OpenResty

1.入门篇

2.Lua入门

3.Nginx

4.OpenResty入门

4.1 Window 平台安装

4.2 CentOS 平台安装

4.3 Ubuntu 平台安装

4.4 Mac 平台安装

4.5 Hello World　配置使用

4.6 与其他 Location 配合

内部调用

内部调用限制，internal关键字

普通调用，ngx.location.capture

队列调用，ngx.location.capture_queue

并行调用，队列调用 ngx.location.capture_multi

流水线方式跳转　流水线模式，逐层过滤，处理 纯粹内部跳转，ngx.exec

外部重定向　重定向 ngx.redirect 地址栏URL变化了，并且是可以跨域名的

4.7 获取 uri 参数

获取请求 URL 参数

Get参数 ngx.req.get_uri_args 返回表数据

Post参数 ngx.req.read_body 再执行 ngx.req.get_post_args

传递请求 uri 参数

使用 ngx.encode_args 来进行编码

```
local res = ngx.location.capture('/print_param',
{
    method = ngx.HTTP_POST,
    args = ngx.encode_args({a=1,b='2&'}),
    body = ngx.encode_args({c=3,d='4&'})
})
```

不编码的情况

```
local res = ngx.location.capture('/print_param',{
    method=ngx.HTTP_POST,
    args='a=1&b=2%26', #也可以直接使用 {a=1,b='2&'}
    body='c=3&4%26'
})
```

获取请求 Body

全局设置，在 Sever块中

配置 lua_need_request_body on; 指令

调用 ngx.req.get_body_data 读取

局部使用，在 location 块中　调用接口 ngx.req.read_body() 开启　调用 ngx.req.get_body_data() 读取

偶尔读取不到：请求体已经被存入临时文件　强制配置 client_body_in_file_only on; 在 server 块中

```
ngx.req.read_body()
if nil == data then
    local file_name = ngx.req.get_body_file()
    ngx.say('>> temp file:',file_name)
    if file_name then
        data = getFile(file_name)
    end
end
```

4.8 输出响应体

ngx.say，ngx.print 都为异步输出

直接调用　等待后，直接输出全部内容

ngx.flush() 刷新缓冲区　等待后，即时刷新缓冲区数据

在 server块中，配置 lua_code_cache off; 指令

大响应体的输出

输出内容本身体积很大

利用 HTTP1.1 的 CHUNKEN 编码完成把一个大的响应体拆分成多个小的应答体，分批，有节制的响应。

```
local data
while true do
    data = file:read(1024)
    if nil == data then
        break
    end
    ngx.print(data)
    ngx.flush(true)
end
file:clost()
```

内容碎片拼凑，碎片数量庞大

利用 ngx.print 的特性，输入参数可以是单个或多个字符串参数，也可以是 table 对象，用数组的方式把碎片数据统一起来

```
local table={"hello, ",{"world: ",true,"or ",false,{": ",nil}}}
ngx.print(table)
```

4.9 日志输出

标准日志输出

使用 ngx.log 输出日志

在 nginx.conf 文件中，可以设定日志文件的配置，error_log logs/error.log error； 指定这个类别的日志输出

使用 ngx.log(ngx.ERR,"num:",num)，ngx.log(ngx.INFO,"string:",str)

日志中返回信息很多（环境信息），可使用 nginx 的 log_format 设定，print语句默认是 INFO 级别的

日志级别类型：ngx.STDER, ngx.EMERG, ngx.ALERT, ngx.CRIT, ngx.ERR, ngx.WARN, ngx.NOTICE, ngx.INFO, ngx.DEBUG

网络日志输出 P201　使用 lua_resty_logger_socket 库 特点：非阻塞IO，容错，全网络操作，日志类型，集体提交

4.10 简单 API Server 框架

结构优化

参数验证的类库（访问文件） access_by_lua_file xxxxx；（location 块中）

内容生成的类库（内容引入这个文件） content_by_lua_file xxxxx；location 块中）

设定 lua 搜索路径 lua_package_path xxxxx； 在 http块中（指令）

在 lua 中引入目录地址使用“.”来代替层级。 #require(“comm.parcn”)

ngx.exit 来退出处理，并返回指定错误信息，ngx.exit(ngx.HTTP_BAD_REQUEST)

4.11 使用 Nginx 内置绑定变量

使用 ngx.var.VARIABLE 变量名 格式。使用前必须是定义好的，变量可读部分可写

tonumber(ngx.var.arg_a) 直接使用参数

可使用的上下文

set_by_lua

rewrite_by_lua

access_by_lua

content_by_lua

hader_by_lua

body_filter_by_lua

常用变量 P207

4.12 子查询

使用 ngx.location.capture(uri) 函数

与 301/302 ngx.redirect 不同

与 ngx.exec 内部重定向不同

使用前应读取完整的 HTTP请求

通过调用 ngx.req.read_body

设置 lua_need_request_body 指令为 on

发起非阻塞的内部请求，来访问目标 location

返回的对象是个表，包含4个值

res.status 响应状态码

res.header 响应头，多值的 响应通使用 lua 的数组

res.body 响应体数据，可能被截断

res.truncuted 截断布尔值，判断

子查询请求配置

URI请求串可与 URI本身连在一起. res.header["set-cookie"] 将存储表 {"a=3","foo=bar"}

可设置请求类型和参数

```
res=ngx.location.capture(
    '/xxxx/xxxx',
    {
        method=ngx.HTTP_POST,
        body="hellow,world ",
        args={b=2,c=";"}
    }
)
```

method 请求方法默认 get

HTTP_GET,HTTP_HEAD,HTTP_PUT,HTTP_POST,HTTP_DELETE,HTTP_OPTIONS

HTTP_MKCOL,HTTP_COPY,HTTP_MOVE,HTTP_PROPFIND,HTTP_PROPPATCH

HTTP_LOCK,HTTP_UNLOCK,HTTP_PATCH,HTTP_TRACE

body 请求的内容

args 设置参数，可使用表的形式，也可直接使用转义后的字符串

注意事项

创建的子请求默认继承当前请求的所有头信息，可能会有副作用，使用 proxy_pass_request_header off; 指令关闭

cpture/capture_multi 函数无法抓取一些指令

add_before_body

add_after_body

auth_request

echo_location

echo_location_async

echo_subrequest

echo_subrequest_sync

4.13 不同阶段共享变量

通过共享内存方式完成不同工作进程数据共享

通过 lua 模块完成单个进程内不同请求块内的数据共享，使用 **ngx.ctx.xxx** 来使用

但是 ngx.ctx 查表需要更贵的元方法调用，性能不高，应多使用 函数传递参数的方式要更直接一些。ngx.ctx保存的是指定请求资源，这个变量时不能直接共享给其他请求使用的。

不同阶段，在同一请求内，不同进程快之间使用

rewrite_by_lua_block

access_by_lua_block

content_by_lua_block

** 每个请求，子请求，都有一份自己的 ngx.ctx 表

4.14 防止SQL注入

保单提交 SQL语句，执行非法操作

在 lua 中执行对数据库的操作，对参数调用 ndk.set_var.set_quote_sql_str(xxx) 函数来进行过滤，

PostgreSQL有不同的过滤函数 ndk.set_var.set_quote_pgsql_str 函数

4.15 如何发起新 HTTP 请求

OpenResty主要应用是 APIServer，需要高效与其他HttpServer调用

利用 proxy_pass 利用负载均衡，提供对外的调用

利用 cosocket 这是一个第三方的 lua 库，该库完成了 连接池，HTTP请求等一系列动作，可以直接调用，不用中转

5. LuaRestyRedisLibrary

Redis连接库操作

```
local redis = require "resty.redis"
local red = redis:new()

red:set_timeout(1000)
local ok,err = red:connect("127.0.0.1",6379)
ok,err = red:set("dog","an animal")

local res,err = red:get("dog")
local ok,err = red:set_keepalive(10000,100)
```

使用Redis 管道

```
red:init_pipeline()
red:set("cat","Marry")
red:set("horse","Bob")
red:get("cat")
red:get("horse")
local results,err = red:commit_pipeline()

if not results then
    ngx.say("failed to commit the piplined requests: ",err)
    return
end

for i,res in ipairs(results) do
    if type(res) == "table" then
        if res[1] == false then
            ngx.say("failed to run command ",i,": ",res[2])
        else
            ngx.say("the value:",res[2])
        end
    else
        ngx.say("the value type is scalar :",res)
    end
end
```

访问有授权验证的 Redis

tcpsock:get_reused_times()方法：判断使用的连接是否是从连接池中获取的，连接池中的连接返回的这个值都是非0的。

```
local count
count,err = red:get_reused_times()
if 0 == count then
    ok,err = red:auth("password")
    if not ok then
        ngx.say("failed to auth:",err)
        return
    end
elseif err then
    ngx.say("failed to get reused times: ",err)
    return
end
```

select + set_keepalive 组合操作引起的读写错误

主要问题在于：使用 select 命令设置了 db的连接，在放回连接池后，还是会更改了db的选择的。下一个请求在从连接池中使用时，如果没有设置 db命令。则在使用时，就会使用选择后的 db，导致数据操作不同。处理原则。：谁制造问题，谁把问题遗留尾巴擦干净。

Redis 接口的二次封装

1.new、connect 函数合体，使用时只负责申请　2.默认 Redis数据库连接地址，可以配置

3.每次Redis调用完毕，自动释放 Redis 连接池　4.支持 Redis的 pipeline

Redis 接口的二次封装（发布订阅）

pipeline 压缩请求数量

需要了解 Redis的长连接机制（每个Redis命令都以一个TCP请求发送给Redis，每个命令都以一个数据包发送）pipeline机制能够将多个命令汇聚到一个请求中，可以有效减少请求数量，减少网络延时。

Script压缩复杂请求

Redis 中的 script压缩复杂命令，在 Redis命令中，嵌入 Lua脚本，实现一些复杂操作。

可以使用脚本相关的 Redis 命令[EVAL,EVALSHA,SCRIPT EXISTS,SCRIPT FLUSH,SCRIPT KILL,SCRIPT LOAD]

基本语法：EVAL script numkeys key[key...] arg[arg...]

说明：第一个参数是一段 lua脚本程序，第二个参数 numkeys是参数个数，key表示在脚本中用到的那些 Redis键

在Lua脚本中通过全局变量KEYS数组访问，arg不是键名参数的附加参数，Lua脚本中通过 ARGV数组访问

例：eval "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1 key2 first second

```
local res,err = red:eval([[
    local info = redis.call('get',KEYS[1])
        info = cjson.decode(info)
        local g_id = info.gid
        local g_info = redis.call('get',g_id)
        return g_info
    ]],1,id)
```

6.LuaCjsonLibrary

7.PostgresNginxModule

8.LuaRestyDNSLibrary

9.LuaRestyLock

10.stream_lua_module

11.balancer_by_lua

12.OpenResty 与 SSL

13.测试

14.Web服务

15.火焰图

16.OPenResty 周边