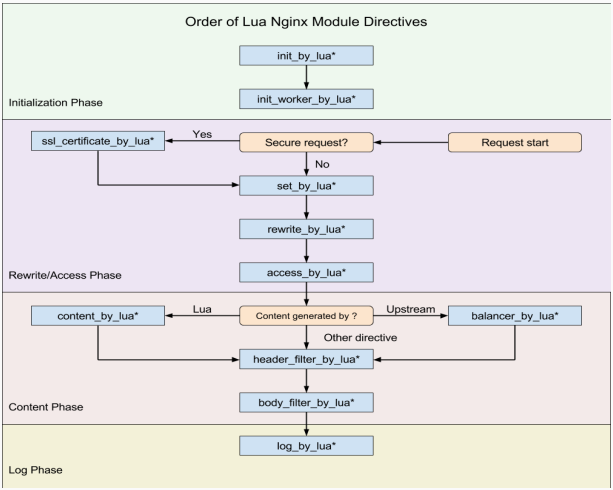
OpenResty 项目模块化最佳实践

更优雅的组织 OpenResty 项目模块

张金政
tianchaijz@gmail.com

December 23, 2017

ngx_lua 模块配置指令执行顺序和阶段



https://github.com/openresty/lua-nginx-module

一个简单的鉴权网关

```
-- module access
local _M = {}

function _M.check()
    -- ...
end

return _M
```

一个简单的鉴权网关

```lua
-- module access
local _M = {}

function _M.check()
    -- ...
end

return _M
```

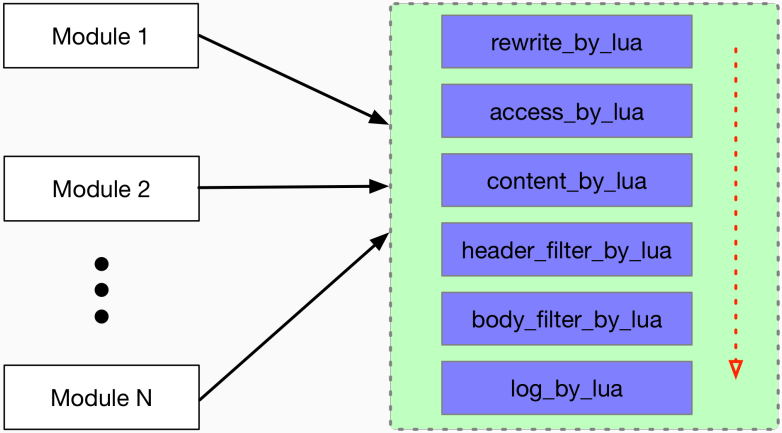
```
location / {
    access_by_lua_block {
        require("mod.access").check()
    }
}
```

复杂一点

一个复杂的网关

```
location / {
    rewrite_by_lua_block {
        if ngx.is_subrequest then
            return
        end
        -- ...
    }

    access_by_lua_block {
        -- module access
        -- other features
    }
}
```

一个复杂的网关



复杂的模块调用，耦合度高，不灵活

缺点

- 模块之间耦合度高
- 模块组织不灵活
- 很多复杂请求不好处理，无法发挥 OpenResty 的威力

那么问题来了

增删一个模块

其他 *location* 需要处理类似的逻辑

subrequest

模块框架

借鉴了 NGINX 的模块机制，将模块调用关系组织成一个链表

对请求分类，将请求处理流程注册到框架中

在主请求中进行路由

lua-resty-master

102 行 Lua 代码，简单强大

模块方法

- new
- run
- exec
- set_type
- next_handler
- *export*

一个简单的鉴权网关

```
local MODULE = "http.mod.access"

local core = require "resty.master.core"

local function check(r)
    -- do check here
    return r:next_handler(MODULE)
end

return {
    [core.ACCESS_PHASE] = { handler = check }
}

return _M
```

一个简单的鉴权网关

```
-- lua config
local _M

_M.handlers = {
    main = {
        "http.mod.access", -- access
    },
}

return _M
```

一个简单的鉴权网关

```
init_worker_by_lua_block {
    local config = require "config"
    local request = require "resty.master.request"
    for typ, hs in pairs(config.handlers) do
        request.register(typ, hs)
    end
}
```

一个简单的鉴权网关

```
location / {
    access_by_lua_block {
        local request = require "resty.master.request"
        local r = request.new("main")
        r:run(r.ACCESS_PHASE)
    }
}
```

新增加的模块只需配置一下

一个简单的鉴权网关

```
-- lua config
local _M

_M.handlers = {
    main = {
        "http.mod.access", -- access
        "http.mod.finalize", -- access
    },
}

return _M
```

一个简单的鉴权网关

```
-- lua config
local _M

_M.handlers = {
    main = {
        "http.mod.access", -- access
        "http.mod.finalize", -- access
    },
}

return _M
```

只需配置

一个复杂的网关

```lua
-- lua config
local _M

_M.handlers = {
    main = {
        "http.mod.router", -- rewrite
        "http.mod.access", -- access
        "http.mod.finalize", -- access
    },
    ["@subrequest"] = {
        "http.mod.subrequest" -- rewrite
    },
    ["@bypass"] = {
        "http.mod.finalize", -- access
    },
    ["@internal"] = {
        -- header_filter
        { "http.mod.common", { [HEADER_FILTER] = true } },
    },
}
```

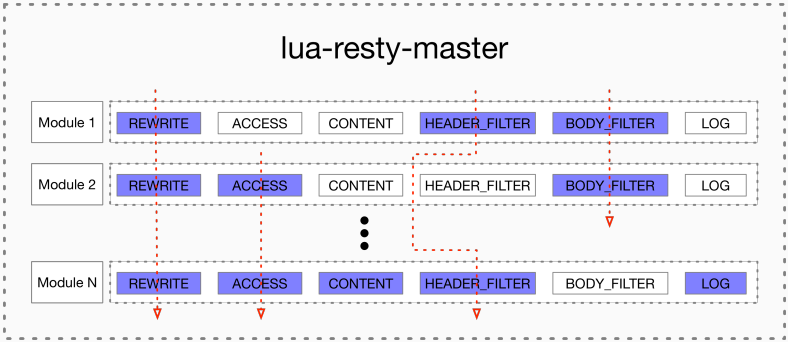
一个复杂的网关

```
local MODULE = "http.mod.router"

local core = require "resty.master.core"

local function route(r)
    if ngx.is_subrequest then
        return r:exec("@subrequest")
    end

    if expr then
      return r:exec("@bypass")
    end

    return r:next_handler(MODULE)
end

return {
    [core.REWRITE_PHASE] = { handler = route }
}
```

一个复杂的网关

```
rewrite_by_lua_block {
    local r = require("resty.master.request").new("main")
    ngx.ctx.request = r
    r:run(r.REWRITE_PHASE)
}

access_by_lua_block {
    local r = ngx.ctx.request
    r:run(r.ACCESS_PHASE)
}
```

一个复杂的网关



复杂的模块调用，耦合度低，灵活

优点

- `ngx_lua hooks` 都变成了占位符，无任何逻辑
- 配合 `ngx.exec` 能发挥出 OpenResty 更大的威力
- 项目组织完全配置化

为所欲为

只需修改配置

- 增删一个模块
- 其他 *location* 需要处理类似的逻辑
- *subrequest*

只需修改配置

- 增删一个模块
- 其他 *location* 需要处理类似的逻辑
- *subrequest*

这些问题已不存在

面向配置编程

优点不止这些

让开发者重新审视自己的模块

让开发精力集中到项目架构中

快乐编程

小技巧

小技巧

- 把处理好的 NGINX 变量都放在 *request* 对象中
- 一个专门的 *finalize* 模块做请求处理收尾，设置 NGINX 变量
- `ngx.exec` 时使用 *freelist* 算法优雅的保存 `ngx.ctx` 变量

优雅的保存 ngx.ctx

```
local stash_ctx = require("mod.ngx_ctx").stash
local function go(r)
    stash_ctx() -- save ngx.ctx
    -- r:set_type("fetch")
    return ngx.exec("/content")
end
```

优雅的保存 ngx.ctx

```
local stash_ctx = require("mod.ngx_ctx").stash
local function go(r)
    stash_ctx() -- save ngx.ctx
    -- r:set_type("fetch")
    return ngx.exec("/content")
end
```

```
location /content {
    content_by_lua_block {
        require("mod.ngx_ctx").apply()
        local r = ngx.ctx.request
        r:run(r.REWRITE_PHASE)
    }
}
```

序列化 request 对象

```
local function serialize(r)
    local rdata = {}
    for k, v in pairs(r) do
        if is_str(k) and string_sub(k, 1, 1) ~= "_"
            and not is_func(v) then
            rdata[k] = v
        end
    end

    return cmsgpack.pack(rdata)
end
```

使用案例

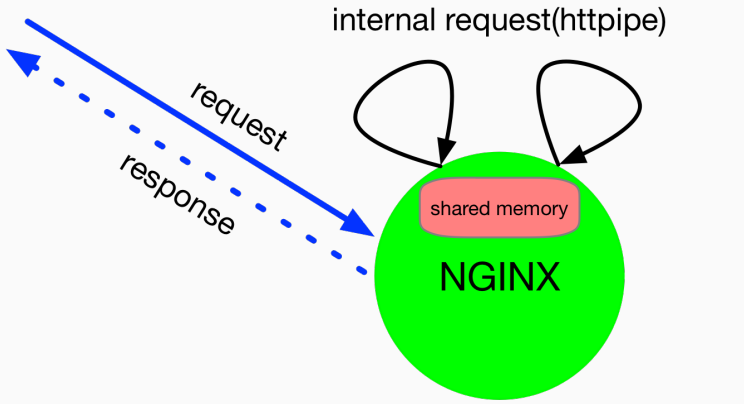
文件分片

- 优雅的隔离 *subrequest*

视频 seek

- 获取视频元数据 + 视频 *seek* 后的内容

视频 seek



serialize request context data to shared memory

CDN 私有请求

将私有请求直接路由到一个没有权限校验的处理流程中

- 绕过鉴权
- 文件预热
- 刷新 URL 转换

推而广之

在其他非 OpenResty 项目中尝试使用这种模块机制

https://github.com/openresty-fan/lua-resty-master

Questions?

广告