OpenResty TCP 服务代理和动态路由

杭州 OpenResty Meetup
20 December 2017

黄励博(huangnauh)
又拍云

slide

```
https://huangnauh.github.io/2017OpenRestyMeetup.html
https://github.com/huangnauh/slides
http://go-talks.appspot.com/github.com/huangnauh/slides/OpenRestyMeetup.slide
```

测试环境

```
git clone git@github.com:huangnauh/slardar.git
git co docker
docker-compose up
```

docker-compose.yml:

```
version: '3.2'

services:
    slardar:
        links:
            - consul
            - mysql3307
            - mysql3306
        image: huangnauh/slardar:test
        volumes:
            - type: bind
              source: ./nginx/conf
              target: /usr/local/slardar/nginx/conf
```

NGINX

NGINX 1.9 开始引入 stream 模块, 实现四层协议的转发和代理, 和 http 类似, 也是采用分阶段处理请求的方式

阶段	简介
Post-accept	接收客户端连接后的第一个阶段
Pre-access	访问的初步检查
Access	实际数据处理之前的客户端访问限制
SSL	SSL 处理
Preread	将数据的初始字节读入预读缓冲区中
Content	实际处理数据
Log	记录请求处理结果的最后阶段

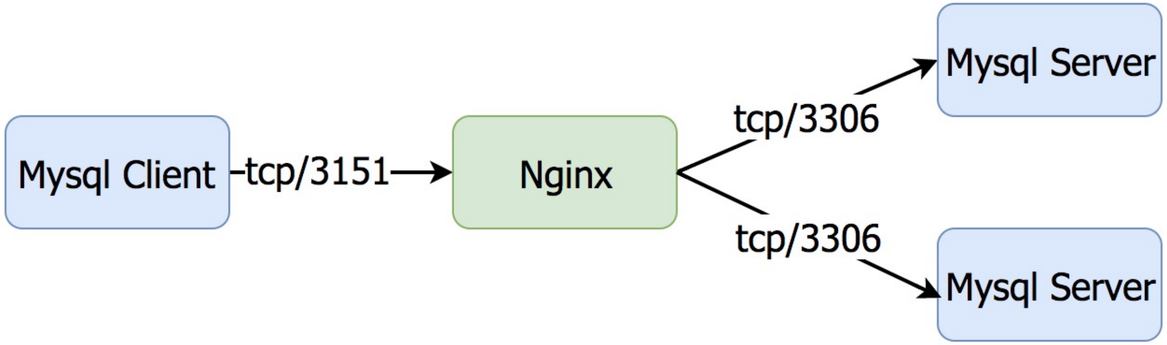
Image credit: stream_processing (http://nginx.org/en/docs/stream/stream_processing.html)

SNI 代理

比如, 模块 ngx_stream_ssl_preread (http://nginx.org/en/docs/stream/ngx_stream_ssl_preread_module.html) 在 preread 阶段, 从 ClientHello 消息中提取信息

```
stream {
    server {
        listen 443;
        ssl_preread on;
        proxy_pass $ssl_preread_server_name:$server_port;
    }
}
```

TCP 负载均衡



```
stream {
    upstream mysql {
        #hash $remote_addr consistent;
        server 127.0.0.1:3306;
        server 127.0.0.1:3307;
    }
    server {
        listen 3151;
        proxy_pass mysql;
    }
}
```

TCP 负载均衡

测试:

```
$ echo "show variables where variable_name = 'hostname'" \
 | mysql --skip-column-names -h 127.0.0.1 -P 3151 -uroot -proot

hostname    e3ac73dd497d <mysql3306 CONTAINER ID>
```

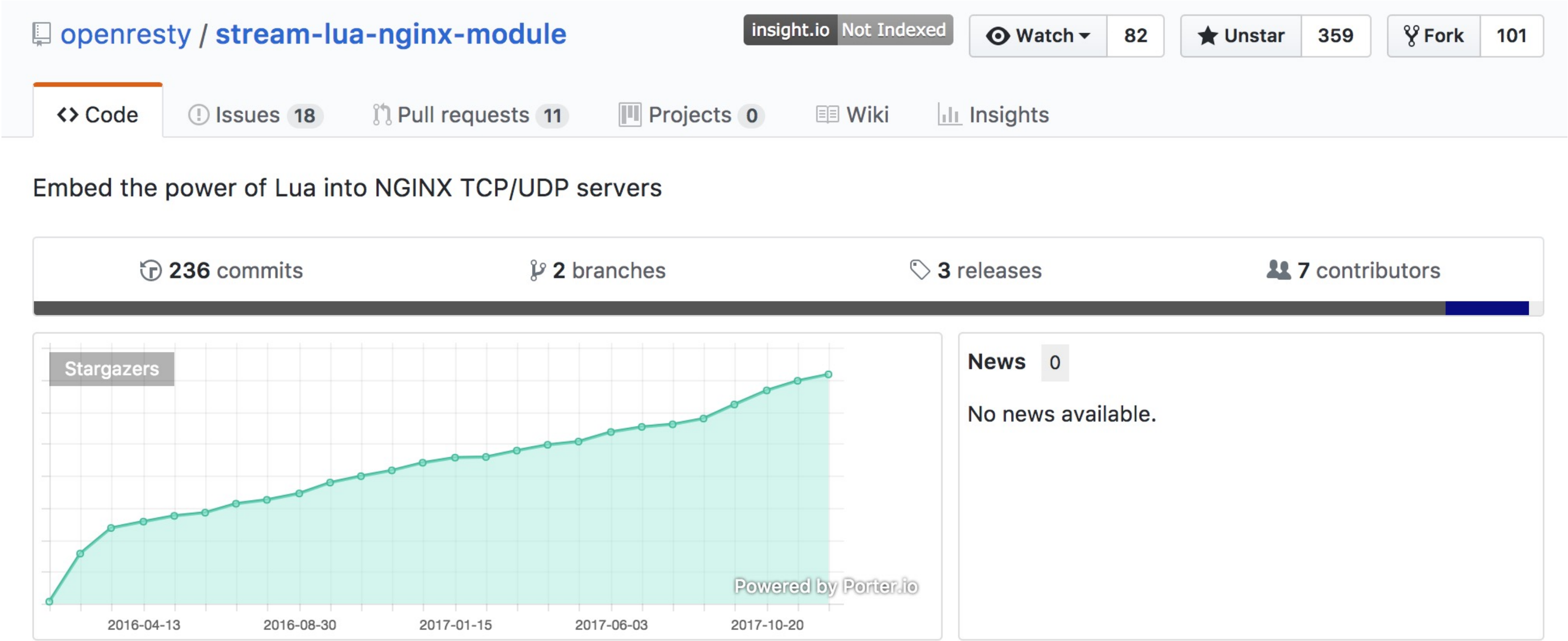
轮询:

```
$ !!;!!

hostname    396662e2585d <mysql3307 CONTAINER ID>
hostname    e3ac73dd497d <mysql3306 CONTAINER ID>
```

hash:

```
$ !!;!!

hostname    396662e2585d <mysql3307 CONTAINER ID>
hostname    396662e2585d <mysql3307 CONTAINER ID>
```

stream-lua-nginx



openresty / stream-lua-nginx-module

Embed the power of Lua into NGINX TCP/UDP servers

236 commits | 2 branches | 3 releases | 7 contributors

News 0

No news available.

Description

This is a port of the ngx_http_lua_module to the NGINX "stream" subsystem so as to support generic stream/TCP clients in the downstream.

Lua APIs and directive names rename the same as the `ngx_http_lua_module`.

Back to TOC

Hello, Lua!

和 http 类似

```
stream {
    server {
        listen 3351;

        content_by_lua_block {
            ngx.say("Hello, Lua!")
        }
    }
}
```

测试:

```
$ nc 127.0.0.1 3351
Hello, Lua!
```

TCP 负载均衡

用 Lua 来管理 upstream

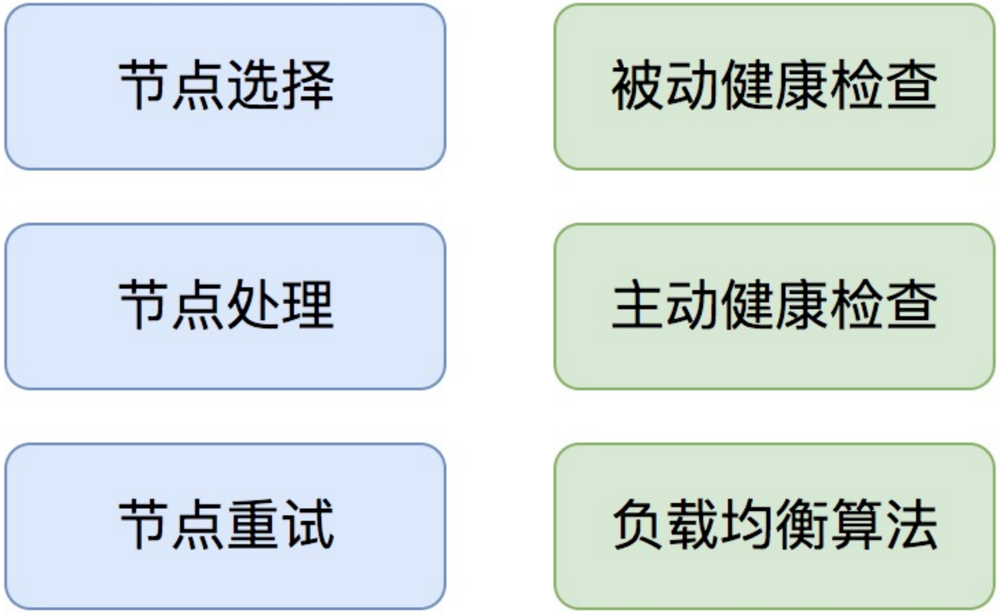
```
stream {
    upstream backend {
        server 0.0.0.1:4321; # an invalid address
        balancer_by_lua_file balance.lua;
    }
    server {
        listen 3131;
        proxy_pass backend;
    }
}
```

通过 balancer_by_lua* 和 ngx.balancer 来完成动态选择和重试 upstream

lua-resty-checkups

https://github.com/upyun/lua-resty-checkups (https://github.com/upyun/lua-resty-checkups)

实现动态 upstream 管理, 之前适用于 http 子系统， 现在也同样适用于 stream 子系统



节点选择

被动健康检查

节点处理

主动健康检查

节点重试

负载均衡算法

动态选择 upstream

在 balancer_by_lua* 中, 通过 tcp 的端口来选择相关可用的 upstream

```
skey = ngx.var.server_port

local peer, err = checkups.select_peer(skey)
if not peer then
    ngx.log(ngx.ERR, "select peer failed, ", err)
    return
end

local ok, err = balancer.set_current_peer(peer.host, peer.port)
```

- 其中，peer.host 不支持域名

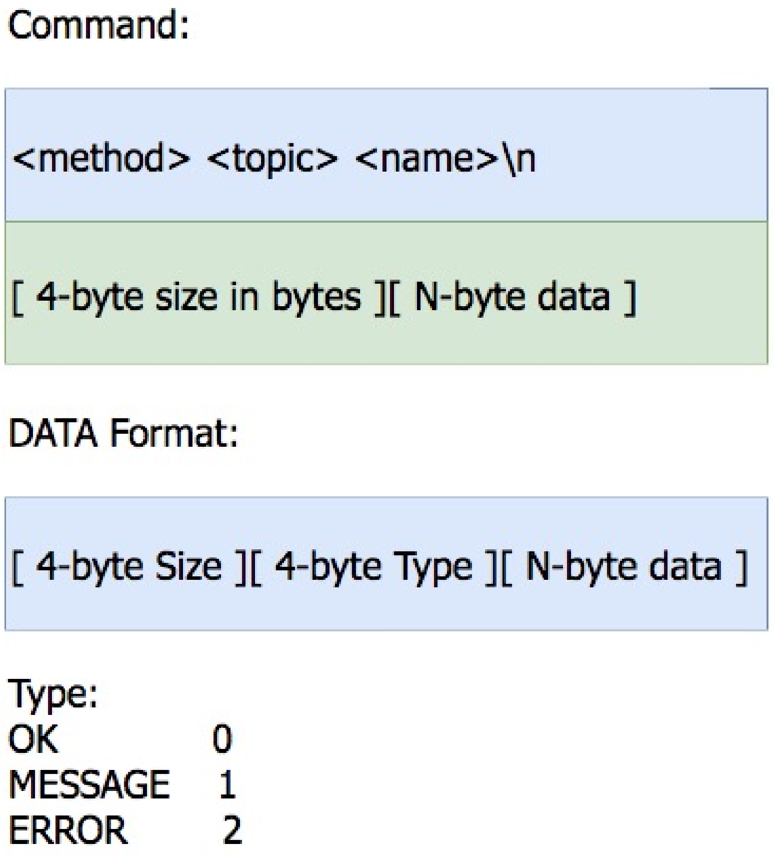
在测试服务中， `set_current_peer` 不能直接使用 { host = "mysql3306", port = 3306 }， 需要自己完成解析的操作

checkups 配置

```
_M.["3131"] = {
    -- 主动健康检查
    enable = true,
    typ = "mysql",   -- connect mysql

    -- mysql 信息
    "user":"runner",
    "pass":"runner123456",
    "name":"upyun",

    -- mysql 地址
    cluster = {
        {
            servers = {
                -- mysql3306 被动健康检查
                { host = "127.0.0.1", port = 3306, "fail_timeout":10, "max_fails":3 },
                -- mysql3307 同上
                { host = "127.0.0.1", port = 3307 },
                -- invalid
                { host = "127.0.0.1", port = 3308 },
            }
        }
    }
}
```

管理 upstream

通过 checkups 的接口, 实现了一个自定义的 tcp 协议, 完成 upstream 的增删改查



Command:

```
<method> <topic> <name>\n
[ 4-byte size in bytes ][ N-byte data ]
```

DATA Format:

```
[ 4-byte Size ][ 4-byte Type ][ N-byte data ]
```

Type:

OK	0
MESSAGE	1
ERROR	2

管理 upstream

```
$ echo -ne 'PUT upstream 3131\n\x00\0x00\0x00\0x3d{"cluster":[{"servers":[{"host":"127.0.0.1",
"port":3306}]}]}' | nc 127.0.0.1 1895 | xxd
00000000: 0000 0006 0000 0000 4f4b                 ........OK

$ echo -ne 'get upstream info\n' | nc 127.0.0.1 1895
00000000: 0000 02e5 0000 0001 ...
```

info 信息:

```
"3131": {
    "cluster":[
        { "servers":[{
            "host":"127.0.0.1",
            "port":3306,
            "weight":1,
            "fail_timeout":10,
            "max_fails":3}]
        }
    ]
}
```

健康检查

```
$ echo -ne 'get upstream status\n' | nc 127.0.0.1 1895 | xxd
00000000: 0000 025a 0000 0001 ...
```

status 信息:

```
{
    -- checkups heartbeat timer is alive.
    "checkup_timer_alive": true,

    -- last heartbeat time
    "last_check_time": "2017-12-20 15:40:58",

    -- status for 3131 cluster
    "cls:3131": [
        [{
                "server": "3131:127.0.0.1:3306",
                "msg":null,
                "status":"ok",
                "lastmodified":"2017-12-20 15:53:21",
                "fail_num":0
        }]
    ]
}
```

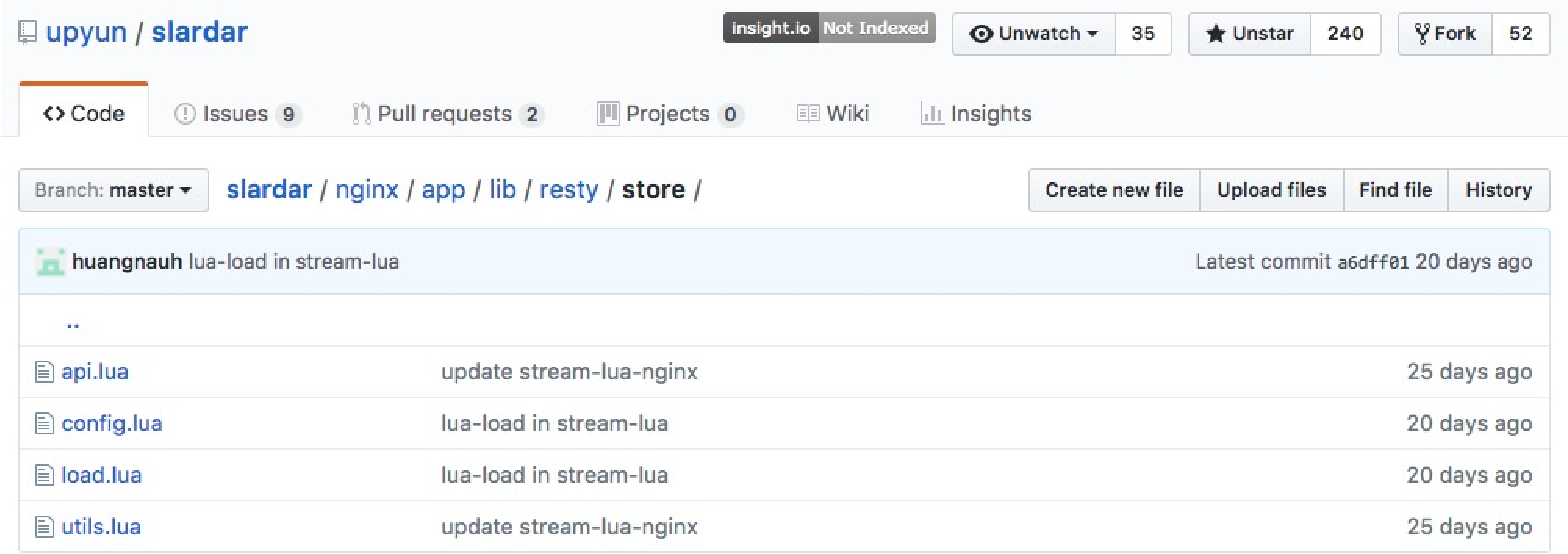
存储 upstream

通过 checkups, 我们可以

- 选择一个工作良好的 upstream
- 对 upstream 进行增删改查

我们还需要: 一个外部数据源来载入 upstream

lua-resty-store



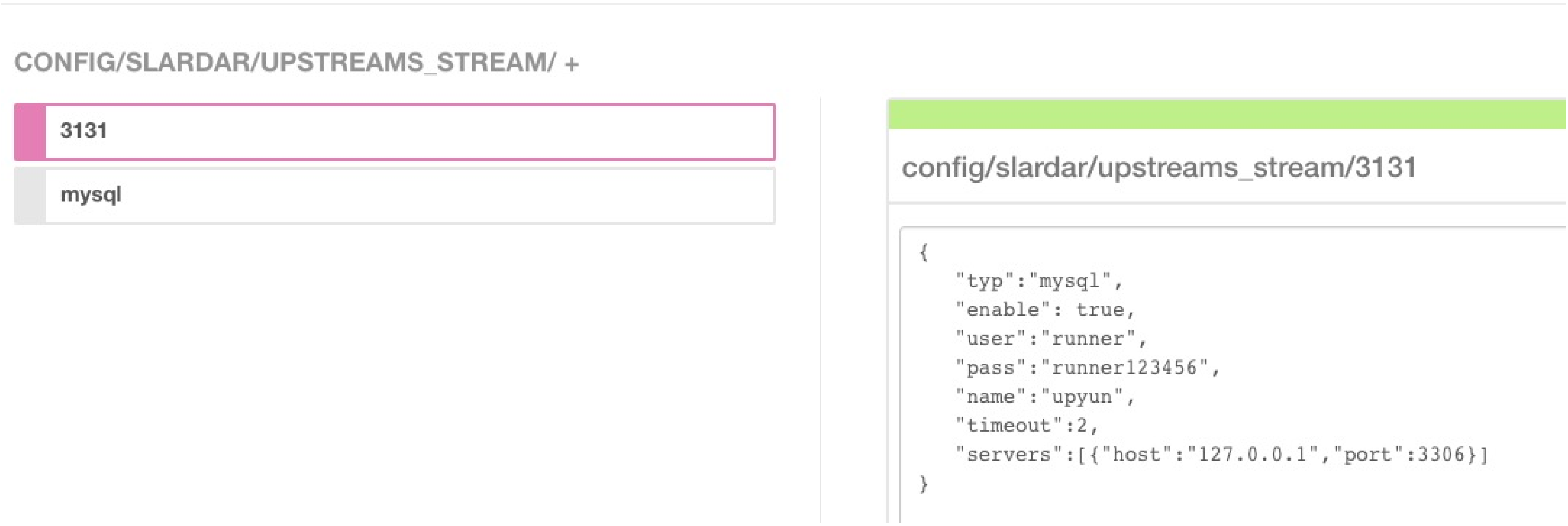
slardar / nginx / app / lib / resty / store /

huangnauh lua-load in stream-lua Latest commit a6dff01 20 days ago

..		
api.lua	update stream-lua-nginx	25 days ago
config.lua	lua-load in stream-lua	20 days ago
load.lua	lua-load in stream-lua	20 days ago
utils.lua	update stream-lua-nginx	25 days ago

- **api:** consul 和 etcd 的 api kv 接口
- **config:** 从 consul 或 etcd 动态加载配置
- **load:** 从 consul 或 etcd 动态加载 lua 源码

upstream in consul

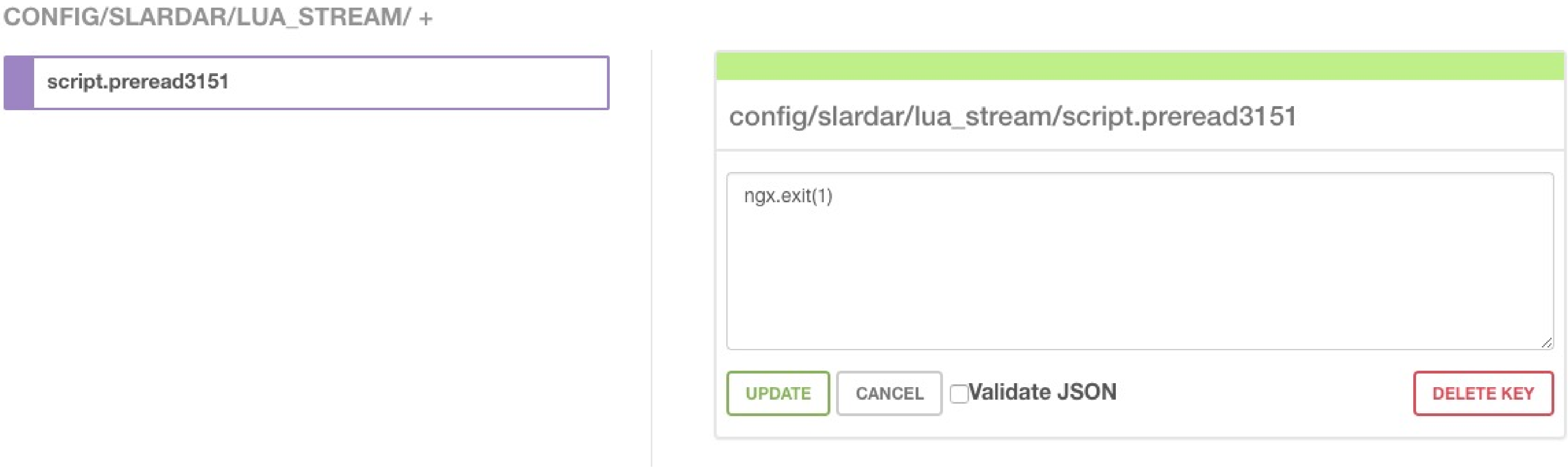


CONFIG/SLARDAR/UPSTREAMS_STREAM/ +

config/slardar/upstreams_stream/3131

```
{
    "typ":"mysql",
    "enable": true,
    "user":"runner",
    "pass":"runner123456",
    "name":"upyun",
    "timeout":2,
    "servers":[{"host":"127.0.0.1","port":3306}]
}
```

code in consul



CONFIG/SLARDAR/LUA_STREAM/ +

script.preread3151

config/slardar/lua_stream/script.preread3151

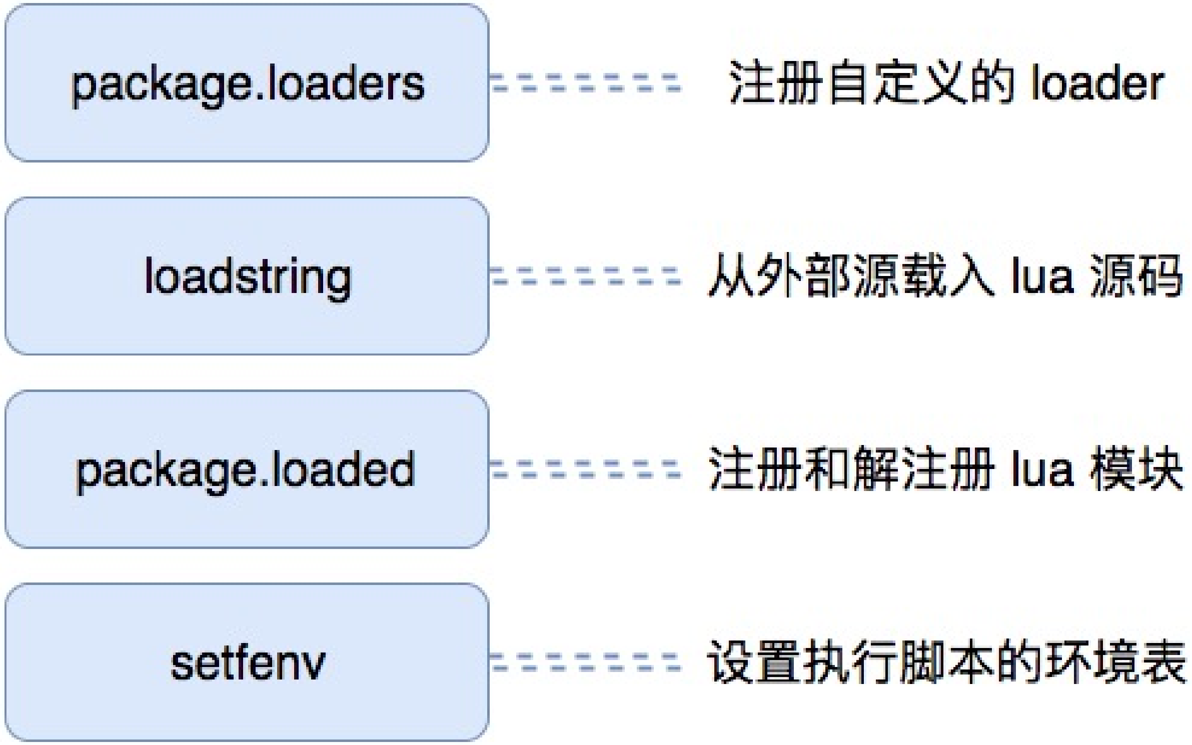
```
ngx.exit(1)
```

lua-resty-load

lua-resty-load

https://github.com/huangnauh/lua-resty-load (https://github.com/huangnauh/lua-resty-load)

从外部数据源动态加载 lua 源码, 无需 reload nginx



package.loaders — 注册自定义的 loader

loadstring — 从外部源载入 lua 源码

package.loaded — 注册和解注册 lua 模块

setfenv — 设置执行脚本的环境表

操作 lua 脚本:

操作 lua 脚本:

与操作 upstream 采用同样的 tcp 协议

```
$ echo -n 'ngx.exit(1)' | wc -c | xargs printf "0x%0.2x"
0x0b
$ echo -ne 'LOAD code script.preread3151\n\0x00\0x00\0x00\0x0bngx.exit(1)' |
nc 127.0.0.1 1895 | xxd
00000000: 0000 0006 0000 0000 4f4b                ........OK
```

测试:

```
$ mysql -h 127.0.0.1 -P 3151 -uroot -proot
ERROR 2013 (HY000): Lost connection to MySQL server
```

获取脚本信息:

```
$ echo -ne 'GET code info\n' | nc 127.0.0.1 1895
{
    "modules":[{
        "time":"2017-12-20 13:54:58",
        "version":"50e9bb007a4a0b3dbd22712f5453a5f1",
        "name":"script.preread3151"}]
}
```

应用举例

流量控制, 以漏桶算法(Leaky Bucket) resty.limit.req (https://github.com/openresty/lua-resty-limit-traffic/blob/9ac7c27212474ceb20213aea4bbf6c673a009d80/lib/resty/limit/req.md) 为例:

```
local lim = limit_req.new("stream_limit_req_store", 1, 3)
local key = ngx.var.remote_addr
local delay, err = lim:incoming(key, true)
if not delay then
    return ngx.exit(1)
end
if delay >= 0.001 then
    ngx.sleep(delay)
end
```

现在还不支持 access_by_lua, 在 preread 阶段完成限制功能

加载前:

```
    fmt.Printf("connected: %s\n", elapsed)
```

Run

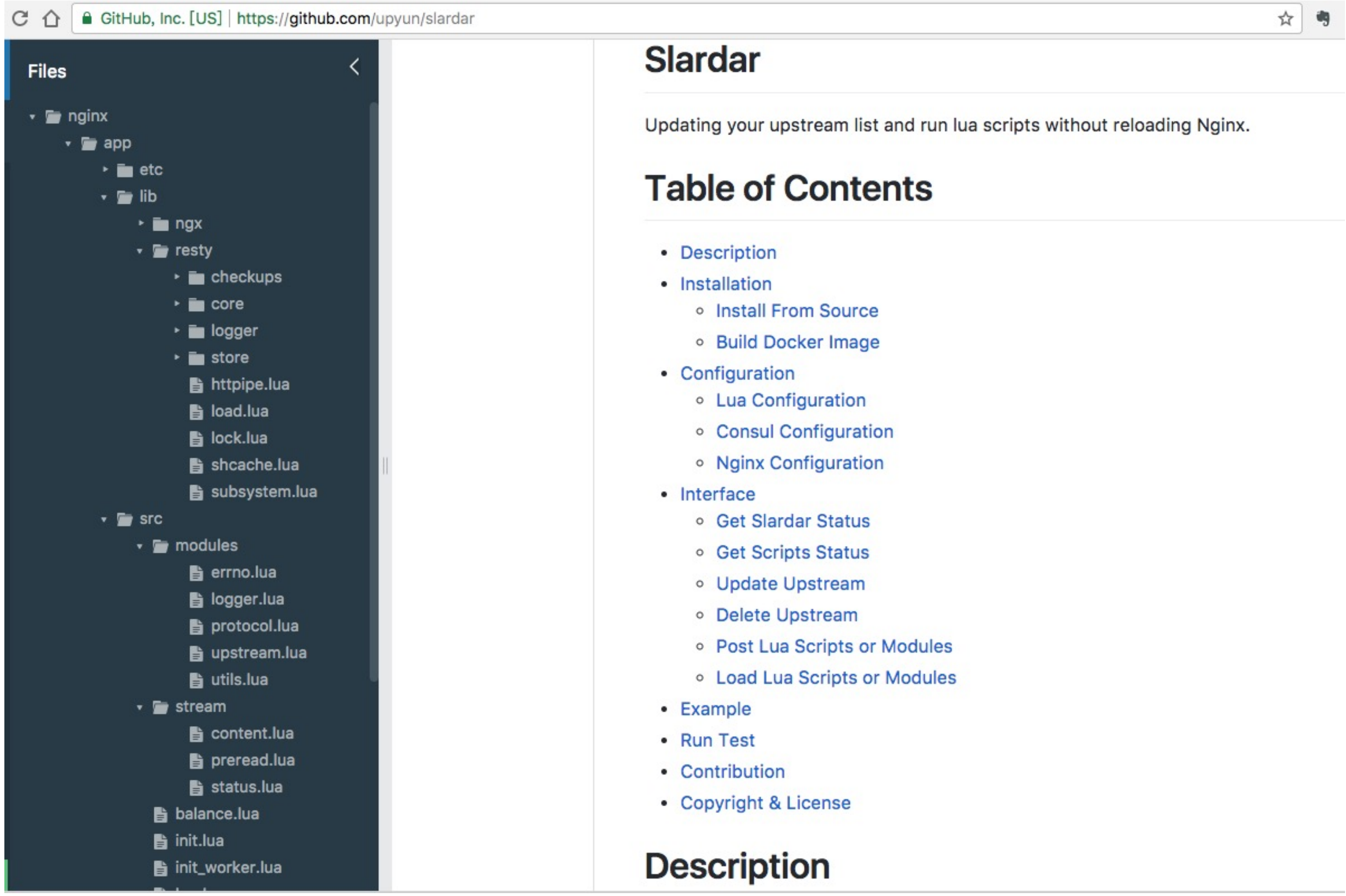
加载后:

```
    fmt.Printf("connected: %s\n", elapsed)
```

Run

Slardar

https://github.com/upyun/slardar (https://github.com/upyun/slardar)



Slardar

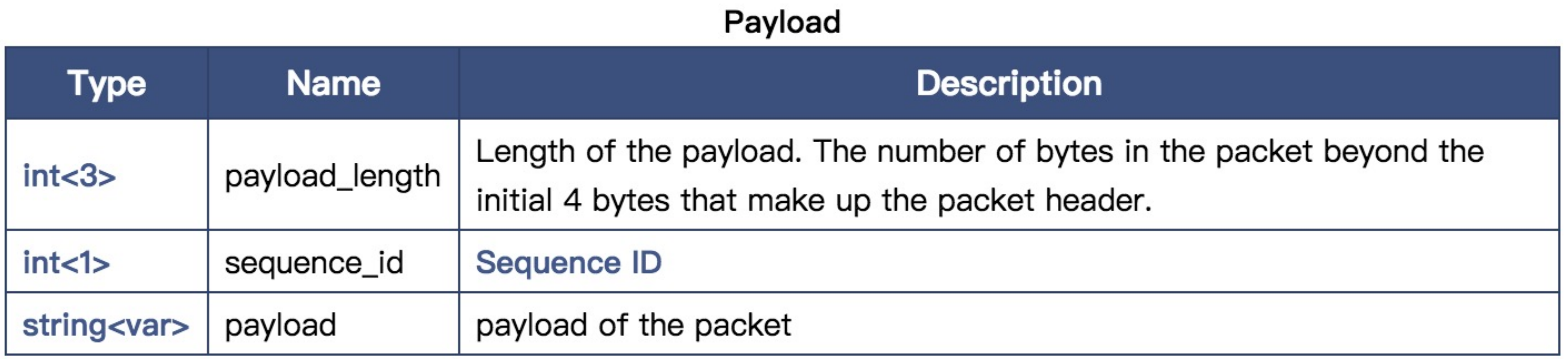
Updating your upstream list and run lua scripts without reloading Nginx.

Table of Contents

Description

MySQL Proxy

mysql packet



Payload

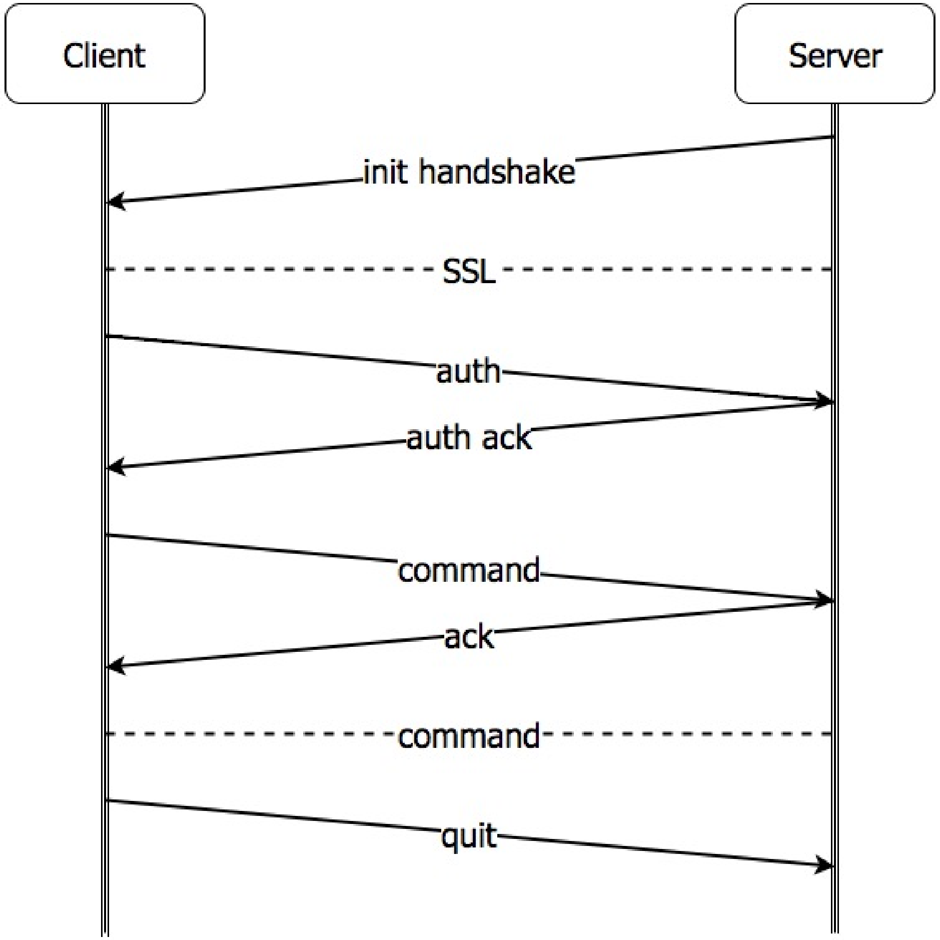
Type	Name	Description
int<3>	payload_length	Length of the payload. The number of bytes in the packet beyond the initial 4 bytes that make up the packet header.
int<1>	sequence_id	Sequence ID
string<var>	payload	payload of the packet

Image credit: mysql packets (https://dev.mysql.com/doc/dev/mysql-server/8.0.0/page_protocol_basic_packets.html/)

- fixed length integer
- length encoded integer 根据第一个 byte 转换 integer
- null terminated string
- length encoded string 根据开始的 integer 决定 string 长度 (客户端认证数据)

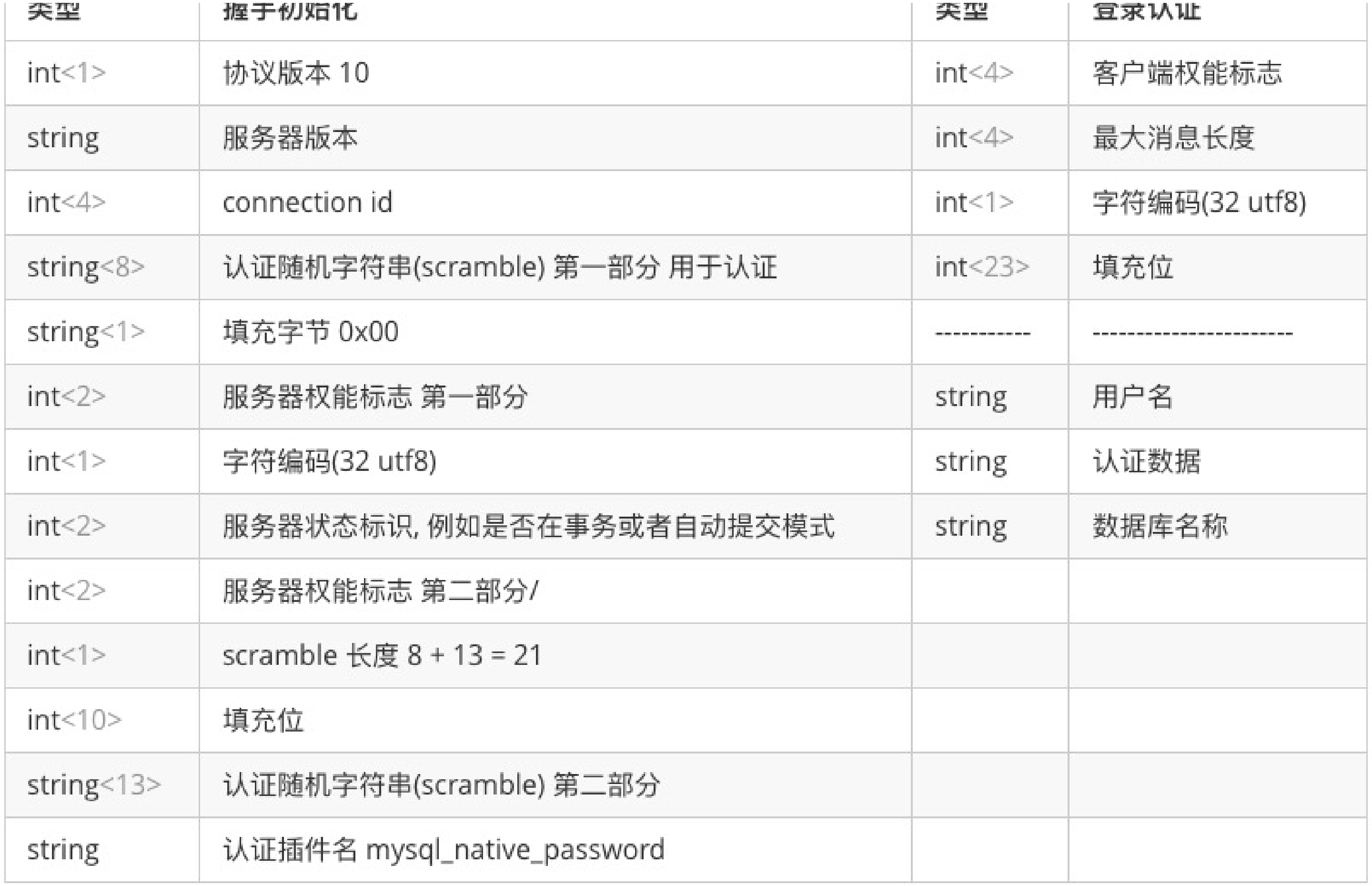
lua-resty-mysql pr69 (https://github.com/openresty/lua-resty-mysql/pull/69) 在获取字符串的时候没有把 null terminated string 的 null 去除掉

MySQL 通讯协议



MySQL 握手协议

类型	握手初始化	类型	登录认证

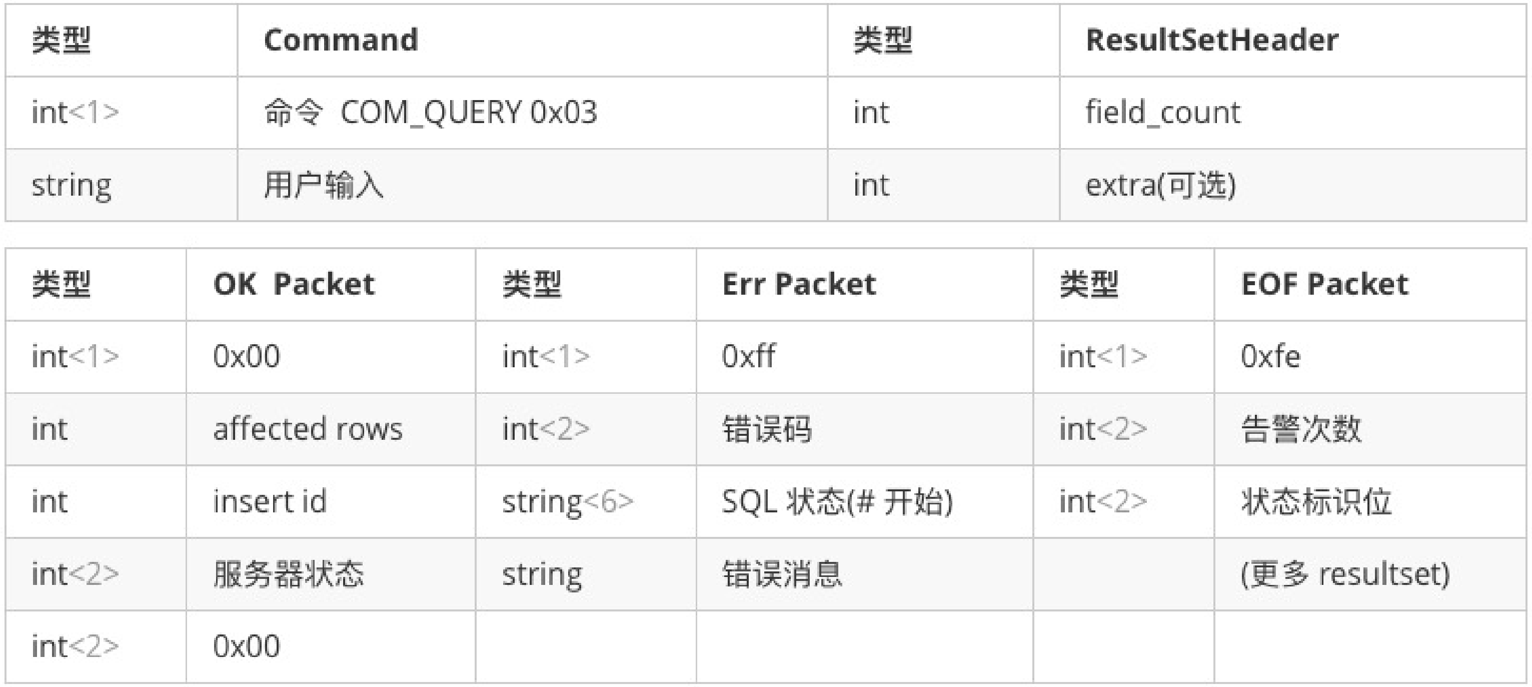


类型	握手初始化	类型	登录认证
int<1>	协议版本 10	int<4>	客户端权能标志
string	服务器版本	int<4>	最大消息长度
int<4>	connection id	int<1>	字符编码(32 utf8)
string<8>	认证随机字符串(scramble) 第一部分 用于认证	int<23>	填充位
string<1>	填充字节 0x00	-----------	-----------------------
int<2>	服务器权能标志 第一部分	string	用户名
int<1>	字符编码(32 utf8)	string	认证数据
int<2>	服务器状态标识, 例如是否在事务或者自动提交模式	string	数据库名称
int<2>	服务器权能标志 第二部分/		
int<1>	scramble 长度 8 + 13 = 21		
int<10>	填充位		
string<13>	认证随机字符串(scramble) 第二部分		
string	认证插件名 mysql_native_password		

命令消息

- COM_QUERY 包括 select，update，insert，delete 等

- COM_QUIT 客户端退出

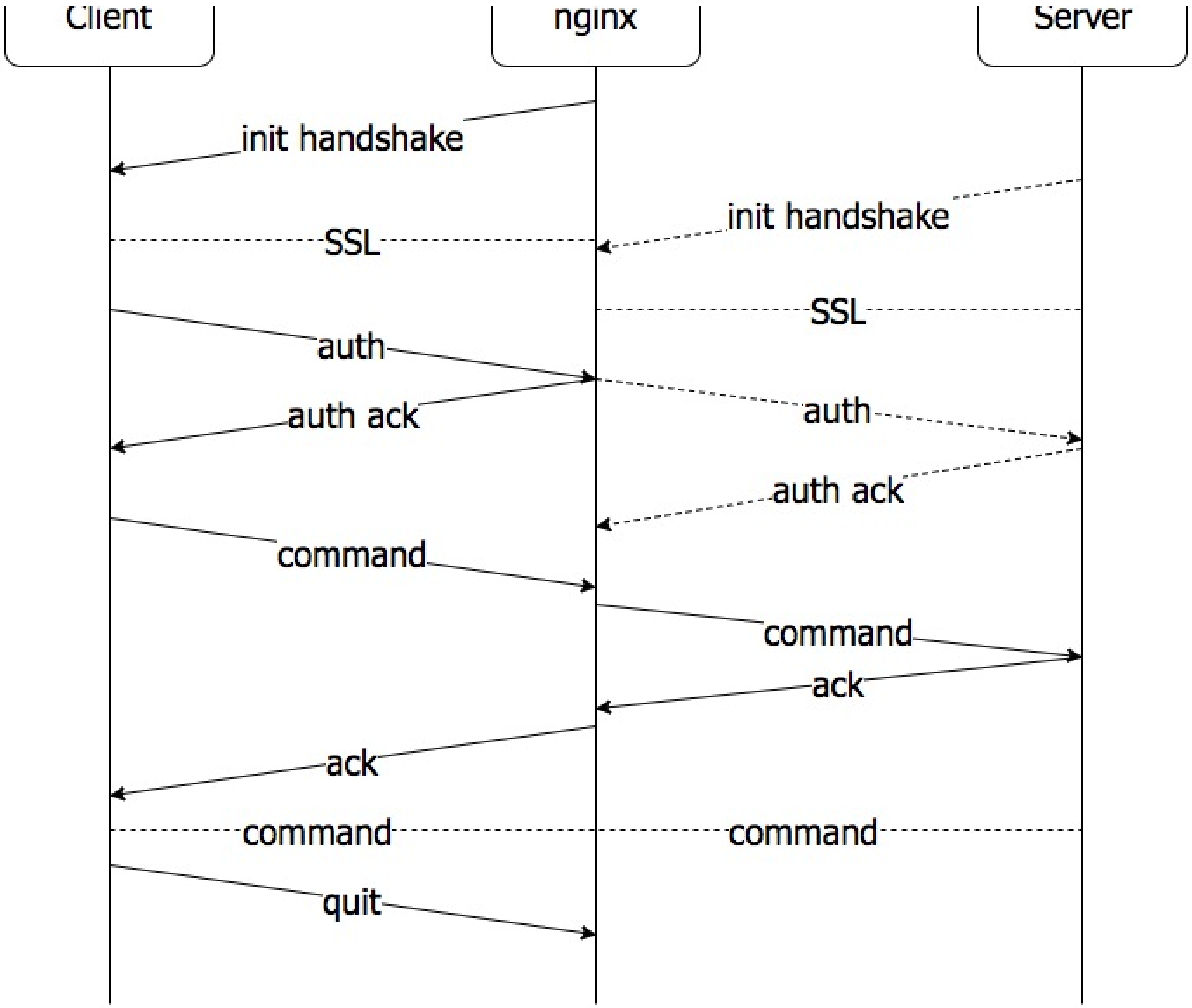


类型	Command	类型	ResultSetHeader
int<1>	命令 COM_QUERY 0x03	int	field_count
string	用户输入	int	extra(可选)

类型	OK Packet	类型	Err Packet	类型	EOF Packet
int<1>	0x00	int<1>	0xff	int<1>	0xfe
int	affected rows	int<2>	错误码	int<2>	告警次数
int	insert id	string<6>	SQL 状态(# 开始)	int<2>	状态标识位
int<2>	服务器状态	string	错误消息		(更多 resultset)
int<2>	0x00				

lua-resty-mysql pr70 (https://github.com/openresty/lua-resty-mysql/pull/70) 不能处理 field_count 大于 250 的情况

MySQL Proxy



lua-resty-mysql-toolset

https://github.com/huangnauh/lua-resty-mysql-toolset (https://github.com/huangnauh/lua-resty-mysql-toolset)

基于以上介绍的基本协议，在 lua-resty-mysql 的基础上加入了 server 的协议部分，包括一个测

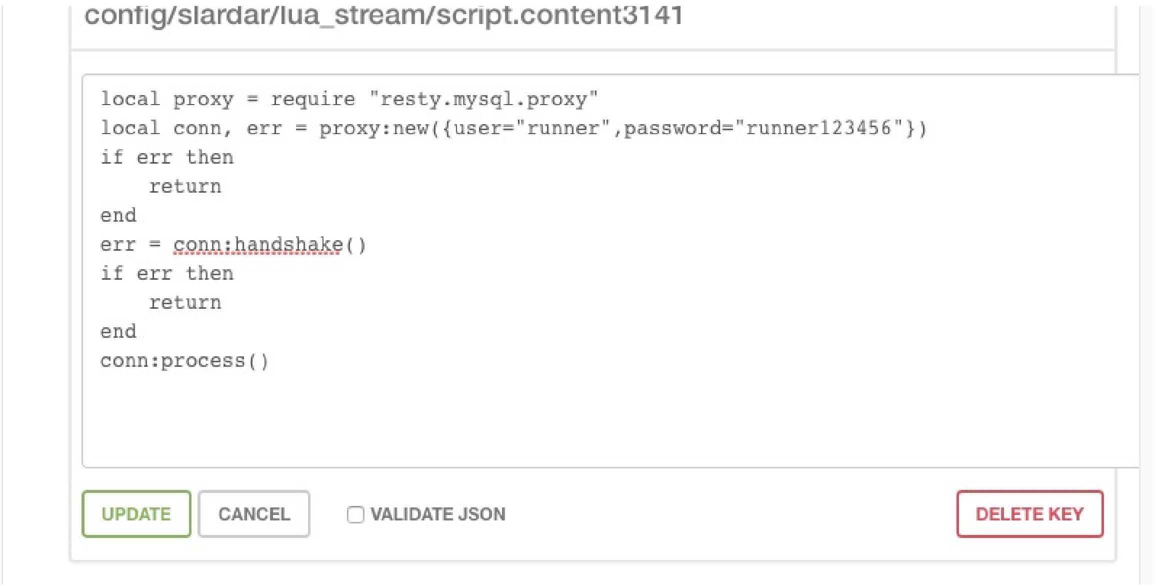
基于以上介绍的基本协议，在 lua-resty-mysql 的基础上加入了 server 的协议部分，包括一个测试用的 proxy

```
stream {
    server {
        listen 3141;
        preread_by_lua_file app/src/stream/preread.lua;
        content_by_lua_file app/src/stream/content.lua;
        log_by_lua_file app/src/log.lua;
   }
}
```

lua in consul

CONFIG/SLARDAR/LUA_STREAM/ +

script.content3141



```
local proxy = require "resty.mysql.proxy"
local conn, err = proxy:new({user="runner",password="runner123456"})
if err then
    return
end
err = conn:handshake()
if err then
    return
end
conn:process()
```

测试:

```
$echo "show variables where variable_name = 'hostname'" |
pipe>  mysql --skip-column-names -h 127.0.0.1 -P 3141 -urunner -prunner123456  upyun

hostname    huangnauh.local
```

读写分离

1. 通过不同端口来区分读写 upstream, 由应用程序来区分读写

```
stream {
    upstream backend {
```

```
    upstream backend {
        server 0.0.0.1:4321; # an invalid address
        balancer_by_lua_file balance.lua;
    }
    server {
        listen 3132;
        proxy_pass backend;
    }
    server {
        listen 3133;
        proxy_pass backend;
    }
}
```

读写分离

2. 分析 COM_QUERY sql 语句

```
local cmd = string.match(sql, "([^%s,/]+)")
if not cmd then
```

```
    return nil, "sql error"
end
cmd = string.lower(cmd)
-- 简单 DML 语句区分读写，不考虑带注释的情况
if cmd == "select" or cmd == "show" then
    ...
else
    ...
end
```

性能比较

```
sysbench --time=10 --threads=100
```

MySQL Proxy:

```
SQL statistics:
    queries performed:
        read:                            48253
        other:                           96506
        total:                           144759
    queries:                             144759 (14396.37 per sec.)
```

MySQL:

```
SQL statistics:
    queries performed:
        read:                            65328
        other:                           130656
        total:                           195984
    queries:                             195984 (19525.11 per sec.)
```

Thank you

黄励博(huangnauh)
又拍云

ihuangnauh@gmail.com (mailto:ihuangnauh@gmail.com)
https://github.com/huangnauh (https://github.com/huangnauh)

https://github.com/huanghuan (https://github.com/huanghuan)