

---

## 一、前景回顾

- 1、location 匹配规则：精准 --》普通 --》正则（非正则除外）
- 2、代理传参：proxy\_pass = ip:port，将整个 path 部分传入 tomcat  
proxy\_pass = ip:port/xxx，只将匹配 path 的剩余部分传入 tomcat
- 3、rewrite 【break/last/redirect/permanent/null】  
中断无 location/中断 location/中断 302/中断 301/不中断 location
- 4、request 全阶段  
Server\_rewrite/Find\_config/Rewrite/access.../Content  
前一阶段命令全部执行完毕 ----》进行下一阶段命令
- 5、index 命令 ----》查找文件存在？ ----》是，刷新 location 匹配（回 Find\_config 阶段）

## 二、upstream--负载

语法格式：

```
upstream 负载名 {  
    [ip_hash;]  
    server ip:port [weight=数字] [down];  
    server ip:port [weight=数字];  
}  
[]内容为可选项
```

### 1、轮询（默认）

```
upstream order {  
    server 192.168.0.128:8383;  
    server 192.168.244.233:8383;  
}
```

不配置 weight（即默认 weight 均为 1）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉，能自动剔除。

### 2、weight

```
upstream order {  
    server 192.168.0.128:8383 weight=3;  
    server 192.168.244.233:8383 weight=1 down;  
}
```

指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况。

down 暂时不参与负载

### 3、ip\_hash

```
upstream order {  
    ip_hash;  
    server 192.168.0.128:8383;  
    server 192.168.244.233:8383;  
}
```

每个请求按访问 ip 的 hash 结果分配，这样同一客户端的请求总是发往同一个后端服务器，可以解决 session 的问题。

### 4、代理时的负载使用

格式：proxy\_pass http://负载名;

如下图，其传参到下游服务器的规则，与 proxy\_pass = http://ip:port 一样

```
location /order/enjoy {  
    ##后台请求为: http://192.168.0.128:8383/enjoy/getPage  
    ##调整后请求: http://test.enjoy.com/order/enjoy/getPage  
    ##故代理需要关闭path1的传递  
    proxy_pass http://order/enjoy;  
}
```

后续/enjoy, 第一部分location匹配path不要忘了

upstream别名

## 三、Openresty 使用

OpenResty 是一个全功能的 Web 应用服务器。它打包了标准的 Nginx 核心，常用的第三方模块以及大多数依赖项。可以把它看成是 Nginx 附加众多的第三方插件的合集。其主体是嵌入 lua 脚本的支持，让你能够使用 lua 灵活地处理运算逻辑。

本课程主要讲 lua 为 Nginx 带来的新的处理方式，及 OpenResty 组件的使用。

### 1、Openresty 的安装配置

#### 1.1、简易的 yum 安装方式

此方式简单，缺点是无法干预启停插件

```
yum install yum-utils
```

```
yum-config-manager --add-repo https://openresty.org/package/centos/openresty.repo
```

```
yum install openresty
```

## 1.2、源码安装方式

wget <https://openresty.org/download/openresty-1.15.8.1.tar.gz>

tar -zxvf openresty-1.15.8.1.tar.gz

##选择需要的插件启用, --with-Components 激活组件, --without 则是禁止组件

./configure --without-http\_redis2\_module --with-http\_iconv\_module

make && make install


vi /etc/profile ##加入 path 路径

```
export PATH=$PATH:/usr/local/openresty/nginx/sbin/
```

source /etc/profile ##生效配置

## 1.3、安装检测

nginx -V ##如下显示, 则表示安装成功



```
[root@test nginx]# nginx -V
nginx version: openresty/1.15.8.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
built with OpenSSL 1.0.2k-fips 26 Jan 2017
TLS SNI support enabled
configure arguments: --prefix=/usr/local/openresty/nginx --with-cc-opt=-O2 --add-module=../ngx_devel_kit-0.3.1rc1 --add-module=../echo-nginx-module-0.61 --add-module=../xss-nginx-module-0.06 --add-module=../ngx_coolkit-0.2 --add-module=../set-misc-nginx-module-0.32 --add-module=../form-input-nginx-module-0.12 --add-module=../encrypted-session-nginx-module-0.08 --add-module=../srcache-nginx-module-0.31 --add-module=../ngx_lua-0.10.15 --add-module=../ngx_lua_upstream-0.07 --add-module=../headers-more-nginx-module-0.33 --add-module=../array-var-nginx-module-0.05 --add-module=../memc-nginx-module-0.19 --add-module=../redis2-nginx-module-0.15 --add-module=../redis-nginx-module-0.3.7 --add-module=../rds-json-nginx-module-0.15 --add-module=../rds-csv-nginx-module-0.09 --add-module=../ngx_stream_lua-0.0.7 --with-ld-opt=-Wl,-rpath,/usr/local/openresty/luajit/lib --with-stream --with-stream_ssl_module --with-stream_ssl_preread_module --with-http_ssl_module
```

## 2、Lua 介入 Nginx 带来的基础 api

主要帮助对 http 请求取参、取 header 头、输出等

ngx.arg	指令参数, 如跟在 content_by_lua_file 后面的参数
ngx.var	request 变量, ngx.var.VARIABLE 引用某个变量
ngx.ctx	请求的 lua 上下文
ngx.header	响应头, ngx.header.HEADER 引用某个头
ngx.status	响应码
ngx.log	输出到 error.log
ngx.send_headers	发送响应头
ngx.headers_sent	响应头是否已发送
ngx.resp.get_headers	获取响应头
ngx.is_subrequest	当前请求是否是子请求
ngx.location.capture	发布一个子请求
ngx.location.capture_multi	发布多个子请求
ngx.print	输出响应
ngx.say	输出响应, 自动添加'\n'
ngx.flush	刷新响应

### 3、Lua 嵌入 Nginx 的时机阶段

Nginx 执行 lua 脚本片断时，需要明确指明执行的 nginx 阶段时机。主要有以下几种时机：

**set\_by\_lua\***：设置 nginx 变量，实现复杂的赋值逻辑

**rewrite\_by\_lua\***：实现转发、重定向等功能

**access\_by\_lua\***：IP 准入、接口访问权限等情况集中处理

**content\_by\_lua\***：接收请求处理并输出响应

**header\_filter\_by\_lua\***：设置 header 和 cookie

**body\_filter\_by\_lua\***：对响应数据进行过滤，如截断/替换等

### 4、Lua 基础功能使用介绍

#### 4.1 hello world

在 content 阶段，执行 lua 脚本，输出 hello, peter

```
location /hello {  
    ##ngx.say--输出内容print  
    content_by_lua 'ngx.say("Hello, Peter!")';  
}
```

在content阶段，去年这段lua脚本

#### 4.2、执行 lua 脚本文件

```
location /args_read {  
    ##执行lua文件脚本  
    content_by_lua_file /etc/nginx/lua/lua_args.lua;  
}
```

脚本文件位置

#### 4.3、lua 取 get 参数

页面请求路径：<http://lua.enjoy.com/args?a=20&b=50>

则 ngx.var.arg\_a 即取得 a 参数值，如下图：

```
location /args {
    ##ngx.var--取请求参数, arg_a指参数a
    content_by_lua_block {
        ngx.say(ngx.var.arg_a)
        ngx.say(ngx.var.arg_b)
    }
}
```

## 4.4、lua 取全量参数

请求: [http://lua.enjoy.com/args\\_read?a=20&b=50](http://lua.enjoy.com/args_read?a=20&b=50)

```
--lua的注释
--key-value形式取得所有的url上的参数--get型参数
local arg = ngx.req.get_uri_args()
for k,v in pairs(arg) do
    ngx.say("[GET ] ", k, " :", v)
end

--key-value形式取得所有post的参数
ngx.req.read_body()-- 解析 body 参数之前一定要先读取 body
local arg = ngx.req.get_post_args()
for k,v in pairs(arg) do
    ngx.say("[POST] ", k, " :", v)
end
```

读get和post参数

## 4.5、lua 取 request 中 header 信息

```
--读请求头信息
local headers = ngx.req.get_headers()
ngx.say("Host : ", headers.Host)
ngx.say("Host : ", headers["Host"])
ngx.say("-----")
for k,v in pairs(headers) do
    if type(v) == "table" then
        --table.concat是table操作, 意指将v内所有值合并
        ngx.say(k, " : ", table.concat(v, ","))
    else
        ngx.say(k, " : ", v)
    end
end
end
```

得到值合集

简易取值方式

遍历取值方式

## 4.6、给 lua 脚本传参

使用端传参：

```
location /setfile {  
    ##给lua脚本传递参数  
    set_by_lua_file $val "/etc/nginx/lua/set.lua" $arg_a $arg_b;  
    echo $val;  
}
```

接收脚本返回值      传入两个参数

脚本中借助 `ngx.arg` 取参

```
local a=tonumber(ngx.arg[1])  
local b=tonumber(ngx.arg[2])  
return a + b  
~
```

取第一个参数      取第二个参数

## 4.7、权限校验

一般校验动作，指定在 `access` 阶段执行脚本

```
location /access {  
    ##权限控制  
    access_by_lua_file "/etc/nginx/lua/access.lua";  
    echo "welcome $arg_name !";  
}
```

在access阶段执行

脚本处理

```
if ngx.var.arg_passwd == "123456"  
then  
    return  
else  
    ngx.exit(ngx.HTTP_FORBIDDEN)  
end
```

取http请求中的passwd参数，注意这里不是脚本传参      校验不通过，则403状态返回

## 4.8、内容过滤

Nginx 有时候，需要对下游服务生成的内容进行处理过滤，如下图

```
location /filter {  
    echo 'hello Peter';  
    echo 'you are welcome!';  
    ##内容过滤  
    body_filter_by_lua_file "/etc/nginx/lua/filter.lua";  
}
```

这两个输出内容，将交由下面的lua脚本进行过滤

脚本中的处理



```
--ngx.arg[1]是输出块内容
local chunk = ngx.arg[1]
if string.match(chunk, "hello") then
    ngx.arg[2] = true -- 设置为true, 表示输出结束 eof
    return
end
```

## 5、Lua 引入第三方模块的使用

OpenResty 提供了非常多的第三方插件，支持操作 redis/mysql 等服务，lua 使用它们的模式一般按以下流程

- ◆ require “resty/xxx”：导入模块功能，类似 java 中的 import 导入类
- ◆ local obj = xxx:new()：模块创建对象 obj
- ◆ local ok, err = obj:connect：对象连接到目标库
- ◆ obj:method：这里可以为所欲为，尽情操纵目标库了

### 1、Lua-resty-redis 连接 redis 用法

Lua-resty-redis 插件，对 Nginx 操作 redis 的支持十分强大，成熟的用法演示如下：  
基础的引入、连接动作

```
local redis = require "resty.redis"
--打开redis连接
local function open_redis()
    local red = redis:new()
    red:set_timeout(1000) -- 超时时间1 second
    local res = red:connect('192.168.0.128',6379)
    if not res then
        return nil
    end
    res = red:auth(123456) --密码校验
    if not res then
        return nil
    end
    red.close = close
    return red
end
```

redis 操作动作

```

local red = open_redis()
local value = red:get(key) --取值
red:set(key,val) --设新值
close(red)

--返回值到页面
ngx.say(key,':',value)

```

你想要查的key

给redis设置值

将值输出到页面

具体全量的程序，见源码配置包

## 2、Lua-resty-mysql 连接 mysql 数据库

引入模块、创建连接

```

local mysql = require "resty.mysql"
local cJSON = require "cjson"

--配置
local config = {
    host = "192.168.0.128",
    port = 3303,
    database = "enjoy",
    user = "root",
    password = "root"
}

--打开连接
local function open_mysql()
    local db, err = mysql:new()
    if not db then
        return nil
    end
    db:set_timeout(1000) -- 1 sec

    local ok, err, errno, sqlstate = db:connect(config)

    if not ok then
        return nil
    end
    db.close = close
    return db
end

```

引入模块

数据库配置，方便整体传入

创建连接

mysql 查询操作



```
local db = open_mysql()
local sql = "select * from t_account "
--设置中文编码
ngx.header['Content-Type']="text/html;charset=UTF-8"

local res, err, errno, sqlstate = db:query(sql)
close(db)
if not res then
    ngx.say(err)
    return {}
end

--json方式输出
ngx.say(cjson.encode(res))
```

查询sql

查询得到结果集

将结果集转换为json串，输出到页面