偶遇Lua——在nginx conf中编程

Written on 12 20, 2014

原来，我从未用过Lua这门语言，根据一些文章的介绍以及周围一些朋友的描述，对Lua的印象一直停留在：**小巧、适用于底层、性能一般般**。

最近，在开发过程中，需要对已有模块的HTTP接口进行修改，并且小幅度更改请求和响应的内容。直接对已有模块进行修改，显得不够灵活，而且还有影响线上服务的风险，不够优雅！

在同事的推荐之下，决定尝试一下。原来，Lua是可以直接在`nginx.conf`中编写的，居然可以这么玩！简单说下我是如何通过Lua + nginx + OpenResty来快速解决一些问题的。

对每一个请求加上唯一的请求ID

需求的整体背景是：需要将一个内部调用的服务接口转为外部调用。在系统内部，让服务调用方生成一个唯一的请求ID，这个是很正常的。但如果外部调用服务的用户，似乎没有义务去这么做。因此，就需要解决第一个问题：对于过来的外部请求，自动生成请求ID。

那么，修改请求参数的时机应该就是：请求打到nginx、转发至真正的后端模块之前。如果使用nginx的朋友应该都比较清楚，对于nginx请求转发，`nginx.conf`一般是这么写：

```
upstream xxx_api-backend {
    server www.server1.com:8000;
    server www.server2.com:8000;
```

```
    server www.server3.com:8000;
}

server {
    listen      80;
    server_name api.xxx.com;
    location ^~ /some_path {
        proxy_pass              http://xxx_api-backend;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;


    }
}
```

如果需要在请求到后端前修改请求参数，只需要在`location`范围内使用`rewrite_by_lua`语句即可，例如：

```
location ^~ /some_path {
    rewrite_by_lua '
        -- 引入一些Lua库
        local resty_random = require "resty.random"

        local str = require "resty.string"
        -- 生成随机字符串
        local random = resty_random.bytes(16)
        local args=ngx.req.get_uri_args()
        -- 添加一个请求字段
        args["requestid"] = "api_prefix"..str.to_hex(random)
        ngx.req.set_uri_args(args)
    ';
    proxy_pass              http://xxx_api-backend;
```

```
    proxy_set_header        Host            $host;
    proxy_set_header        X-Real-IP       $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

这样，我们就成功的在request参数中添加了`requestid`这个字段。

读取HTTP Response

修改了Request，下面需要做的就是获取后端返回的Response了。Google了一些资料，都是类似这里的讨论，得到的结论是：

无法直接通过nginx配置截获一份完整的Response data，你所能获取到的都是一份一份的块（chunk）数据。

但是我的Response是JSON格式的，没办法获取完整的Response数据，就没办法进行解析并修改呀，着急。不过后来，还是找到了解决方案：`sub-requests`。它会向指定的地址发送非阻塞IO请求，就像在其他地方直接请求这个地址，方法的返回数据包括了`Status Code`、`Header`以及`Body`，这样一来，我们就能获取到完整的返回数据了。例子如下：

```
location ^~ /some_path {
    content_by_lua '
        ngx.req.read_body()
        local data = ngx.req.get_body_data()
        local args = ngx.req.get_uri_args()
        local res = ngx.location.capture("/some_other_path", {method = ngx.
        ngx.say(res.body)
    ';
}
```

解析JSON

获取到了Response data，下面就要解析JSON并修改了，这里直接使用lua调用`cjson`库即可，简单到不能再简单。

```
location ^~ /some_path {
    content_by_lua '
        ngx.req.read_body()
        local data = ngx.req.get_body_data()
        local args = ngx.req.get_uri_args()
        local res = ngx.location.capture("/some_other_path", {method = ngx.
        local cjson = require "cjson"
        value = cjson.decode(res.body)
        -- 根据原始JSON中的errorMsg字段值，来增加一个code字段
        if value["errorMsg"] == "" then
            value["code"] = 0
        else
            value["code"] = 2
        end
        -- 输出修改过的JSON
        ngx.say(cjson.encode(value))
    ';
}
```

总结

就这样，在不到两个小时的时间内，使用nginx + Lua + OpenResty快速解决了问题。使用下来，感觉上手极快，没什么学习成本。在`nginx.conf`中直接使用Lua，可以快速解决问题或进行测试。

无疑，对于处理逻辑简单、底层的场景来说，Lua是十分合适的。

Reference

- Scripting Nginx with Lua