo localhost:8080
<apache.org home page goes here>

$ curl --user-agent bar localhost:8080
<nginx.org home page goes here>

More information

- http://openresty.org
- http://wiki.nginx.org/HttpLuaModule
- thanks @agentzh @chaoslawful and opensource

Questions?

thanks :-)