咕泡学院 VIP 课：Nginx 的扩展-OpenRestry

课程目标

1. Nginx 进程模型简介

\2. Nginx 的高可用方案

\3. OpenResty 安装及使用

\4. 什么是 api 网关？

\5. OpenResty 实现灰度发布功能

Nginx 进程模型简介

多进程

多进程+多路复用

master 进程 、 worker 进程

```
root      7473     1  0 20:09 ?        00:00:00 nginx: master process .
/nginx
nobody    7474  7473  0 20:09 ?        00:00:00 nginx: worker process
```

worker_processes 1 cpu 总核心数

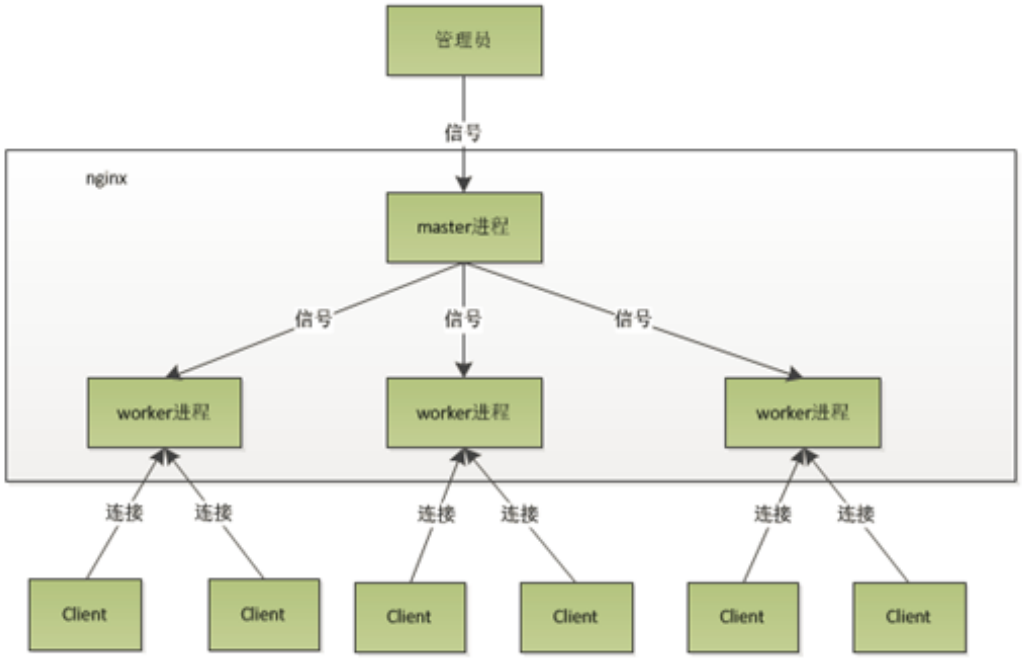
epoll . select ....

```
#user  nobody;            用户
worker_processes  1;      工作进程数

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
```

```
#pid        logs/nginx.pid;


events {
    use epoll ;  io 模型
    worker_connections  1024;    理论上  processes* connections
}
```



Nginx 的高可用方案

Nginx 作为反向代理服务器，所有的流量都会经过 Nginx，所以 Nginx 本身的可靠性是我们首先要考虑的问题

keepalived

Keepalived 是 Linux 下一个轻量级别的高可用解决方案，Keepalived 软件起初是专为 LVS 负载均衡软件设计的，用来管理并监控 LVS 集群系统中各个服务节点的状态，后来又加入了可以实现高可用的 VRRP 功能。因此，Keepalived 除了能够管理 LVS 软件外，还可以作为其他服务（例如：Nginx、Haproxy、MySQL 等）的高可用解决方案软件

Keepalived 软件主要是通过 VRRP 协议实现高可用功能的。VRRP 是 Virtual Router RedundancyProtocol(虚拟路由器冗余协议）的缩写，VRRP 出现的目的就是为了解决静态路由单点故障问题的，它能够保证当个别节点宕机时，整个网络可以不间断地运行;(简单来说，vrrp 就是把两台或多态路由器设备虚拟成一个设备，实现主备高可用)

所以，Keepalived 一方面具有配置管理 LVS 的功能，同时还具有对 LVS 下面节点进行健康检查的功能，另一方面也可实现系统网络服务的高可用功能

LVS 是 Linux Virtual Server 的缩写，也就是 Linux 虚拟服务器，在 linux2.4 内核以后，已经完全内置了 LVS 的各个功能模块。

它是工作在四层的负载均衡，类似于 Haproxy, 主要用于实现对服务器集群的负载均衡。

关于四层负载，我们知道 osi 网络层次模型的 7 层模模型（应用层、表示层、会话层、传输层、网络层、数据链路层、物理层）；四层负载就是基于传输层，也就是 ip+端口的负载；而七层负载就是需要基于 URL 等应用层的信息来做负载，同时还有二层负载（基于 MAC）、三层负载（IP）；

常见的四层负载有：LVS、F5； 七层负载有:Nginx、HAproxy; 在软件层面，Nginx/LVS/HAProxy 是使用得比较广泛的三种负载均衡软件

对于中小型的 Web 应用，可以使用 Nginx、大型网站或者重要的服务并且服务比较多的时候，可以考虑使用 LVS

轻量级的高可用解决方案

LVS 四层负载均衡软件（Linux virtual server）

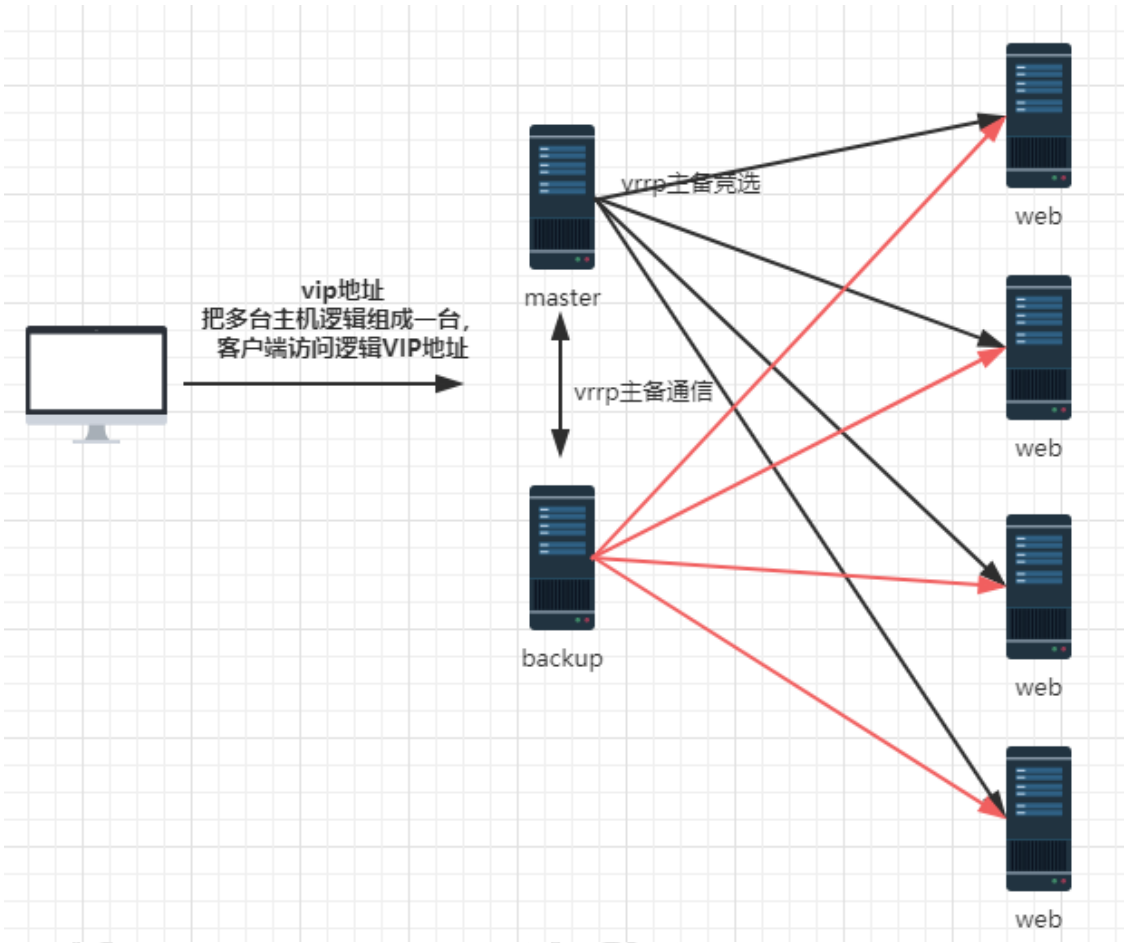
监控 lvs 集群系统中的各个服务节点的状态

VRRP 协议（虚拟路由冗余协议）

linux2.4 以后，是内置在 linux 内核中的

lvs(四层) -> HAproxy 七层

lvs(四层) -> Nginx(七层)



实践

\1. 下载 keepalived 的安装包

\2. tar -zxvf keepalived-2.0.7.tar.gz

\3. 在/data/program/目录下创建一个 keepalived 的文件

\4. cd 到 keepalived-2.0.7 目录下，执行 ./configure --prefix=/data/program/keepalived --sysconf=/etc

\5. 如果缺少依赖库，则 yum install gcc; yum install openssl-devel ; yum install libnl libnl-devel

\6. 编译安装 make && make install

\7. 进入安装后的路径 cd /data/program/keepalived, 创建软连接: ln -s sbin/keepalived /sbin

\8. cp /data/program/keepalived-2.0.7/keepalived/etc/init.d/keepalived /etc/init.d

\9. chkconfig --add keepalived

\10. chkconfig keepalived on

\11. service keepalived start

keepalived 的配置

master

```
! Configuration File for keepalived
global_defs {
   router_id LVS_DEVEL    运行 keepalived 服务器的标识，在一个网络内应该是唯一的
}
vrrp_instance VI_1 {    #vrrp 实例定义部分
    state MASTER  #设置 lvs 的状态，MASTER 和 BACKUP 两种，必须大写
    interface ens33   #设置对外服务的接口
    virtual_router_id 51    #设置虚拟路由标示，这个标示是一个数字，同一个 vrrp 实例使用唯一标示
    priority 100 #定义优先级，数字越大优先级越高，在一个 vrrp——instance 下，master 的优先级必须大于 backup
    advert_int 1 #设定 master 与 backup 负载均衡器之间同步检查的时间间隔，单位是秒
    authentication {  #设置验证类型和密码
        auth_type PASS
        auth_pass 1111    #验证密码，同一个 vrrp_instance 下 MASTER 和 BACKUP 密码必须相同
    }
    virtual_ipaddress { #设置虚拟 ip 地址，可以设置多个，每行一个
        192.168.11.100
    }
}

virtual_server 192.168.11.100 80 { #设置虚拟服务器，需要指定虚拟 ip 和服务端口
    delay_loop 6  #健康检查时间间隔
    lb_algo rr  #负载均衡调度算法
    lb_kind NAT  #负载均衡转发规则
    persistence_timeout 50  #设置会话保持时间
    protocol TCP #指定转发协议类型，有 TCP 和 UDP 两种

    real_server 192.168.11.160 80 { #配置服务器节点 1，需要指定 real server 的真实 IP 地址和端口
        weight 1  #设置权重，数字越大权重越高
```

```
        TCP_CHECK {  #realserver 的状态监测设置部分单位秒
           connect_timeout 3  #超时时间
           delay_before_retry 3 #重试间隔
           connect_port 80    #监测端口
        }
    }
}
```

backup

```
! Configuration File for keepalived

global_defs {
   router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 50
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.11.100
    }
}

virtual_server 192.168.11.100 80 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    real_server 192.168.11.161 80 {
        weight 1
        TCP_CHECK {
           connect_timeout 3
           delay_before_retry 3
           connect_port 80
        }
    }
}
```

keepalived 日志文件配置

\1. 首先看一下/etc/sysconfig/keepalived 文件

vi /etc/sysconfig/keepalived

KEEPALIVED_OPTIONS="-D -d -S 0"

“-D” 就是输出日志的选项

这里的“-S 0”表示 local0.* 具体的还需要看一下/etc/syslog.conf 文件

\2. 在/etc/rsyslog.conf 里添加:local0.* /var/log/keepalived.log

\3. 重新启动 keepalived 和 rsyslog 服务：

service rsyslog restart

service keepalived restart

通过脚本实现动态切换

\1. 在 master 和 slave 节点的 /data/program/nginx/sbin/nginx-ha-check.sh 目录下增加一个脚本

–no-headers 不打印头文件

Wc –l

统计行数

```
#!bin/sh   #! /bin/sh 是指此脚本使用/bin/sh 来执行

A=`ps -C nginx --no-header |wc -l`

if [ $A -eq 0 ]
   then
   echo 'nginx server is died'
   service keepalived stop
fi
```

\2. 修改 keepalived.conf 文件，增加如下配置

track_script: #执行监控的服务。

chk*nginx*service #

引用 VRRP 脚本，即在 vrrp_script 部分指定的名字。定期运行它们来改变优先级，并最终引发主备切换。

```
global_defs {
   router_id LVS_DEVEL
   enable_script_security
}
vrrp_script chk_nginx_service {
  script "/data/program/keepalived/nginx-ha-check.sh"
  interval 3
  weight -10
  user root
}
vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.11.100
    }
    track_script {
       chk_nginx_service
    }
```

Openresty

OpenResty 是一个通过 Lua 扩展 Nginx 实现的可伸缩的 Web 平台，内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地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。

安装

\1. 下载安装包

https://openresty.org/cn/download.html

\2. 安装软件包

tar -zxvf openresty-1.13.6.2.tar.gz

cd openrestry-1.13.6.2

./configure [默认会安装在/usr/local/openresty 目录] --prefix= 指定路径

make && make install

\3. 可能存在的错误，第三方依赖库没有安装的情况下会报错

yum install readline-devel / pcre-devel /openssl-devel

安装过程和 Nginx 是一样的，因为他是基于 Nginx 做的扩展

HelloWorld

开始第一个程序，HelloWorld

cd /data/program/openresty/nginx/conf

```
location / {
   default_type  text/html;
   content_by_lua_block {
      ngx.say("helloworld");
   }
}
```

在 sbin 目录下执行.nginx 命令就可以运行，看到 helloworld

建立工作空间

创建目录

或者为了不影响默认的安装目录，我们可以创建一个独立的空间来练习，先到在安装目录下创建 demo 目录,安装目录为/data/program/openresty/demo

mkdir demo

然后在 demo 目录下创建两个子目录，一个是 logs 、一个是 conf

创建配置文件

```
worker_processes 1;
error_log logs/error.log;
events {
        worker_connections 1024;
}
http {
```

```
    server {
        listen 8888;
        location / {
                default_type text/html;
                content_by_lua_block {
                        ngx.say("Hello world")
                }
        }
    }
}
```

执行：./nginx -p /data/program/openresty/demo 【-p 主要是指明 nginx 启动时的配置目录】

总结

我们刚刚通过一个 helloworld 的简单案例来演示了 nginx+lua 的功能，其中用到了 ngx.say 这个表达式，通过在 content*by*lua*block 这个片段中进行访问；这个表达式属于 ngx*lua 模块提供的 api， 用于向客户端输出一个内容。

库文件使用

通过上面的案例，我们基本上对 openresty 有了一个更深的认识，其中我们用到了自定义的 lua 模块。实际上 openresty 提供了很丰富的模块。让我们在实现某些场景的时候更加方便。可以在 /openresty/lualib 目录下看到；比如在 resty 目录下可以看到 redis.lua、mysql.lua 这样的操作 redis 和操作数据库的模块。

使用 redis 模块连接 redis

```
worker_processes 1;
error_log       logs/error.log;
events {
  worker_connections 1024;
}

http {
    lua_package_path '$prefix/lualib/?.lua;;'; 添加”;;”表示默认路径下的 l
ualib
    lua_package_cpath '$prefix/lualib/?.so;;';

   server {
     location /demo {
        content_by_lua_block {
              local redisModule=require "resty.redis";
              local redis=redisModule:new();   # lua 的对象实例
```

```
                redis:set_timeout(1000);
                ngx.say("=======begin connect redis server");
                local ok,err = redis:connect("127.0.0.1",6379);  #连接 re
dis

                if not ok then
                   ngx.say("==========connection redis failed,error mess
age:",err);
                end

                ngx.say("======begin set key and value");
                ok,err=redis:set("hello","world");
                if not ok then
                   ngx.say("set value failed");
                   return;
                end

                ngx.say("===========set value result:",ok);
                redis:close();
        }
    }
  }
}
```

演示效果

到 nginx 路径下执行 ./nginx -p /data/program/openresty/redisdemo

在浏览器中输入：http://192.168.11.160/demo 即可看到输出内容

并且连接到 redis 服务器上以后，可以看到 redis 上的结果

redis 的所有命令操作，在 lua 中都有提供相应的操作 .比如 redis:get("key")、redis:set()等

网关

通过扩展以后的，在实际过程中应该怎么去应用呢？一般的使用场景： 网关、web 防火墙、缓存服务器（对响应内容进行缓存，减少到达后端的请求，来提升性能），接下来重点讲讲网关的概念以及如何通过 Openresty 实现网关开发

网关的概念

从一个房间到另一个房间，必须必须要经过一扇门，同样，从一个网络向另一个网络发送信息，必须经过一道“关口”，这道关口就是网关。顾名思义，网关(Gateway)就是一个网络连接到另一个网络的“关口”。

那什么是 api 网关呢？

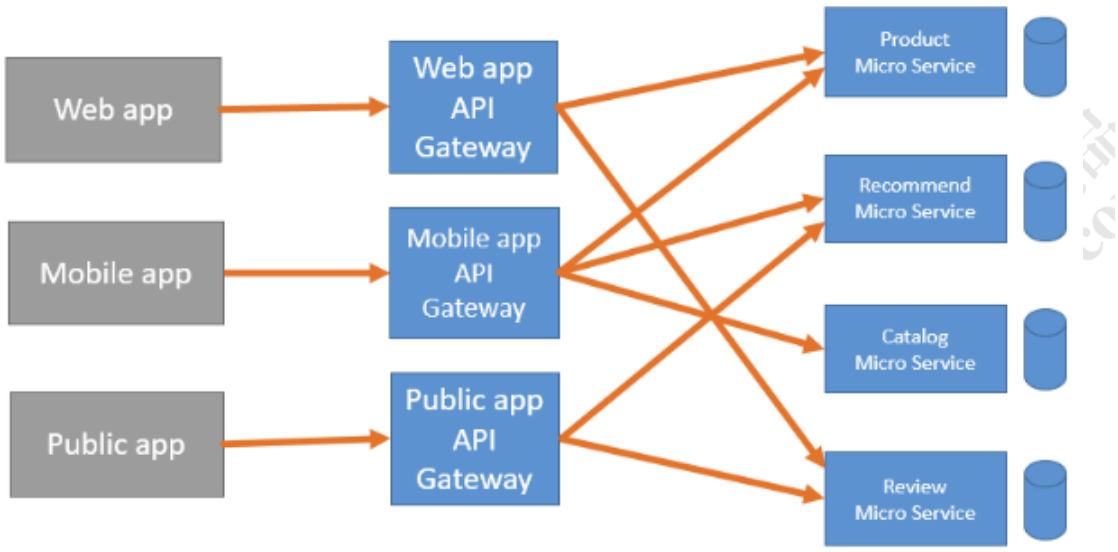
在微服务流行起来之前，api 网关就一直存在，最主要的应用场景就是开放平台，也就是 open api; 这种场景大家接触的一定比较多，比如阿里的开放平台；当微服务流行起来以后，api 网关就成了上层应用集成的标配组件

为什么需要网关？

对微服务组件地址进行统一抽象

API 网关意味着你要把 API 网关放到你的微服务的最前端，并且要让 API 网关变成由应用所发起的每个请求的入口。这样就可以简化客户端实现和微服务应用程序之间的沟通方式

Backends for frontends



当服务越来越多以后，我们需要考虑一个问题，就是对某些服务进行安全校验以及用户身份校验。甚至包括对流量进行控制。 我们会对需要做流控、需要做身份认证的服务单独提供认证功能，但是服务越来越多以后，会发现很多组件的校验是重复的。这些东西很明显不是每个微服务组件需要去关心的事情。微服务组件只需要负责接收请求以及返回响应即可。可以把身份认证、流控都放在 API 网关层进行控制

灰度发布

在单一架构中，随着代码量和业务量不断扩大，版本迭代会逐步变成一个很困难的事情，哪怕是一点小的修改，都必须要对整个应用重新部署。 但是在微服务中，各个模块是是一个独立运行的组件，版本迭代会很方便，影响面很小。

同时，为服务化的组件节点，对于我们去实现灰度发布（金丝雀发布：将一部分流量引导到新的版本）来说，也会变的很简单；

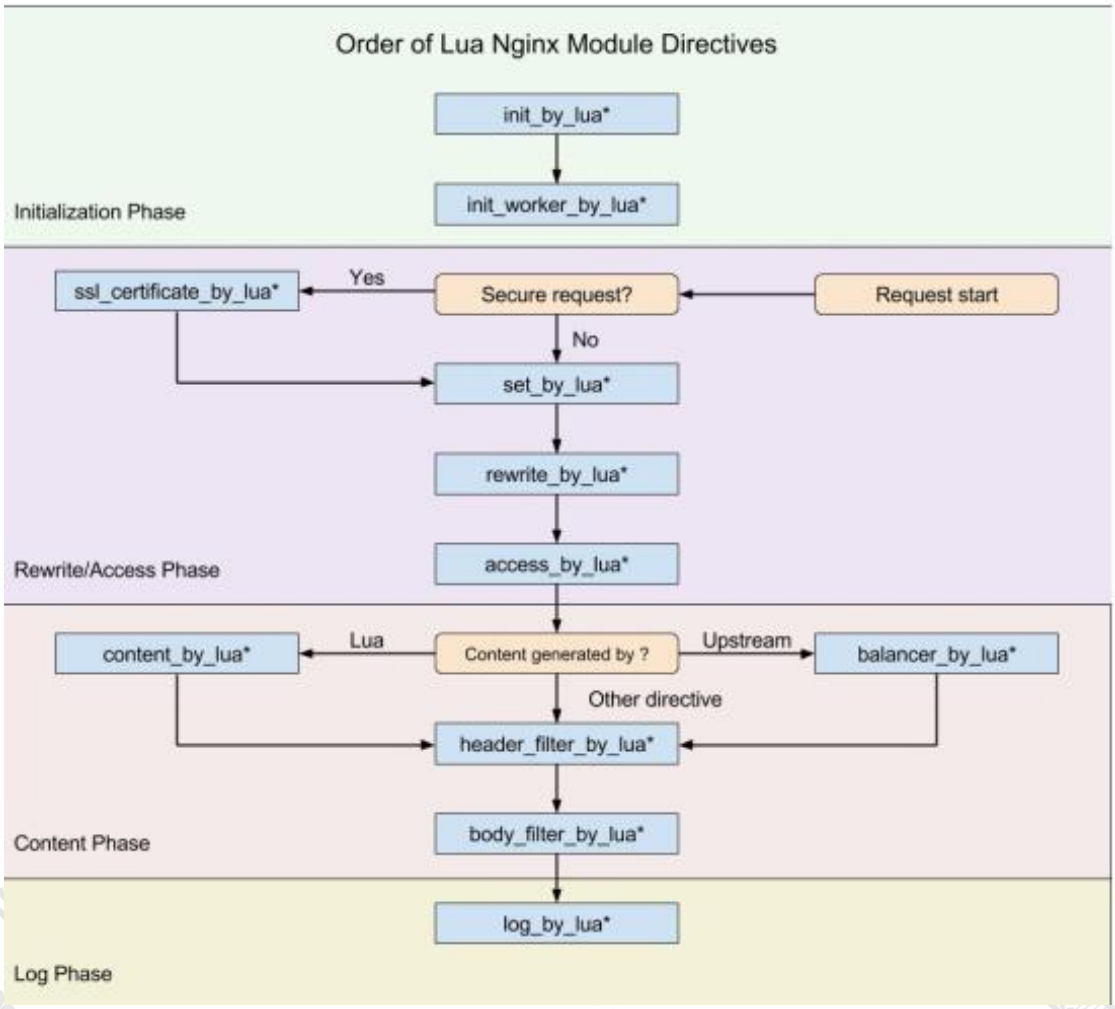
所以通过 API 网关，可以对指定调用的微服务版本，通过版本来隔离。如下图所示

OpenResty 实现 API 网关限流及登录授权

OpenResty 为什么能做网关？

前面我们了解到了网关的作用，通过网关，可以对 api 访问的前置操作进行统一的管理，比如鉴权、限流、负载均衡、日志收集、请求分片等。所以 API 网关的核心是所有客户端对接后端服务之前，都需要统一接入网关，通过网关层将所有非业务功能进行处理。

OpenResty 为什么能实现网关呢？ OpenResty 有一个非常重要的因素是，对于每一个请求，Openresty 会把请求分为不同阶段，从而可以让第三方模块通过挂载行为来实现不同阶段的自定义行为。而这样的机制能够让我们非常方便的设计 api 网关



Nginx 本身在处理一个用户请求时，会按照不同的阶段进行处理，总共会分为 11 个阶段。而 openresty 的执行指令，就是在这 11 个步骤中挂载 lua 执行脚本实现扩展，我们分别看看每个指令的作用

init*by*lua : 当 Nginx master 进程加载 nginx 配置文件时会运行这段 lua 脚本，一般用来注册全局变量或者预加载 lua 模块

init*woker*by_lua: 每个 Nginx worker 进程启动时会执行的 lua 脚本，可以用来做健康检查

set*by*lua:设置一个变量

rewrite*by*lua:在 rewrite 阶段执行，为每个请求执行指定的 lua 脚本

access*by*lua:为每个请求在访问阶段调用 lua 脚本

content*by*lua:前面演示过，通过 lua 脚本生成 content 输出给 http 响应

balancer*by*lua:实现动态负载均衡，如果不是走 content*by*lua，则走 proxy_pass,再通过 upstream 进行转发

header*filter*by_lua: 通过 lua 来设置 headers 或者 cookie

body*filter*by_lua:对响应数据进行过滤

log*by*lua： 在 log 阶段执行的脚本，一般用来做数据统计，将请求数据传输到后端进行分析

灰度发布的实现

1. 文件件目录， /data/program/openresty/gray [conf、logs、lua]

2. 编写 Nginx 的配置文件 nginx.conf

```
worker_processes 1;
error_log  logs/error.log;

events{
  worker_connections 1024;
}
http{
   lua_package_path "$prefix/lualib/?.lua;;";
   lua_package_cpath "$prefix/lualib/?.so;;";
   upstream prod {
       server 192.168.11.156:8080;
   }
   upstream pre {
       server 192.168.11.156:8081;
   }
   server {
       listen 80;
       server_name localhost;
       location /api {
          content_by_lua_file lua/gray.lua;
       }
       location @prod {
          proxy_pass http://prod;
       }
       location @pre {
          proxy_pass http://pre;
       }
   }
   server {
       listen 8080;
       location / {
          content_by_lua_block {
             ngx.say("I'm prod env");
          }
```

```
        }
    }
    server {
        listen 8081;
        location / {
            content_by_lua_block {
                ngx.say("I'm pre env");
            }
        }
    }
}
```

3. 编写 gray.lua 文件

```
local redis=require "resty.redis";
local red=redis:new();
red:set_timeout(1000);

local ok,err=red:connect("192.168.11.156",6379);

if not ok then
   ngx.say("failed to connect redis",err);
   return;
end

local_ip=ngx.var.remote_addr;

local ip_lists=red:get("gray");

if string.find(ip_lists,local_ip) == nil then
   ngx.exec("@prod");
else
   ngx.exec("@pre");
end

local ok,err=red:close();
```

4. \1. 执行命令启动 nginx: [./nginx -p /data/program/openresty/gray]

   \2. 启动 redis，并设置 set gray 192.168.11.160

   \3. 通过浏览器运行: http://192.168.11.160/api 查看运行结果

   修改 redis gray 的值， 讲客户端的 ip 存储到 redis 中 set gray 1. 再次运行结果，即可看到访问结果已经发生了变化