[**第八章 流量复制/AB测试/协程**](http://jinnianshilongnian.iteye.com/blog/2190343)

**博客分类：**

* [跟我学Nginx+Lua开发](http://jinnianshilongnian.iteye.com/category/333854)

[nginx](http://www.iteye.com/blogs/tag/nginx)[lua](http://www.iteye.com/blogs/tag/lua)[coroutine](http://www.iteye.com/blogs/tag/coroutine)[ngx\_lua](http://www.iteye.com/blogs/tag/ngx_lua)[openresty](http://www.iteye.com/blogs/tag/openresty)

**流量复制**

在实际开发中经常涉及到项目的升级，而该升级不能简单的上线就完事了，需要验证该升级是否兼容老的上线，因此可能需要并行运行两个项目一段时间进行数据比对和校验，待没问题后再进行上线。这其实就需要进行流量复制，把流量复制到其他服务器上，一种方式是使用如[tcpcopy](https://github.com/session-replay-tools/tcpcopy)引流；另外我们还可以使用nginx的HttpLuaModule模块中的ngx.location.capture\_multi进行并发执行来模拟复制。

构造两个服务

**Java代码  [收藏代码](javascript:void())**

1. location /test1 {
2. keepalive\_timeout 60s;
3. keepalive\_requests 1000;
4. content\_by\_lua '
5. ngx.print("test1 : ", ngx.req.get\_uri\_args()["a"])
6. ngx.log(ngx.ERR, "request test1")
7. ';
8. }
9. location /test2 {
10. keepalive\_timeout 60s;
11. keepalive\_requests 1000;
12. content\_by\_lua '
13. ngx.print("test2 : ", ngx.req.get\_uri\_args()["a"])
14. ngx.log(ngx.ERR, "request test2")
15. ';
16. }

通过ngx.location.capture\_multi调用

**Java代码  [收藏代码](javascript:void())**

1. location /test {
2. lua\_socket\_connect\_timeout 3s;
3. lua\_socket\_send\_timeout 3s;
4. lua\_socket\_read\_timeout 3s;
5. lua\_socket\_pool\_size 100;
6. lua\_socket\_keepalive\_timeout 60s;
7. lua\_socket\_buffer\_size 8k;
9. content\_by\_lua '
10. local res1, res2 = ngx.location.capture\_multi{
11. { "/test1", { args = ngx.req.get\_uri\_args() } },
12. { "/test2", { args = ngx.req.get\_uri\_args()} },
13. }
14. **if** res1.status == ngx.HTTP\_OK then
15. ngx.print(res1.body)
16. end
17. **if** res2.status ~= ngx.HTTP\_OK then
18. --记录错误
19. end
20. ';
21. }

此处可以根据需求设置相应的超时时间和长连接连接池等；ngx.location.capture底层通过cosocket实现，而其支持Lua中的协程，通过它可以以同步的方式写非阻塞的代码实现。

此处要考虑记录失败的情况，对失败的数据进行重放还是放弃根据自己业务做处理。

**AB测试**

AB测试即多版本测试，有时候我们开发了新版本需要灰度测试，即让一部分人看到新版，一部分人看到老版，然后通过访问数据决定是否切换到新版。比如可以通过根据区域、用户等信息进行切版本。

比如京东商城有一个cookie叫做\_\_jda，该cookie是在用户访问网站时种下的，因此我们可以拿到这个cookie，根据这个cookie进行版本选择。

比如两次清空cookie访问发现第二个数字串是变化的，即我们可以根据第二个数字串进行判断。

\_\_jda=122270672.1059377902.1425691107.1425691107.1425699059.1

\_\_jda=122270672.556927616.1425699216.1425699216.1425699216.1。

判断规则可以比较多的选择，比如通过尾号；要切30%的流量到新版，可以通过选择尾号为1，3,5的切到新版，其余的还停留在老版。

**1、使用map选择版本**

**Java代码  [收藏代码](javascript:void())**

1. map $cookie\_\_\_jda $ab\_key {
2. **default**                                       "0";
3. ~^\d+\.\d+(?P<k>(1|3|5))\.                    "1";
4. }

使用[map](http://nginx.org/cn/docs/http/ngx_http_map_module.html)映射规则，即如果是到新版则等于"1"，到老版等于“0”； 然后我们就可以通过ngx.var.ab\_key获取到该数据。

**Java代码  [收藏代码](javascript:void())**

1. location /abtest1 {
2. **if** ($ab\_key = "1") {
3. echo\_location /test1 ngx.var.args;
4. }
5. **if** ($ab\_key = "0") {
6. echo\_location /test2 ngx.var.args;
7. }
8. }

此处也可以使用proxy\_pass到不同版本的服务器上

**Java代码  [收藏代码](javascript:void())**

1. location /abtest2 {
2. **if** ($ab\_key = "1") {
3. rewrite ^ /test1 **break**;
4. proxy\_pass http://backend1;
5. }
6. rewrite ^ /test2 **break**;
7. proxy\_pass http://backend2;
8. }

**2、直接在Lua中使用lua-resty-cookie获取该Cookie进行解析**

首先下载lua-resty-cookie

**Java代码  [收藏代码](javascript:void())**

1. cd /usr/example/lualib/resty/
2. wget https://raw.githubusercontent.com/cloudflare/lua-resty-cookie/master/lib/resty/cookie.lua

**Java代码  [收藏代码](javascript:void())**

1. location /abtest3 {
2. content\_by\_lua '
4. local ck = require("resty.cookie")
5. local cookie = ck:**new**()
6. local ab\_key = "0"
7. local jda = cookie:get("\_\_jda")
8. **if** jda then
9. local v = ngx.re.match(jda, [[^\d+\.\d+(1|3|5)\.]])
10. **if** v then
11. ab\_key = "1"
12. end
13. end
15. **if** ab\_key == "1" then
16. ngx.exec("/test1", ngx.var.args)
17. **else**
18. ngx.print(ngx.location.capture("/test2", {args = ngx.req.get\_uri\_args()}).body)
19. end
20. ';
22. }

 首先使用[lua-resty-cookie](https://github.com/cloudflare/lua-resty-cookie)获取cookie，然后使用ngx.re.match进行规则的匹配，最后使用ngx.exec或者ngx.location.capture进行处理。此处同时使用ngx.exec和ngx.location.capture目的是为了演示，此外没有对ngx.location.capture进行异常处理。

**协程**

Lua中没有线程和异步编程编程的概念，对于并发执行提供了协程的概念，个人认为协程是在A运行中发现自己忙则把CPU使用权让出来给B使用，最后A能从中断位置继续执行，本地还是单线程，CPU独占的；因此如果写网络程序需要配合非阻塞I/O来实现。

ngx\_lua 模块对协程做了封装，我们可以直接调用ngx.thread API使用，虽然称其为“轻量级线程”，但其本质还是Lua协程。该API必须配合该ngx\_lua模块提供的非阻塞I/O API一起使用，比如我们之前使用的ngx.location.capture\_multi和lua-resty-redis、lua-resty-mysql等基于cosocket实现的都是支持的。

通过Lua协程我们可以并发的调用多个接口，然后谁先执行成功谁先返回，类似于BigPipe模型。

**1、依赖的API**

**Java代码  [收藏代码](javascript:void())**

1. location /api1 {
2. echo\_sleep 3;
3. echo api1 : $arg\_a;
4. }
5. location /api2 {
6. echo\_sleep 3;
7. echo api2 : $arg\_a;
8. }

 我们使用echo\_sleep等待3秒。

**2、串行实现**

**Java代码  [收藏代码](javascript:void())**

1. location /serial {
2. content\_by\_lua '
3. local t1 = ngx.now()
4. local res1 = ngx.location.capture("/api1", {args = ngx.req.get\_uri\_args()})
5. local res2 = ngx.location.capture("/api2", {args = ngx.req.get\_uri\_args()})
6. local t2 = ngx.now()
7. ngx.print(res1.body, "<br/>", res2.body, "<br/>", tostring(t2-t1))
8. ';
9. }

即一个个的调用，总的执行时间在6秒以上，比如访问http://192.168.1.2/serial?a=22

**Java代码  [收藏代码](javascript:void())**

1. api1 : 22
2. api2 : 22
3. 6.0040001869202

**3、ngx.location.capture\_multi实现**

**Java代码  [收藏代码](javascript:void())**

1. location /concurrency1 {
2. content\_by\_lua '
3. local t1 = ngx.now()
4. local res1,res2 = ngx.location.capture\_multi({
5. {"/api1", {args = ngx.req.get\_uri\_args()}},
6. {"/api2", {args = ngx.req.get\_uri\_args()}}
8. })
9. local t2 = ngx.now()
10. ngx.print(res1.body, "<br/>", res2.body, "<br/>", tostring(t2-t1))
11. ';
12. }

直接使用ngx.location.capture\_multi来实现，比如访问http://192.168.1.2/concurrency1?a=22

**Java代码  [收藏代码](javascript:void())**

1. api1 : 22
2. api2 : 22
3. 3.0020000934601

**4、协程API实现**

**Java代码  [收藏代码](javascript:void())**

1. location /concurrency2 {
2. content\_by\_lua '
3. local t1 = ngx.now()
4. local function capture(uri, args)
5. **return** ngx.location.capture(uri, args)
6. end
7. local thread1 = ngx.thread.spawn(capture, "/api1", {args = ngx.req.get\_uri\_args()})
8. local thread2 = ngx.thread.spawn(capture, "/api2", {args = ngx.req.get\_uri\_args()})
9. local ok1, res1 = ngx.thread.wait(thread1)
10. local ok2, res2 = ngx.thread.wait(thread2)
11. local t2 = ngx.now()
12. ngx.print(res1.body, "<br/>", res2.body, "<br/>", tostring(t2-t1))
13. ';
14. }

使用[ngx.thread.spawn](http://wiki.nginx.org/HttpLuaModule#ngx.thread.spawn)创建一个轻量级线程，然后使用[ngx.thread.wait](http://wiki.nginx.org/HttpLuaModule#ngx.thread.wait)等待该线程的执行成功。比如访问http://192.168.1.2/concurrency2?a=22

**Java代码  [收藏代码](javascript:void())**

1. api1 : 22
2. api2 : 22
3. 3.0030000209808

其有点类似于Java中的线程池执行模型，但不同于线程池，其每次只执行一个函数，遇到IO等待则让出CPU让下一个执行。我们可以通过下面的方式实现任意一个成功即返回，之前的是等待所有执行成功才返回。

**Java代码  [收藏代码](javascript:void())**

1. local  ok, res = ngx.thread.wait(thread1, thread2)

Lua协程参考资料

[《Programming in Lua》](http://www.lua.org/pil/)

<http://timyang.net/lua/lua-coroutine-vs-java-wait-notify/>

<https://github.com/andycai/luaprimer/blob/master/05.md>

<http://my.oschina.net/wangxuanyihaha/blog/186401>

<http://manual.luaer.cn/2.11.html>