零声教育 Mark 老师 QQ：2548898954

openresty 简介

- *openresty* 是一个**基于 *nginx* 与 *lua*** 的高性能 *web* 平台，其内部集成了大量精良的 *lua* 库、第三方模块以及大数的依赖项。用于方便搭建能够处理超高并发、**扩展性**极高的**动态 web 应用**、*web* 服务和动态网关。
- *openresty* 通过**汇聚**各种设计精良的 **nginx 模块**，从而将 nginx 有效地变成一个强大的通用 *Web* 应用平台。这样，*Web* 开发人员和系统工程师可以使用 *Lua* 脚本语言调动 Nginx 支持的各种 *C* 以及 *Lua* 模块，快速构造出足以胜任 *10K* 乃至 *1000K* 以上单机并发连接的高性能 *Web* 应用系统。
- *openresty* 的目标是让你的 *Web* 服务直接跑在 *Nginx* 服务内部，**充分利用 *Nginx* 的非阻塞 I/O 模型**（多*reactor* 模型），不仅仅对 *HTTP* 客户端请求（*stream*），甚至于对远程后端诸如 *MySQL*、*PostgreSQL*、*Memcached* 以及 *Redis etcd kafka grpc* 等都进行一致的高性能响应（*upstream*）。

openresty 安装

> 官网： http://openresty.org/cn/
>
> 下载页面：http://openresty.org/cn/download.html
>
> - openresty-1.21.4.1.tar.gz 5.0MB PGP 变更列表 - 2022年5月18日

```
# 安装依赖
apt-get install libpcre3-dev \
    libssl-dev perl make build-essential curl
# 解压源码
tar -xzvf openresty-VERSION.tar.gz

# 配置：默认，--prefix=/usr/local/openresty 程序会被安装到/usr/local/openresty目录。
./configure
make -j2
sudo make install

cd ~
export PATH=/usr/local/openresty/bin:$PATH
```

启动、关闭、重启 openresty

```
# 指定配置启动 openresty
openresty -p . -c conf/nginx.conf
# 优雅退出
openresty -p . -s quit
# 重启 openresty
openresty -p . -s reload
```

openresty 应用场景

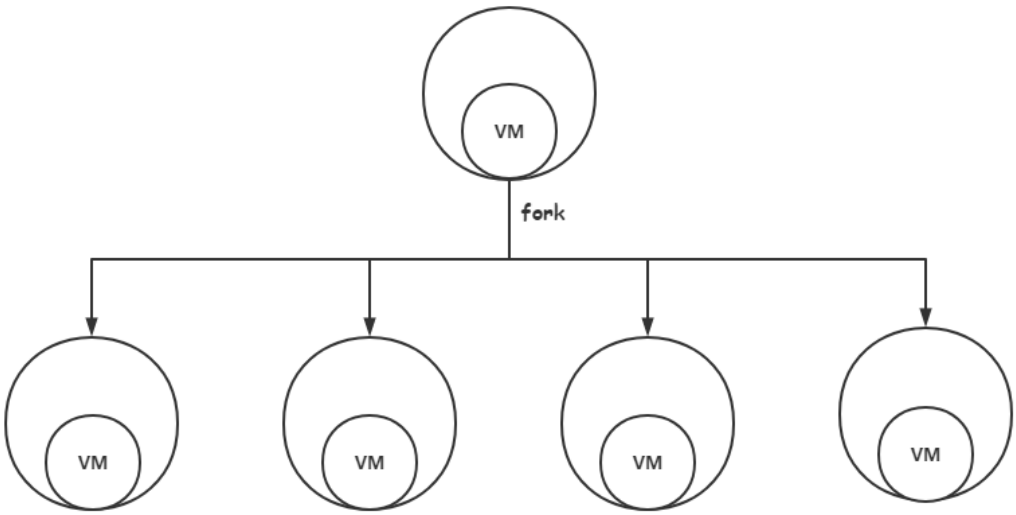
> 奇虎360的所有服务端团队都在使用，京东、百度、魅族、知乎、优酷、新浪这些互联网公司都在使用。有用来写 WAF（web application firewall）、有做 CDN 调度、有做广告系统、消息推送系统，API server 的。还有用在非常关键的业务上，比如高可用架构分享的京东商品详情页。

- 在请求真正到达上游服务之前，*Lua* 可以随心所欲的做复杂的访问控制和安全检测
- 随心所欲的操控响应头里面的信息

- 从外部存储服务（比如 *Redis*，*Memcached*，*MySQL*，*Postgres*）中获取后端信息，并用这些信息来实时选择哪一个后端来完成业务访问
- 在内容 *handler* 中随意编写复杂的 *Web* 应用，使用同步但依然非阻塞的方式，访问后端数据库和其他存储
- 在 *rewrite* 阶段，通过 *Lua* 完成非常复杂的 URL *dispatch*
- 用 *Lua* 可以为 *nginx* 子请求和任意 *location*，实现高级缓存机制

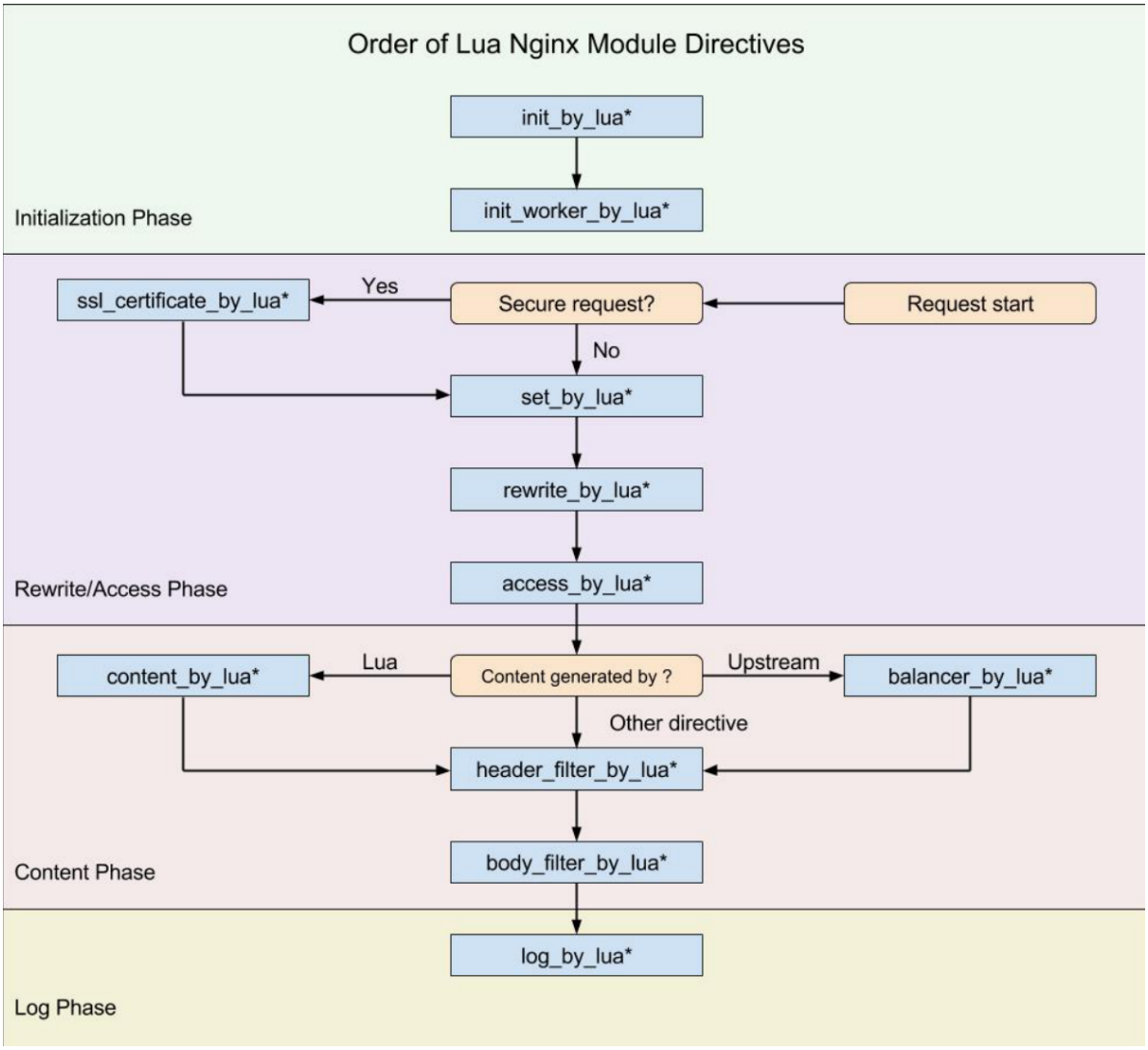
lua-nginx-module

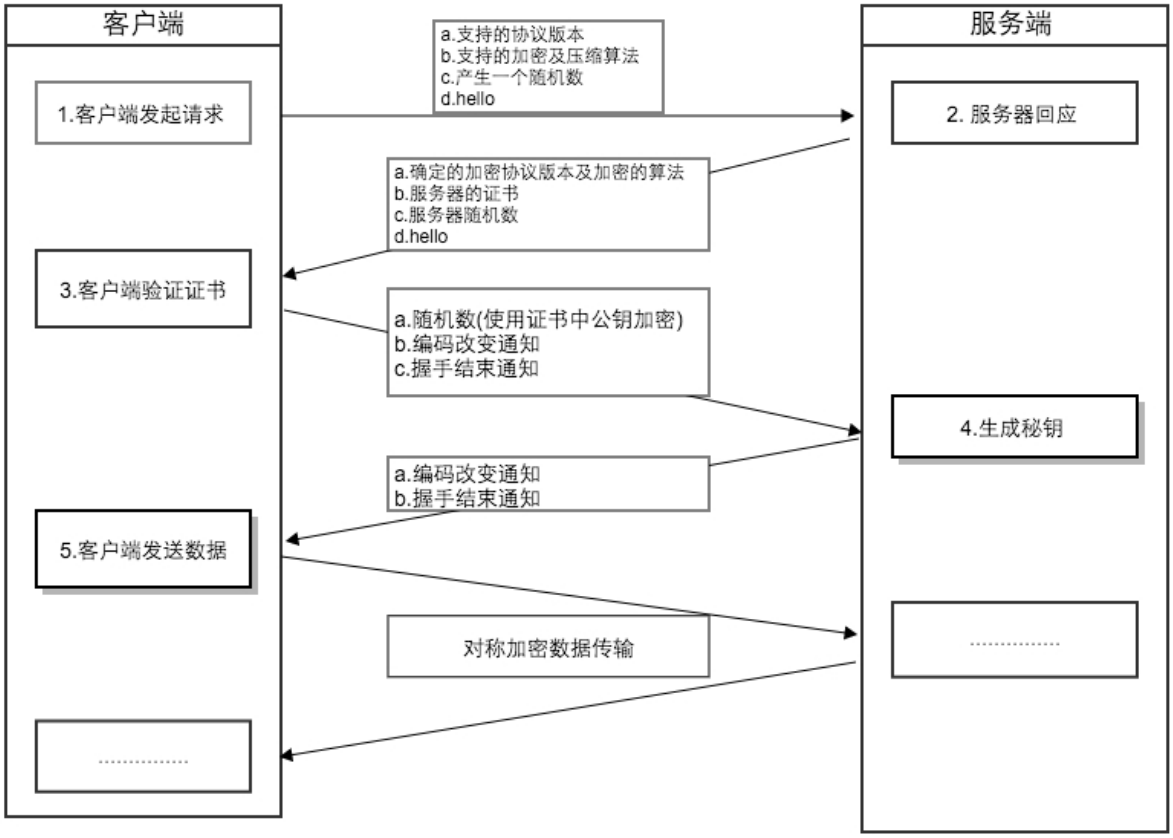
> *nginx* 采用模块化设计，使得每一个 *http* 模块可以仅专注于完成一个独立的、简单的功能，而一个请求的完整处理过程可以由无数个 *http* 模块共同合作完成。为了灵活有效地指定下一个 *http* 处理模块是哪一个；http 框架依据常见的的处理流程将处理阶段划分为 11 个阶段，其中每一个阶段都可以由任意多个 *http* 模块流水式地处理请求。
>
> *openresty* 将 *lua* 脚本嵌入到 *nginx* 阶段处理的末尾模块下；这样以来并不会影响 *nginx* 原有的功能，而是在 *nginx* 基础上丰富它的功能；
>
> 嵌入 lua 的优点是：使用 openresty 开发，不需要重新编译，直接修改 lua 脚本，重新启动即可；



lua 模块指令顺序

> 问题：访问某个页面，先验证是否用户权限是否合法，否则跳到用户验证界面；
>
> 问题：黑白名单在哪个阶段实现？




客户端
服务端
a.支持的协议版本
b.支持的加密及压缩算法
c.产生一个随机数
d.hello
1.客户端发起请求
2. 服务器回应
a.确定的加密协议版本及加密的算法
b.服务器的证书
c.服务器随机数
d.hello
3.客户端验证证书
a.随机数(使用证书中公钥加密)
b.编码改变通知
c.握手结束通知
4.生成秘钥
a.编码改变通知
b.握手结束通知
5.客户端发送数据
对称加密数据传输
.....
.....

函数名	lua-nginx-module context	描述
init_worker()	init_worker_by_lua	在每个 Nginx 工作进程启动时执行
certificate()	ssl_certificate_by_lua	ssl 阶段，在“握手”时设置安全证书
rewrite()	rewrite_by_lua	从客户端接收作为重写阶段处理程序的每个请求执行。在这个阶段，无论是API还是消费者都没有被识别，因此这个处理器只在插件被配置为全局插件时执行
access()	access_by_lua	为客户的每一个请求而执行，并在它被代理到上游服务之前执行（路由）
header_filter()	header_filter_by_lua	从上游服务接收到所有响应头字节时执行
body_filter()	body_filter_by_lua	从上游服务接收的响应体的每个块时执行。由于响应流回客户端，它可以超过缓冲区大小，因此，如果响应较大，该方法可以被多次调用

函数名	lua-nginx-module context	描述
log()	log_by_lua	当最后一个响应字节已经发送到客户端时执行

- *init_by_lua*

  > 在 *nginx* 重新加载配置文件时，运行里面 *lua* 脚本，常用于全局变量的申请。例如 *lua_shared_dict* 共享内存的申请，只有当 *nginx* 重启后，共享内存数据才清空，这常用于统计。

- *set_by_lua*

  > 设置一个变量，常用与计算一个逻辑，然后返回结果，该阶段不能运行Output API、Control API、Subrequest API、Cosocket API

- *rewrite_by_lua*

  > 在 *access* 阶段前运行，主要用于 *rewrite* url；

- *access_by_lua*

  > 主要用于**访问控制**，这条指令运行于 *nginx access* 阶段的末尾，因此总是在 *allow* 和 *deny* 这样的指令之后运行，它们同属 *access* 阶段。可用来判断请求是否具备访问权限；

- *content_by_lua*

  > 阶段是所有请求处理阶段中最为重要的一个，运行在这个阶段的配置指令一般都肩负着**生成内容**（*content*）**并输出 HTTP 响应**。

- *header_filter_by_lua*

> 一般只**用于设置 *Cookie* 和 *Headers*** 等。

- *body_filter_by_lua*

> 一般会在一次请求中被调用多次，因为这是实现基于 HTTP 1.1 *chunked* 编码的所谓“流式输出”的。

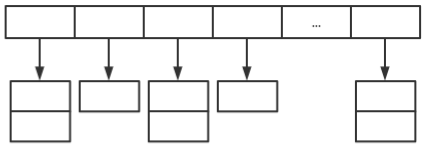
- *log_by_lua*

> 该阶段总是运行在请求结束的时候，用于请求的后续操作，如在共享内存中进行统计数据，如果要高精确的数据统计，应该使用 *body_filter_by_lua*

嵌入原理

```
ngx_http_phase_t phases[NGX_HTTP_LOG_PHASE + 1];
```

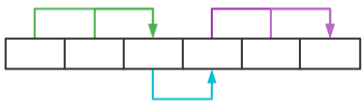
```
typedef ngx_int_t (*ngx_http_handler_pt)(ngx_http_request_t *r);
typedef struct {
    ngx_array_t                handlers;
} ngx_http_phase_t;
```



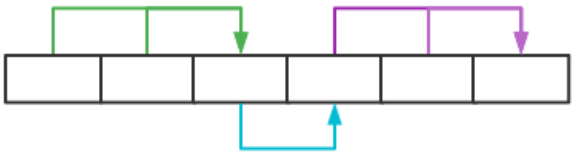
```
ngx_http_phase_engine_t phase_engine;
```

```
typedef struct {
    ngx_http_phase_handler_pt  checker;
    ngx_http_handler_pt        handler;
    ngx_uint_t                 next;
}ngx_http_phase_handler_t;

typedef struct {
    ngx_http_phase_handler_t  *handlers;
    ngx_uint_t                 server_rewrite_index;
    ngx_uint_t                 location_rewrite_index;
} ngx_http_phase_engine_t;
```



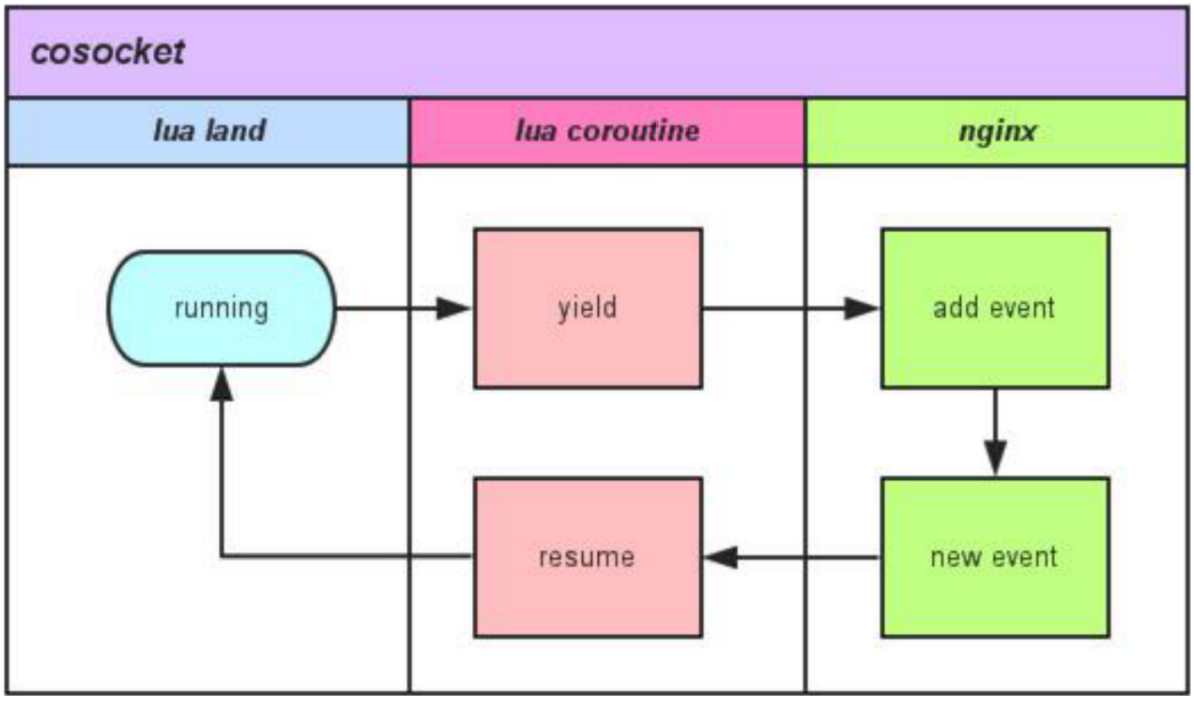
责任链模式



```
ph = cmcf->phase_engine.handlers;
while (ph[r->phase_handler].checker) {
    rc = ph[r->phase_handler].checker(r, &ph[r->phase_handler]);
    if (rc == NGX_OK) {
        return;
    }
}
```

cosocket

> *openresty* 为 *nginx* 添加的最核心的功能就是 *cosocket*；自 *cosocket* 加入，可以在 *http* 请求处理中访问第三方服务；*cosocket* 主要依据 *nginx* 中的事件机制和 *lua* 的协程结合后实现了**非阻塞网络 io**；在业务逻辑使用层面上可以通过**同步非阻塞**的方式来写代码；
>
> 引入 *cosocket* 后，*nginx* 中相当于有了多条**并行同步**逻辑线（*lua* 协程），*nginx* 中单线程负责唤醒或让出其中 *lua* 协程；唤醒或让出依据来源于协程运行的条件是否得到满足；
>
> 问题：比较 *openresty* 、*skynet*、*zvnet* 的 *lua* 虚拟机抽象和 *lua* 协程抽象？



openresty 应用（课上讲解）

- 黑白名单
- 反向代理-协议转换