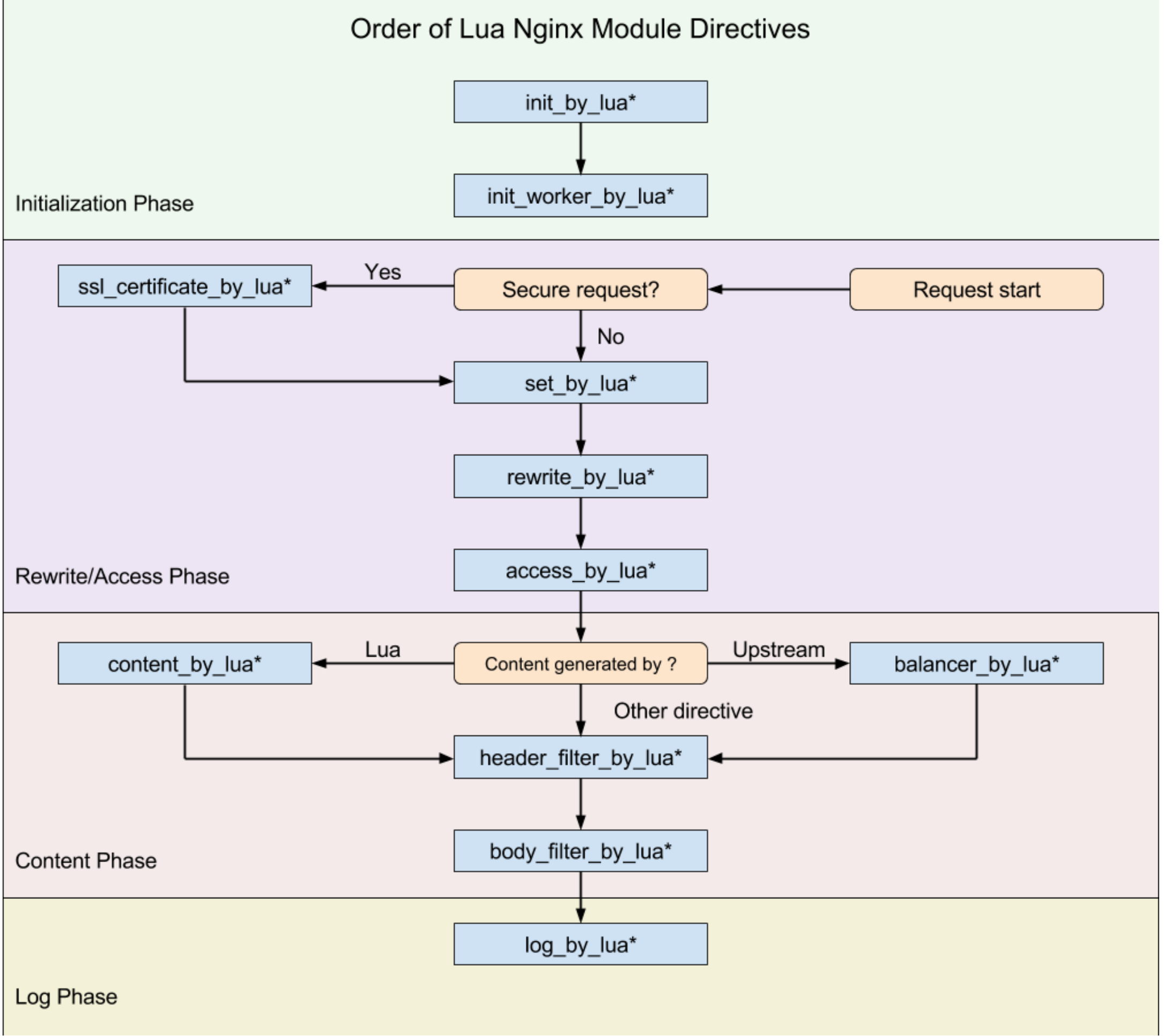
OR一些特性的介绍

by 河马大侠


Order of Lua Nginx Module Directives
Initialization Phase
init_by_lua*
init_worker_by_lua*
Request start
Secure request?
Yes
ssl_certificate_by_lua*
No
set_by_lua*
rewrite_by_lua*
access_by_lua*
Rewrite/Access Phase
Content generated by ?
Lua
content_by_lua*
Upstream
balancer_by_lua*
Other directive
header_filter_by_lua*
body_filter_by_lua*
Content Phase
log_by_lua*
Log Phase

基础的Http处理

- 请求头获取 ngx.say | ngx.print
- 返回头修改 ngx.resp.get_headers
- 内容控制 body_filter_by_*
- 状态码控制 ngx.exit(), ngx.status

- 控制error的输出内容

```
location =/error {
    content_by_lua_block {
        return ngx.exit(500)
    }
}

GET /error
======
<html>
<head><title>500 Internal Server Error</title></head>
<body bgcolor="white">
<center><h1>500 Internal Server Error</h1></center>
<hr><center>openresty</center>
</body>
</html>
```

```
location =/error {
    content_by_lua_block {
        ngx.status = 500
        ngx.print("your awesome error page!")
        return ngx.exit(200)
    }
}

GET /erro
======
your awesome error page!
```

- 访问 redis

```
location = /access_to_redis {
    content_by_lua_block {
        local redis = require "resty.redis"
        local red = redis:new()
        local res, err = red:hget("routes", "some_field")
        if not err then
            ngx.say(res)
        end
    }
}
```

- 访问 postgres

```
location = /access_to_postgres {
    content_by_lua_block {
        local pgmoon = require "resty.pgmoon"
        local pg = pgmoon.new({
          host = singletons.postgres.ip,
          port = singletons.postgres.port,
          database = singletons.postgres.database,
          user = singletons.postgres.user,
          socket_type = "nginx"
        })
        pg:settimeout(1000)
        local ok, err = pg:connect()
        local id = 1   -- or get id from url args
        local res = assert(pg:query(string.format(
            "select *  from table where id = %s limit 1" ,
            pg:escape_literal(id))))
        ngx.say(res)
    }
}
```

body_filter_by_lua*

- 可能在一次请求中调用多次，跟响应数据量无关，取决于响应次数
- 最后一次调用时，ngx.arg[1] 一般为空字符串
- 会在 subrequest 之中调用　　if ngx.arg[1] and not ngx.is_subrequest then
- 有些时候离不开有 header_filter_by_lua* 辅佐

```
header_filter_by_lua_block {
    ngx.header.content_length = nil
}

body_filter_by_lua_...
```

ngx_http_定时器.at

```lua
local delay = 5
local handler
handler = function (premature)
    -- do some routine job in Lua just like a cron job
    if premature then
        return
    end
    local ok, err = ngx.timer.at(delay, handler)  -- <<=== 注意这里
    if not ok then
        ngx.log(ngx.ERR, "failed to create the timer: ", err)
        return
    end
end

local ok, err = ngx.timer.at(delay, handler)
if not ok then
    ngx.log(ngx.ERR, "failed to create the timer: ", err)
    return
end
```

ngx.worker.id()

跨作用域

```
init_worker_by_lua_block {
    local handler = function()
        local redis = require "resty.redis"
        redc = redis:new(anyaddr)
        redc:set('foo', 'bar')
    end

    ngx.timer.at(0, handler)
}
```

init 阶段怎么解?

tips: lua 同样提供了 socket，值得一试

```
return {
  new = function(socket_type)
    if socket_type == nil then
      if ngx and ngx.get_phase() ~= "init" then
        socket_type = "nginx"
      else
        socket_type = "luasocket"
      end
    end
    local socket
    local _exp_0 = socket_type
    if "nginx" == _exp_0 then
      socket = ngx.socket.tcp()
    elseif "luasocket" == _exp_0 then
      socket = luasocket.tcp()
    elseif "cqueues" == _exp_0 then
      socket = require("pgmoon.cqueues").CqueuesSocket()
    else
      socket = error("unknown socket type: " .. tostring(socket_type))
    end
    return socket, socket_type
  end
}
```

跨进程共享

- SD+轮询
- 外援 （Redis + Memcached） 注意空table的转换和ngx.null的使用
- ngx_lua_ipc
- 动态监听unix socket
- 每个 worker 分配不同的 unix socket

  listen unix:socket flag_a

Https

```
http
{
    server {
        listen       443 ssl;
        server_name  www.foo.com;

        ssl_certificate      cert/foo.pem;
        ssl_certificate_key  cert/foo.key;

        location / {
            proxy_pass http://foo;
        }
    }

    server {
        listen       443 ssl;
        server_name  www.bar.com;

        ssl_certificate      cert/bar.pem;
        ssl_certificate_key  cert/bar.key;


        location / {
            proxy_pass http://bar;
        }
    }
}
```

ssl_certificate_by_lua*

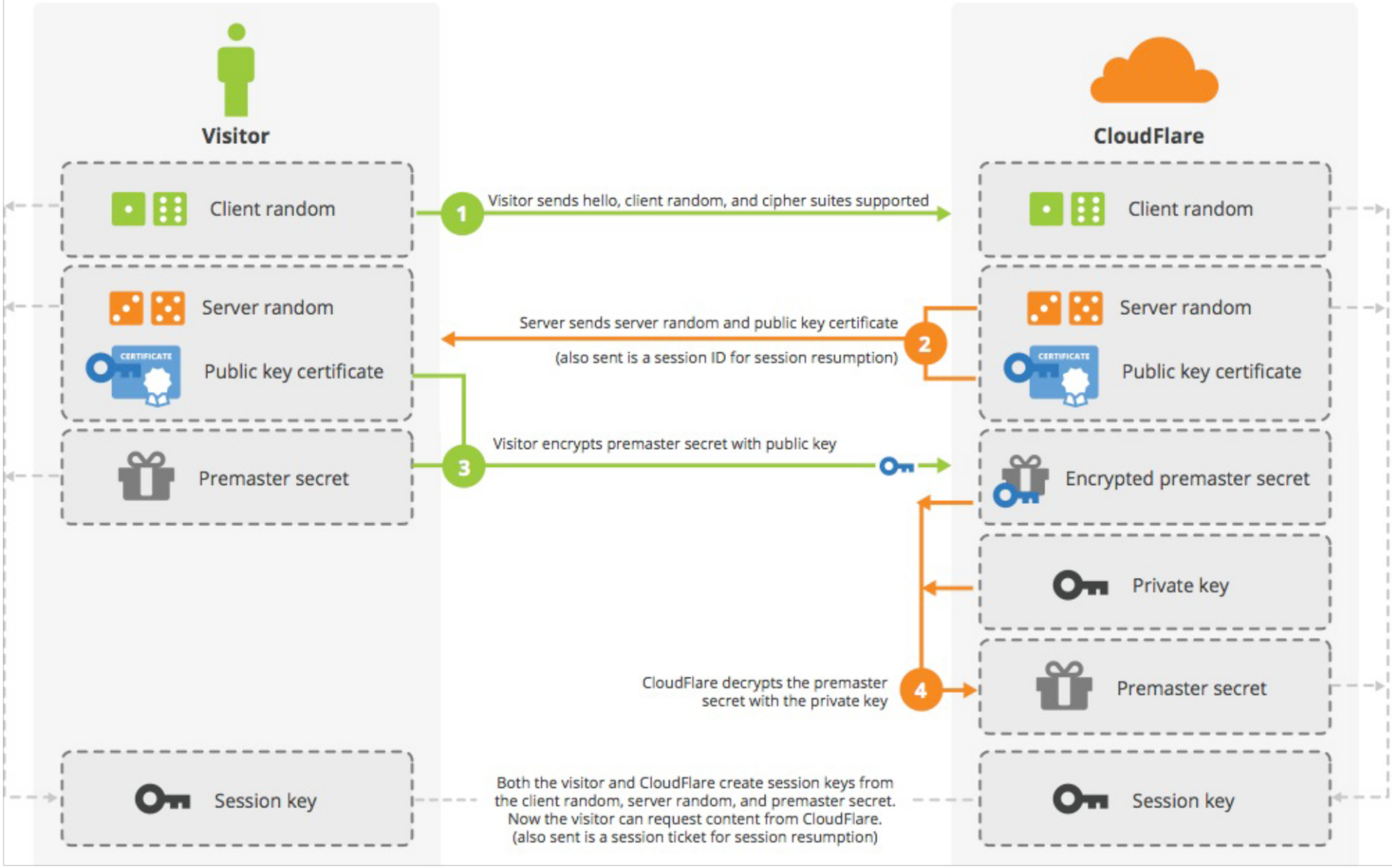
function:

ngx.ssl.parse_pem_cert

ngx.ssl.parse_pem_priv_key

```
server {
    listen 443 ssl;
    server_name   test.com;

    # 虽然我们可以动态的加载证书，但是为了避免nginx报错，这里需要配置用于占位的证书和私钥
    ssl_certificate placeholder.crt;
    ssl_certificate_key placeholder.key;

    ssl_certificate_by_lua_block {
        local ssl = require "ngx.ssl"

        -- 清理掉当前的证书，为后续加载证书做准备
        local ok, err = ssl.clear_certs()

        if not ok then
            ngx.log(ngx.ERR, "clear current cert err:", err)
            return ngx.exit(500)
        end

        -- x509 格式的证书有两种存储格式 PEM 和 DER，这里只描述PEM格式

        -- 获取 Server Name Indication 简称 (SNI)
        local sni = ssl.server_name()

        -- 这里我们假设已经通过cosocket实现了一个从数据库获取证书的函数
        -- 该函数以sni为索引查询对应的证书并返回
        local cert, err = get_pem_format_cert_by_server_name(sni)

        -- cert_of_x509 是一个cdata类型的数据
        local cert_of_x509, err = ssl.parse_pem_cert(cert)

        local ok, err = ssl.set_cert(cert_of_x509)

        if not ok then
            ngx.log(ngx.ERR, "set cert failed, err:", err)
            return ngx.exit(500)
        end

        --- 这里还需要设置对应的私钥，相关函数请参考如下

    }
}
```

https 性能问题


Visitor
CloudFlare
Client random
1
Visitor sends hello, client random, and cipher suites supported
Client random
Server random
Server sends server random and public key certificate
(also sent is a session ID for session resumption)
2
Server random
CERTIFICATE
Public key certificate
CERTIFICATE
Public key certificate
Premaster secret
3
Visitor encrypts premaster secret with public key
Encrypted premaster secret
Private key
CloudFlare decrypts the premaster secret with the private key
4
Premaster secret
Session key
Both the visitor and CloudFlare create session keys from the client random, server random, and premaster secret. Now the visitor can request content from CloudFlare. (also sent is a session ticket for session resumption)
Session key

- session cookie
- session ticket

ssl_session_fetch_by_xxx

ssl_session_store_by_xxx

```
ssl_session_store_by_lua_block {
    local ssl_sess = require "ngx.ssl.session"

    local sess_id, err = ssl_sess.get_session_id()
    if not sess_id then
        ngx.log(ngx.ERR, "failed to get session ID: ", err)
        -- just give up
        return
    end


    local sess, err = ssl_sess.get_serialized_session()
    if not sess then
        ngx.log(ngx.ERR, "failed to get SSL session from the ",
                "current connection: ", err)
        -- just give up
        return
    end

    -- 由于在这个阶段不能使用cosocket，所以我们只能通过前文所诉的timer.at的嫁接办法来实现存储
    local function save_it(premature, sess_id, sess)
        local sess, err = my_save_ssl_session_by_id(sess_id, sess)
        if not sess then
            if err then
                ngx.log(ngx.ERR, "failed to save the session by ID ",
                        sess_id, ": ", err)
                return ngx.exit(ngx.ERROR)
            end

            return
        end
    end

    -- timer.at 嫁接
    local ok, err = ngx.timer.at(0, save_it, sess_id, sess)
    if not ok then
        ngx.log(ngx.ERR, "failed to create a 0-delay timer: ", err)
        return
    end
}
```

```
ssl_session_fetch_by_lua_block {
    local ssl_sess = require "ngx.ssl.session"

    local sess_id, err = ssl_sess.get_session_id()
    if not sess_id then
        ngx.log(ngx.ERR, "failed to get session ID: ", err)
        return
    end

    -- my_lookup_ssl_session_by_id() 函数是自定义的session 获取函数,
    -- 在这个 ssl_session_fetch_by 阶段, 是用可以使用 cosocket 的, 所以我们可以在这个函数里
    local sess, err = my_lookup_ssl_session_by_id(sess_id)
    if not sess then
        if err then
            ngx.log(ngx.ERR, "failed to look up the session by ID ",
                    sess_id, ": ", err)
            return
        end

        -- 没有找到缓存, 直接返回, 走协商逻辑, 做 RSA 加解密运算
        return
    end

    -- 找到缓存, 复用 primary key, 节约cpu开销
    local ok, err = ssl_sess.set_serialized_session(sess)
    if not ok then
        ngx.log(ngx.ERR, "failed to set SSL session for ID ", sess_id,
                ": ", err)
        -- consider it as a cache miss...
        return
    end

}
```

```
bogon:Desktop [illegible]$ wrk -t8 -c500 -d5s --script=test.lua --latency http://[illegible]
Running 5s test @ http://[illegible]
  8 threads and 500 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    54.16ms   10.62ms 108.59ms   92.08%
    Req/Sec   334.73    255.12   650.00     33.33%
  Latency Distribution
     50%   51.65ms
     75%   55.24ms
     90%   62.03ms
     99%  103.09ms
  4999 requests in 5.10s, 5.75MB read
Requests/sec:    979.86
Transfer/sec:      1.13MB
```

```
bogon:Desktop [illegible]$ wrk -t8 -c500 -d30s --script=test.lua --latency https://[illegible]/with_ssl
Running 30s test @ https://[illegible]/with_ssl
  8 threads and 500 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    50.41ms    6.68ms 153.61ms   87.57%
    Req/Sec   139.83    143.29   620.00     83.44%
  Latency Distribution
     50%   48.29ms
     75%   53.13ms
     90%   58.15ms
     99%   73.41ms
  28031 requests in 30.10s, 31.81MB read
Requests/sec:    931.31
Transfer/sec:      1.06MB
bogon:Desktop [illegible]$
```

```
msgtc.exe      [illegible]   00      3,036 K  C:\Windows\System32\msgtc.exe                                  Microso...
nginx.exe      SYSTEM        00      2,032 K  "C:\Program Files (x86)\360\skylar6\nginx\nginx.exe"            nginx
nginx.exe      SYSTEM        07      6,776 K  "C:\Program Files (x86)\360\skylar6\nginx\nginx.exe"            nginx
notepad++      Admin         00    255,752 K  "C:\Program Files (x86)\Notepad++\notepad++.exe"               Notepad
```

- ffi
- resty core
- test nginx
- 框架
- 周边

Q&A

谢谢