# Nginx简介

## 什么是 Nginx

Nginx 是俄罗斯人编写的十分轻量级的 HTTP 服务器,Nginx，它的发音为“engine X”，是一个高性能的HTTP和反向代理服务器，同时也是一个 IMAP/POP3/SMTP 代理服务器。

Nginx 因为它的稳定性、丰富的模块库、灵活的配置和低系统资源的消耗而闻名．业界一致认为它是 Apache2.2＋mod\_proxy\_balancer 的轻量级代替者，不仅是因为响应静态页面的速度非常快，而且它的模块数量达到 Apache 的近 2/3。对 proxy 和 rewrite 模块的支持很彻底，还支持 mod\_fcgi、ssl、vhosts ，适合用来做 mongrel clusters 的前端 HTTP 响应。

目前Nginx在国内很多大型企业都有应用，且普及率呈逐年上升趋势。选择Nginx的理由也很简单：

第一，它可以支持5W高并发连接；

第二，内存消耗少；

第三，成本低。

## Nginx在架构中发挥的作用

* **网关**

---面向客户的总入口。

* **虚拟主机**

---一台机器为不同的域名/ip/端口提供服务

* **路由**

---使用反向代理，整合后续服务为一个完整业务

* **静态服务器**

---mvvm模式中，用来发布前端html/css/js/img

* **负载集群**

---使用upstream，负载多个tomcat

# Nginx架构设计

## Nginx的模块化设计

高度模块化的设计是 Nginx 的架构基础。Nginx 服务器被分解为多个模块，每个模块就是一个功能模块，只负责自身的功能，模块之间严格遵循“高内聚，低耦合”的原则。



* **核心模块**

核心模块是 Nginx 服务器正常运行必不可少的模块，提供错误日志记录、配置文件解析、事件驱动机制、进程管理等核心功能。

* **标准 HTTP 模块**

标准 HTTP 模块提供 HTTP 协议解析相关的功能，如：端口配置、网页编码设置、HTTP 响应头设置等。

* **可选 HTTP 模块**

可选 HTTP 模块主要用于扩展标准的 HTTP 功能，让 Nginx 能处理一些特殊的服务，如：Flash 多媒体传输、解析 GeoIP 请求、SSL 支持等。

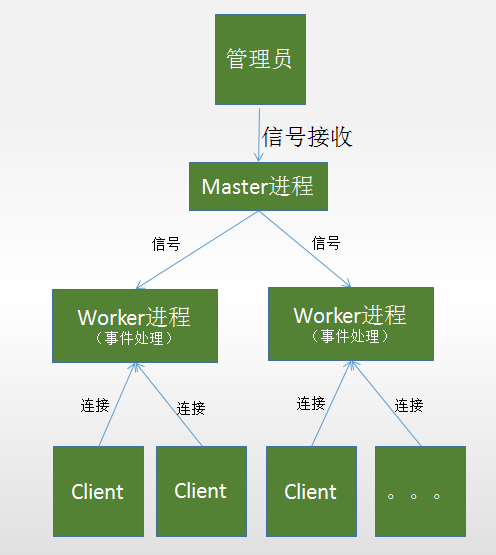
* **邮件服务模块**

邮件服务模块主要用于支持 Nginx 的邮件服务，包括对 POP3 协议、IMAP 协议和 SMTP 协议的支持。

* **第三方模块**

第三方模块是为了扩展 Nginx 服务器应用，完成开发者自定义功能，如：Json 支持、Lua 支持等。

## Nginx多进程模型



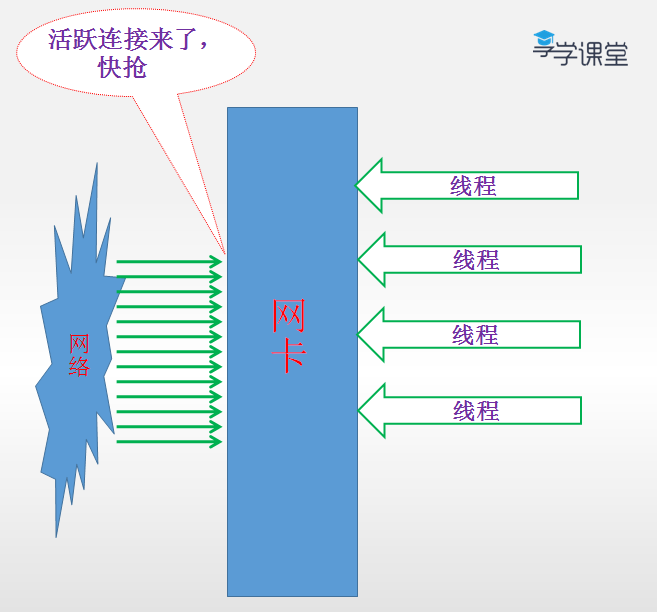
2.1、服务器每当收到一个客户端时。就有服务器主进程（master process）生成一个子进程（worker  process）出来和客户端建立连接进行交互，直到连接断开，该子进程结束。

2.2、使用进程的好处是各个进程之间相互独立，不需要加锁，减少了使用锁对性能造成影响，同时降低编程的复杂度，降低开发成本。

其次，采用独立的进程，可以让进程互相之间不会影响，如果一个进程发生异常退出时，其它进程正常工作，master 进程则很快启动新的 worker 进程，确保服务不中断，将风险降到最低。

缺点是操作系统生成一个子进程需要进行内存复制等操作，在资源和时间上会产生一定的开销；当有大量请求时，会导致系统性能下降。

## Nginx的epoll模式



select和poll的处理模式如上图：

--在某一时刻，进程收集所有的连接，其实这100万连接中大部分是没有事件发生的。因此，如果每次收集事件时，都把这100万连接的套接字传给操作系统（这首先就是用户态内存到内核内存的大量复制），而由操作系统内核寻找这些链接上没有处理的事件，将会是巨大的浪费。

而epoll改进了收集连接的动作，提高效率。

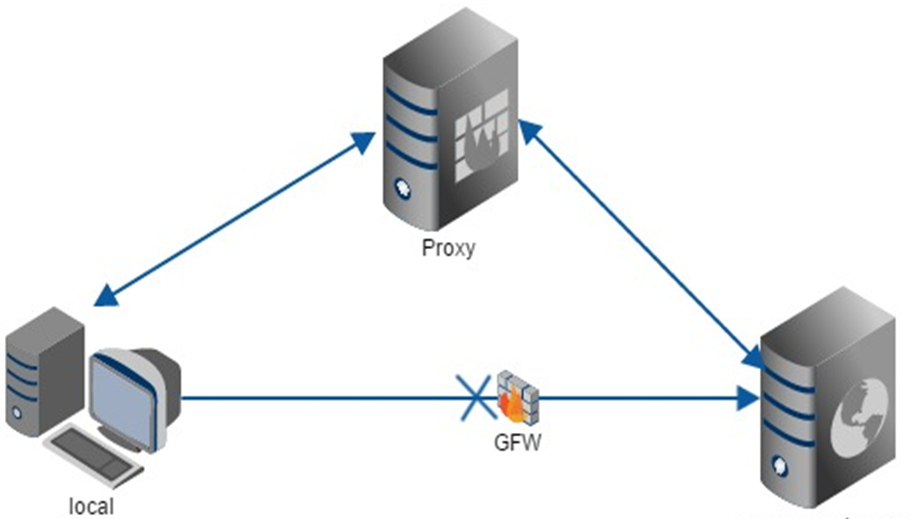
**epoll的优点：**

* 支持一个进程打开大数目的socket描述符(FD)
* IO效率不随FD数目增加而线性下降
* 使用mmap加速内核与用户空间的消息传递

## 正向代理与反向代理

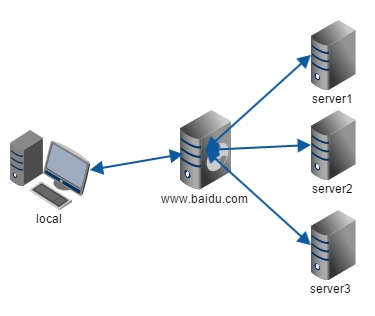
4.1、代理(正向代理)：意思是一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。

正向代理是代理了客户端



4.2、反向代理，服务端推出的一个代理招牌。

反向代理是代理了服务器



# nginx安装配置：

## 源码编译方式：

安装make：yum -y install autoconf automake make

安装g++: yum -y install gcc gcc-c++

#一般系统中已经装了了make和g++，无须再装

yum -y install pcre pcre-devel

yum -y install zlib zlib-devel

yum install -y openssl openssl-devel

#安装nginx依赖的库

wget  <http://nginx.org/download/nginx-1.15.8.tar.gz>

tar -zxvf nginx-1.15.8.tar.gz

cd nginx-1.15.8

./configure   --prefix=/usr/local/nginx --with-http\_stub\_status\_module --with-http\_ssl\_module

#配置

#--prefix指定安装目录

#--with-http\_ssl\_module安装https模块

#creating objs/Makefile 代表编译成功

make && make install

#make编译

#make install安装

## yum方式：

yum install yum-utils

yum-config-manager --add-repo https://openresty.org/package/centos/openresty.repo

yum install openresty

## Nginx目录结构：

* Conf  配置文件
* Html   网页文件
* Logs   日志文件
* Sbin    二进制程序

## Nginx常用命令

**启停命令：**

./nginx -c nginx.conf的文件。如果不指定，默认为NGINX\_HOME/conf/nginx.conf

./nginx -s stop  停止

./nginx -s quit退出

./nginx -s reload 重新加载nginx.conf

./nginx -t 检查nginx.conf配置

# nginx模型概念：

**Nginx会按需同时运行多个进程：**

一个主进程(master)和几个工作进程(worker)，配置了缓存时还会有缓存加载器进程(cache loader)和缓存管理器进程(cache manager)等。

所有进程均是仅含有一个线程，并主要通过“共享内存”的机制实现进程间通信。

主进程以root用户身份运行，而worker、cache loader和cache manager均应以非特权用户身份（user配置项）运行。

**主进程主要完成如下工作：**

1. 读取并验正配置信息；

2. 创建、绑定及关闭套接字；

3. 启动、终止及维护worker进程的个数；

4. 无须中止服务而重新配置工作特性；

5. 重新打开日志文件；

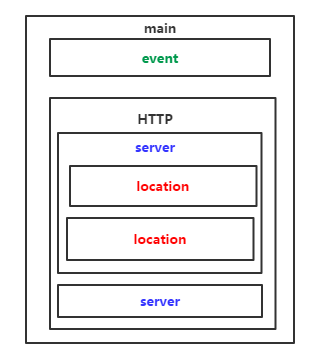
**worker进程主要完成的任务包括：**

1. 接收、传入并处理来自客户端的连接；

2. 提供反向代理及过滤功能；

3. nginx任何能完成的其它任务；

# nginx.conf配置文件结构



#user nobody; #主模块命令， 指定Nginx的worker进程运行用户以及用户组，默认由nobody账号运行。

worker\_processes 1;#指定Nginx要开启的进程数。

worker\_rlimit\_nofile 100000; #worker进程的最大打开文件数限制

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#pid logs/nginx.pid;

events {

use epoll;

worker\_connections 1024;

}

/\*

**以上这块配置代码是对nginx全局属性的配置。**

  user :主模块命令， 指定Nginx的worker进程运行用户以及用户组，默认由nobody账号运行。

  worker\_processes: 指定Nginx要开启的进程数。

  error log:用来定义全局错设日志文件的路径和日志名称。

日志输出级别有debug，info，notice，warn，error，crit 可供选择，其中debug输出日志最为详细，面crit（严重）输出日志最少。默认是error

  pid: 用来指定进程id的存储文件位置。

  event：设定nginx的工作模式及连接数上限，

其中参数use用来指定nginx的工作模式（这里是epoll，epoll是多路复用IO(I/O Multiplexing)中的一种方式）,

nginx支持的工作模式有select ,poll,kqueue,epoll,rtsig,/dev/poll。

其中select和poll都是标准的工作模式，kqueue和epoll是高效的工作模式，对于linux系统，epoll是首选。

    worker\_connection是设置nginx每个进程最大的连接数，默认是1024，所以nginx最大的连接数max\_client=worker\_processes \* worker\_connections。

进程最大连接数受到系统最大打开文件数的限制，需要设置ulimit。

\*/

**#下面部分是nginx对http服务器相关属性的设置**

http {

include mime.types; 主模块命令，对配置文件所包含文件的设定，减少主配置文件的复杂度，相当于把部分设置放在别的地方，然后在包含进来，保持主配置文件的简洁

default\_type application/octet-stream; 默认文件类型，当文件类型未定义时候就使用这类设置的。

#log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" ' 指定nginx日志的格式

# '$status $body\_bytes\_sent "$http\_referer" '

# '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

#access\_log logs/access.log main;

sendfile on; 开启高效文件传输模式（zero copy 方式），避免内核缓冲区数据和用户缓冲区数据之间的拷贝。

#tcp\_nopush on; 开启TCP\_NOPUSH套接字（sendfile开启时有用）

#keepalive\_timeout 0; 客户端连接超时时间

keepalive\_timeout 65;

#gzip on; 设置是否开启gzip模块

**#下面是server段虚拟主机的配置**

server {

listen 80; 虚拟主机的服务端口

server\_name localhost; 用来指定ip或者域名，多个域名用逗号分开

#charset koi8-r;

location / {

#地址匹配设置，支持正则匹配，也支持条件匹配，这里是默认请求地址，用户可以location命令对nginx进行动态和静态网页过滤处理

root html; 虚拟主机的网页根目录

index index.html index.htm; 默认访问首页文件

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

# Nginx日志

Nginx日志对于统计、系统服务排错很有用。

Nginx日志主要分为两种：access\_log(访问日志)和error\_log(错误日志)。

通过访问日志我们可以得到用户的IP地址、浏览器的信息，请求的处理时间等信息。

错误日志记录了访问出错的信息，可以帮助我们定位错误的原因。

因此，将日志好好利用，可以得到很多有价值的信息。

查看日志命令：

1. tail -f /usr/local/nginx/logs/access.log

## 设置access\_log

访问日志主要记录客户端的请求。客户端向Nginx服务器发起的每一次请求都记录在这里。客户端IP，浏览器信息，referer，请求处理时间，请求URL等都可以在访问日志中得到。当然具体要记录哪些信息，你可以通过log\_format指令定义。

### 语法

1. access\_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]]; *# 设置访问日志*
2. access\_log off; *# 关闭访问日志*

* path 指定日志的存放位置。
* format 指定日志的格式。默认使用预定义的combined。
* buffer 用来指定日志写入时的缓存大小。默认是64k。
* gzip 日志写入前先进行压缩。压缩率可以指定，从1到9数值越大压缩比越高，同时压缩的速度也越慢。默认是1。
* flush 设置缓存的有效时间。如果超过flush指定的时间，缓存中的内容将被清空。
* if 条件判断。如果指定的条件计算为0或空字符串，那么该请求不会写入日志。
* 另外，还有一个特殊的值off。如果指定了该值，当前作用域下的所有的请求日志都被关闭。

### 示例

1. http {
2. include mime.types;
3. default\_type application/octet-stream;
4. log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '
5. '$status $body\_bytes\_sent "$http\_referer" '
6. '"$http\_user\_agent" "$http\_x\_forwarded\_for"';
7. ##日志格式使用默认的combined，指定日志的缓存大小为32k，日志写入前启用gzip进行压缩，压缩比使用默认值1，缓存数据有效时间为1分钟。
8. access\_log /var/logs/nginx-access.log buffer=32k gzip flush=1m;
9. ...
10. }

### 作用域

access\_log指令的作用域分别有http，server，location。

## log\_format自定义格式

默认的日志格式

1. log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '
2. '$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

各参数明细表：

|  |  |
| --- | --- |
| $remote\_addr | 客户端的ip地址(代理服务器，显示代理服务ip) |
| $remote\_user | 用于记录远程客户端的用户名称（一般为“-”） |
| $time\_local | 用于记录访问时间和时区 |
| $request | 用于记录请求的url以及请求方法 |
| $status | 响应状态码，例如：200成功、404页面找不到等。 |
| $body\_bytes\_sent | 给客户端发送的文件主体内容字节数 |
| $http\_user\_agent | 用户所使用的代理（一般为浏览器） |
| $http\_x\_forwarded\_for | 可以记录客户端IP，通过代理服务器来记录客户端的ip地址 |
| $http\_referer | 可以记录用户是从哪个链接访问过来的 |

## 设置error\_log

错误日志在Nginx中是通过error\_log指令实现的。该指令记录服务器和请求处理过程中的错误信息。

错误日志不支持自定义。

### 语法

1. error\_log path [level];

* path参数指定日志的写入位置。
* level参数指定日志的级别（不写为全部）。level可以是debug, info, notice, warn, error, crit, alert,emerg中的任意值（等级从低到高排列）。

只有日志的错误级别等于或高于level指定的值才会写入错误日志中。默认值是error。

### 示例

error\_log logs/error.log;

error\_log logs/error\_notice.log notice;

error\_log logs/error\_info.log info; ##可以将不同的错误类型分开存储

## 日志配置和及切割

/etc/init.d/rsyslog start #系统日志，如不开启，看不到定时任务日志

/etc/rc.d/init.d/crond start #定时任务开启

**编写sh：**

#!/bin/bash

#设置日志文件存放目录

LOG\_HOME="/usr/local/nginx/logs/"

#备分文件名称

LOG\_PATH\_BAK="$(date -d yesterday +%Y%m%d%H%M)"

#重命名日志文件

mv ${LOG\_HOME}/access.log ${LOG\_HOME}/access.${LOG\_PATH\_BAK}.log

mv ${LOG\_HOME}/error.log ${LOG\_HOME}/error.${LOG\_PATH\_BAK}.log

#向nginx主进程发信号重新打开日志

kill -USR1 `cat ${LOG\_HOME}/nginx.pid`

**配置cron：**

\*/1 \* \* \* \* /usr/local/nginx/sbin/logcut.sh

# nginx安装第三方模块echo

本堂课将要使用第三方模块ngx\_echo的功能，请重新配置添加到nginx插件中

##下载第三方模块

wget https://github.com/openresty/echo-nginx-module/archive/v0.61.tar.gz

tar -zxvf v0.61.tar.gz ##解压

cd nginx-1.15.8 ##进入nginx源码目录，准备重新配置nginx

##配置，--add-module指向模块目录即会安装插件到nginx中

./configure --add-module=/usr/local/src/echo-nginx-module-0.61/

make && make install

# 路由--Location的使用

## Location语法规则

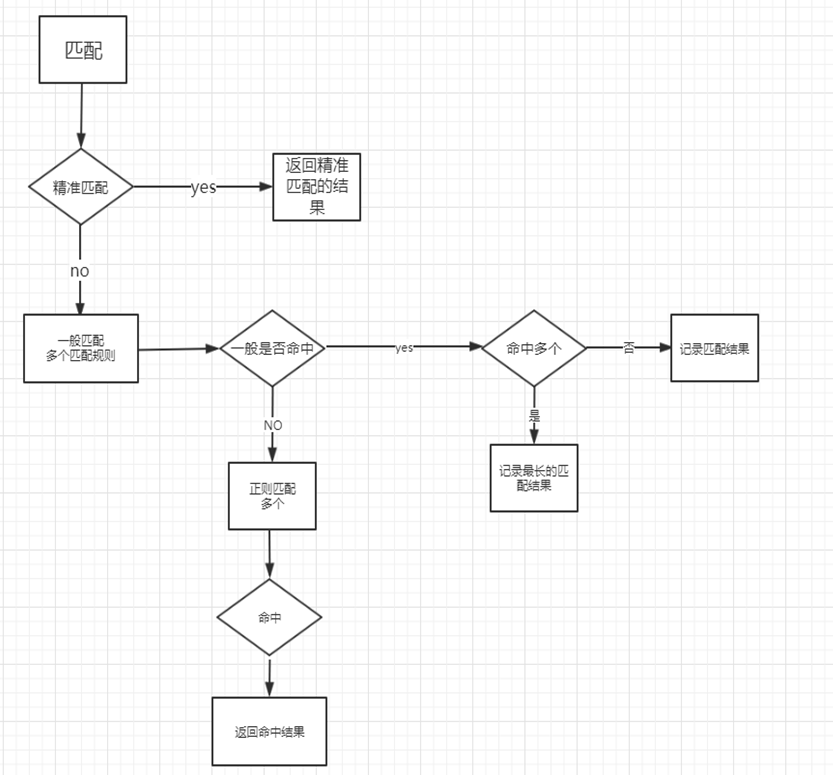
Location可以理解为Nginx路由的if-else

语法规则： location [=|~|~\*|^~] /uri/ {… }

首先匹配 =，其次匹配^~,其次是按文件中顺序的正则匹配，最后是交给 /通用匹配。当有匹配成功时候，停止匹配，按当前匹配规则处理请求。

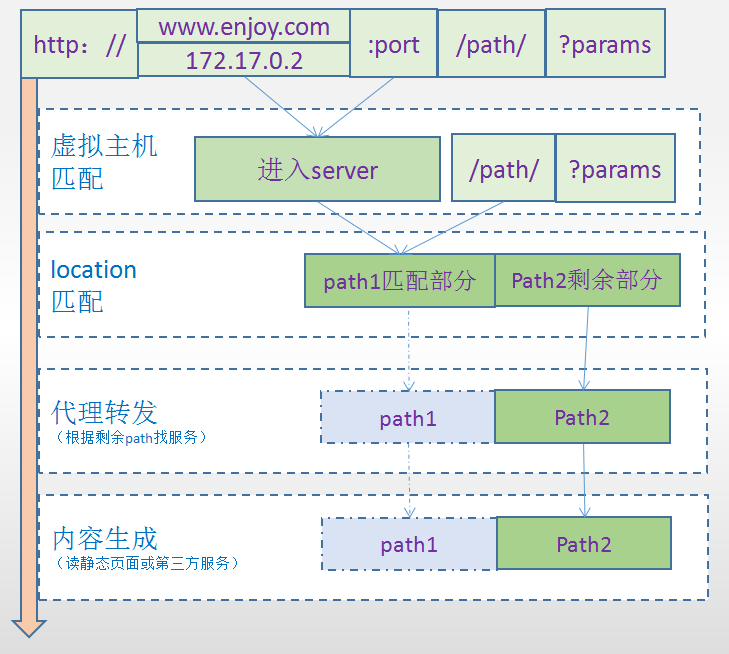
|  |  |
| --- | --- |
| 符号 | 含义 |
| = | = 开头表示精确匹配 |
| ^~ | ^~开头表示uri以某个常规字符串开头，理解为匹配 url路径即可（禁止正则匹配）。 |
| ~ | ~ 开头表示区分大小写的正则匹配 |
| ~\* | ~\* 开头表示不区分大小写的正则匹配 |
| !~和!~\* | !~和!~\*分别为区分大小写不匹配及不区分大小写不匹配的正则 |
| / | 用户所使用的代理（一般为浏览器） |

匹配规则优先级如下：



* =精准匹配命中时，停止location动作，直接走精准匹配，
* 一般匹配（含非正则）命中时，先收集所有的普通匹配，最后对比出最长的那一条
* 如果最长的那一条普通匹配声明为非正则，直接此条匹配，停止location
* 如果最长的那一条普通匹配是正则，继续往下走正则location
* 按代码顺序执行正则匹配，当第一条正则location命中时，停止location

## path匹配过程



假设http请求路径为

http://192.168.0.132:8088/mvc/index?id=2 ，匹配过程如下：

* 将整个url拆解为域名/端口/path/params
* 先由域名/端口，对应到目标server虚拟主机
* path部分参与location匹配，path = path1匹配部分 + path2剩余部分
* 进入location方法体内部流程。
* 若是静态文件处理，则进入目标目录查找文件：root指令时找path1+path2对应的文件;alias指令时找path2对应的文件
* 若是proxy代理，则形如proxy\_pass=ip:port时转发path1+path2路径到tomcat;形如proxy\_pass=ip:port/xxx时转发path2路径到tomcat。params始终跟随转发。

# rewrite使用：

**rewrite regex replacement [flag];**

**flag=【break/last/redirect/permanent】**

* regex 是正则表达式
* replacement 是替换值，新值
* flag -- 后续处理标识

## flag=break

发生nginx内部重定向，path值被更新，rewrite层面的命令会中断。原控制流程逻辑不变往下走

## flag=last

发生nginx内部重定向，path值被更新，rewrite层面的命令会中断。控制流程刷新，重新进行整个location层的逻辑流程。

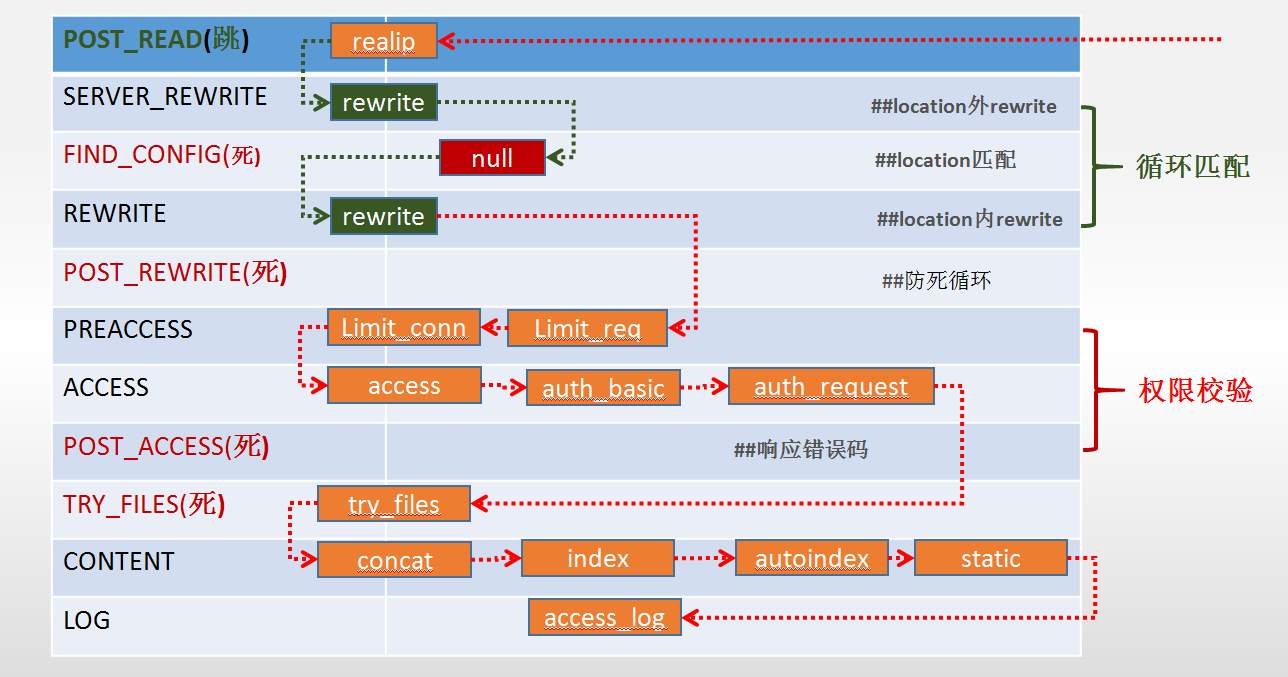
## flag= redirect/permanent

发生页面重定向（301永久重定向/302临时重定向），nginx流程结束，返回http响应到浏览器，页面url更新

## flag为空

发生nginx内部重定向，path值被更新，rewrite层面的命令继续。最后一个rewrite完毕，刷新控制流程，重新进行location重匹配

# Nginx处理请求的11个阶段



Nginx 处理请求的全过程一共划分为 11 个阶段（如图），按阶段由上到下依次执行 （上一阶段的所有指令执行完毕，才进入下一阶段）

各阶段的含义如下：

* post-read: 接收到完整的http头部后处理的阶段，在uri重写之前。一般跳过
* server-rewrite: location匹配前，修改uri的阶段，用于重定向，location块外的重写指令（**多次执行**）
* find-config: uri寻找匹配的location块配置项（**多次执行**）
* rewrite: 找到location块后再修改uri，location级别的uri重写阶段（**多次执行**）
* post-rewrite: 防死循环，跳转到对应阶段
* preaccess:  权限预处理
* access: 判断是否允许这个请求进入
* post-access: 向用户发送拒绝服务的错误码，用来响应上一阶段的拒绝
* try-files: 访问静态文件资源
* content : 内容生成阶段，该阶段产生响应，并发送到客户端
* log: 记录访问日志

# upstream--负载

语法格式：

upstream 负载名 {

[ip\_hash;]

server ip:port [weight=数字] [down];

server ip:port [weight=数字];

}

[]内容为可选项

## 轮询（默认）

upstream order {

server 192.168.0.128:8383;

server 192.168.244.233:8383;

}

不配置weight（即默认weight均为1）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

## weight

upstream order {

server 192.168.0.128:8383 weight=3;

server 192.168.244.233:8383 weight=1 down;

}

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

down 暂时不参与负载

## ip\_hash

upstream order {

ip\_hash;

server 192.168.0.128:8383;

server 192.168.244.233:8383;

}

每个请求按访问ip的hash结果分配，这样同一客户端的请求总是发往同一个后端服务器，可以解决session的问题。

## 代理时的负载使用

格式：proxy\_pass http://负载名;

如下图，其传参到下游服务器的规则，与proxy\_pass = http://ip:port 一样



# Openresty使用

OpenResty是一个全功能的 Web 应用服务器。它打包了标准的 Nginx 核心，常用的第三方模块以及大多数依赖项。 可以把它看成是Nginx附加众多的第三方插件的合集。其主体是嵌入lua脚本的支持，让你能够使用lua灵活地处理运算逻辑。

本课程主要讲lua为Nginx带来的新的处理方式，及OpenResty组件的使用。

## Openresty的安装配置

### 简易的yum安装方式

此方式简单，缺点是无法干预启停插件

yum install yum-utils

yum-config-manager --add-repo https://openresty.org/package/centos/openresty.repo

yum install openresty

### 源码安装方式

wget <https://openresty.org/download/openresty-1.15.8.1.tar.gz>

tar -zxvf [openresty-1.15.8.1.tar.gz](https://openresty.org/download/openresty-1.15.8.1.tar.gz)

##选择需要的插件启用, --with-Components 激活组件，--without 则是禁止组件

./configure --without-http\_redis2\_module --with-http\_iconv\_module

make && make install

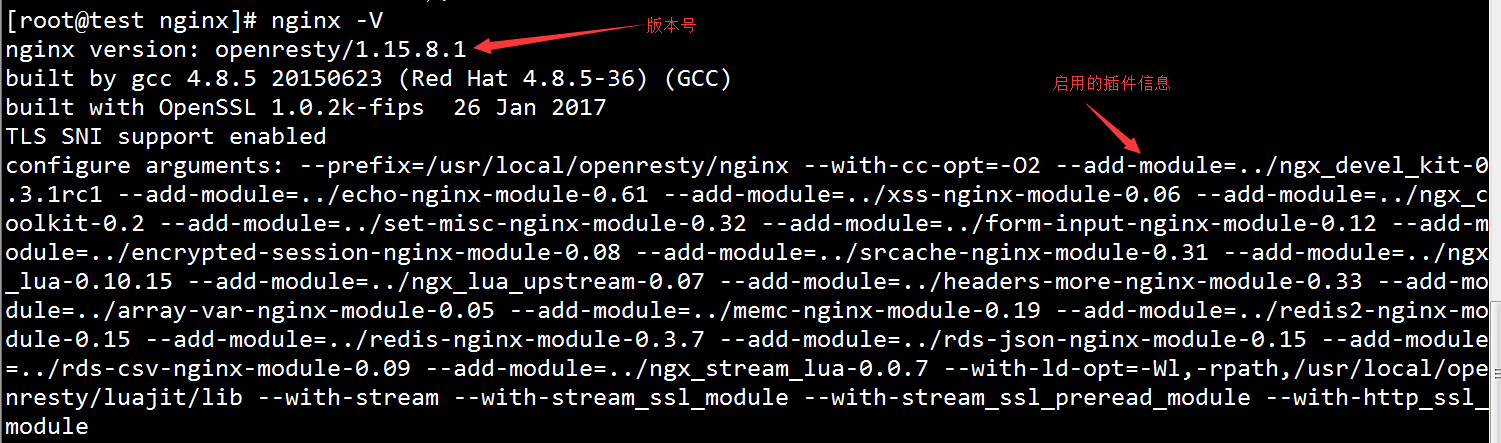
vi /etc/profile ##加入path路径

export PATH=$PATH:/usr/local/openresty/nginx/sbin/

source /etc/profile ##生效配置

### 安装检测

nginx -V ##如下显示，则表示安装成功



## Lua介入Nginx带来的基础api

主要帮助对http请求取参、取header头、输出等

|  |  |
| --- | --- |
| **ngx.arg** | 指令参数，如跟在content\_by\_lua\_file后面的参数 |
| **ngx.var** | request变量，ngx.var.VARIABLE引用某个变量 |
| **ngx.ctx** | 请求的lua上下文 |
| **ngx.header** | 响应头，ngx.header.HEADER引用某个头 |
| **ngx.status** | 响应码 |
| **ngx.log** | 输出到error.log |
| **ngx.send\_headers** | 发送响应头 |
| **ngx.headers\_sent** | 响应头是否已发送 |
| **ngx.resp.get\_headers** | 获取响应头 |
| **ngx.is\_subrequest** | 当前请求是否是子请求 |
| **ngx.location.capture** | 发布一个子请求 |
| **ngx.location.capture\_multi** | 发布多个子请求 |
| **ngx.print** | 输出响应 |
| **ngx.say** | 输出响应，自动添加‘\n‘ |
| **ngx.flush** | 刷新响应 |
| **ngx.exit** | 结束请求 |

## Lua嵌入Nginx的时机阶段

Nginx执行lua脚本片断时，需要明确指明执行的nginx阶段时机。主要有以下几种时机：

**set\_by\_lua\* : 设置nginx变量，实现复杂的赋值逻辑**

**rewrite\_by\_lua\* : 实现转发、重定向等功能**

**access\_by\_lua\* : IP 准入、接口访问权限等情况集中处理**

**content\_by\_lua\* : 接收请求处理并输出响应**

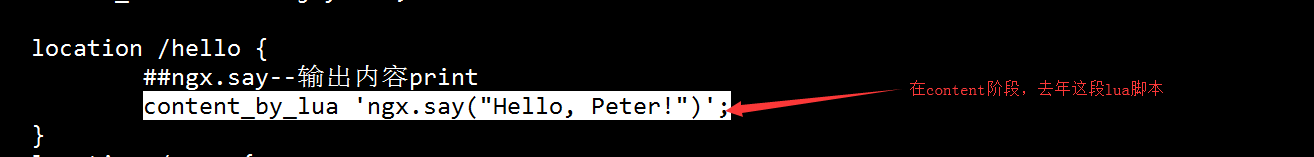
**header\_filter\_by\_lua\* : 设置header和cookie**

**body\_filter\_by\_lua\* : 对响应数据进行过滤，如截断/替换等**

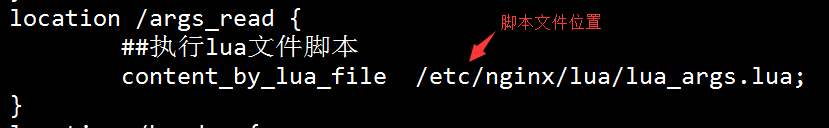
## Lua基础功能使用介绍

### hello world

在content阶段，执行lua脚本，输出hello，peter



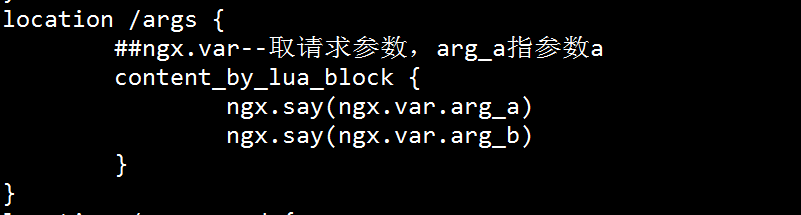
### 执行lua脚本文件



### lua取get参数

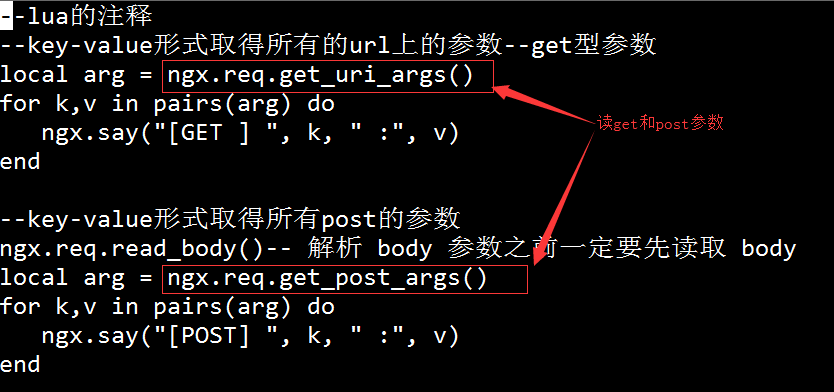
页面请求路径：<http://lua.enjoy.com/args?a=20&b=50>

则ngx.var.arg\_a即取得a参数值，如下图：

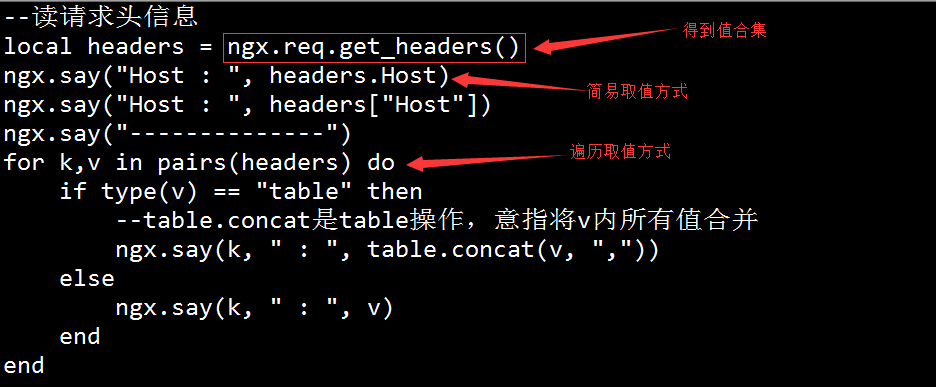


### lua取全量参数

请求：<http://lua.enjoy.com/args_read?a=20&b=50>

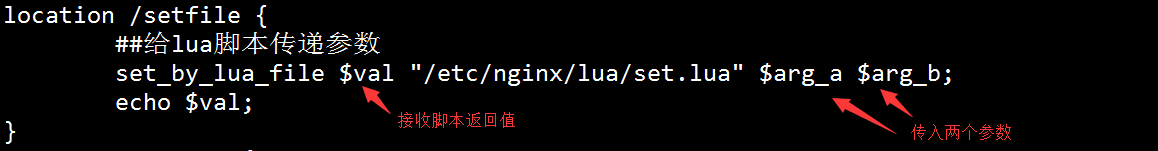


### lua取request中header信息

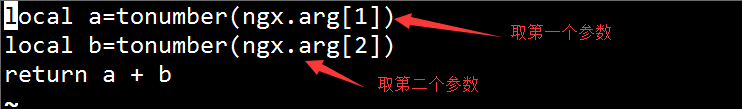


### 给lua脚本传参

使用端传参：

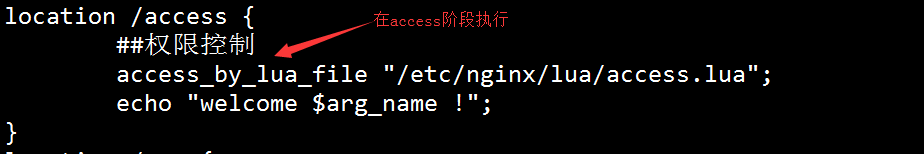


脚本中借助**ngx.arg取参**

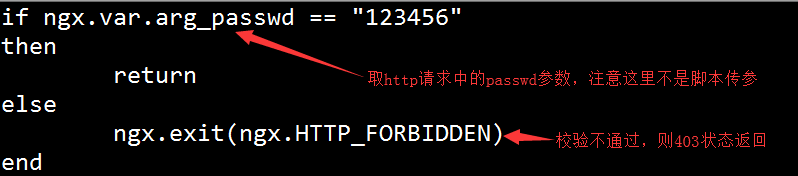
****

### 权限校验

一般校验动作，指定在access阶段执行脚本



脚本处理

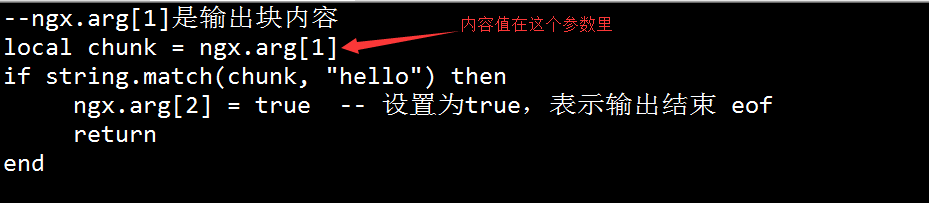


### 内容过滤

Nginx有时候，需要对下游服务生成的内容进行处理过滤，如下图



脚本中的处理



## Lua引入第三方模块的使用

OpenResty提供了非常多的第三方插件，支持操作redis/mysql等服务，lua使用它们的模式一般按以下流程

* require “resty/xxx” ：导入模块功能，类似java中的import导入类
* local obj = xxx:new() ：模块创建对象obj
* **local** ok, err = obj :connect ：对象连接到目标库
* obj :method ：这里可以为所欲为，尽情操纵目标库了

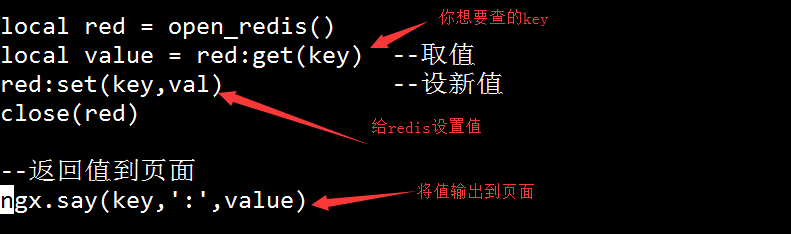
### Lua-resty-redis连接redis用法

**Lua-resty-redis插件，对Nginx操作redis的支持十分强大，成熟的用法演示如下：**

**基础的引入、连接动作**

****

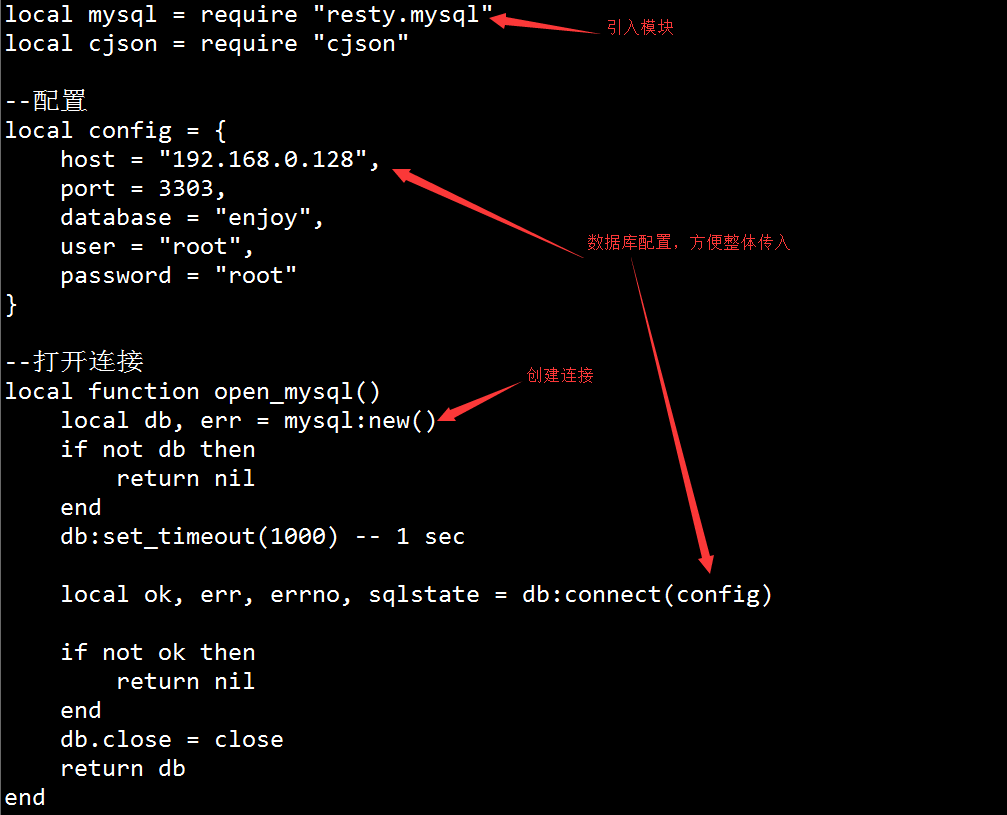
redis操作动作



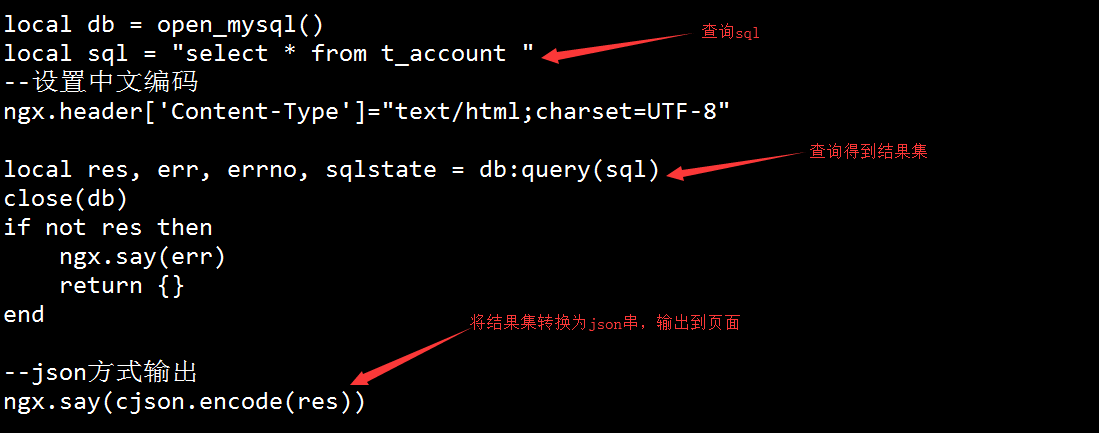
具体全量的程序，见源码配置包

### Lua-resty-mysql连接mysql数据库

**引入模块、创建连接**

****

**mysql查询操作**

****

# 小功能合集

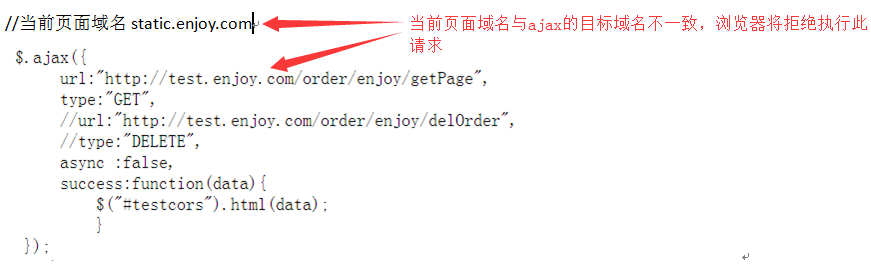
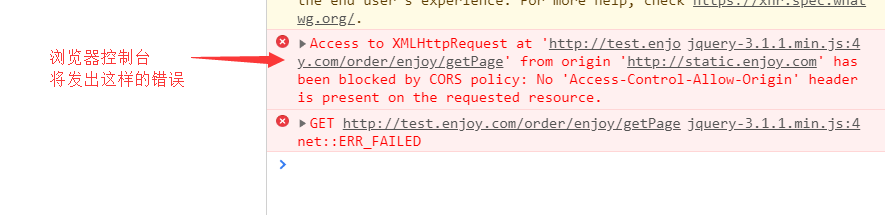
## 跨域处理

### 问题由来：浏览器拒绝执行其它域名下的ajax运作

---如果浏览器在static.enjoy.com对应的html页面内，发起ajax请求偷盗www.enjoy.com域名下的内容来填充自己的页面,整个互联网秩序将混乱.

为了防止这种混乱,W3C组织制定了浏览器安全规范，即html页面发起的ajax请求仅限于同域名后端范围，跨越域名的ajax请求不得执行，此谓跨域问题。

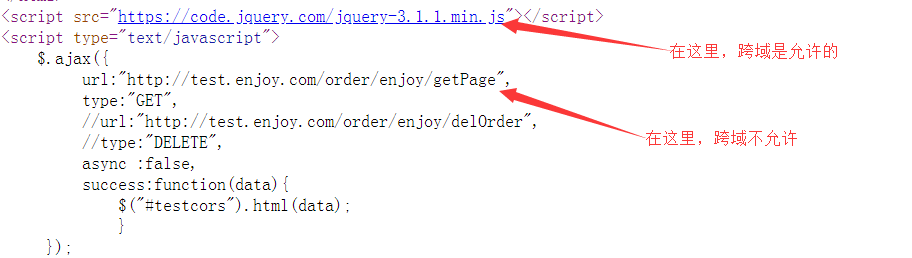
如下图：

而在日常工作中，我们自己有多个子系统，避免不了要有跨越子系统的ajax请求，此时，我们希望自己内部的各个子系统不必有这种跨域限制

### Jsonp的解决之道

w3c制定的规则不允许ajax跨域请求，却允许script标签发起跨域请求，如下：



因此，有人便扩展的script标签src源可以跨域的用法，来得到跨域名的请求信息。这便是jsonp的解决办法。

jsonp的方法有其不美的地方，主要是两点：

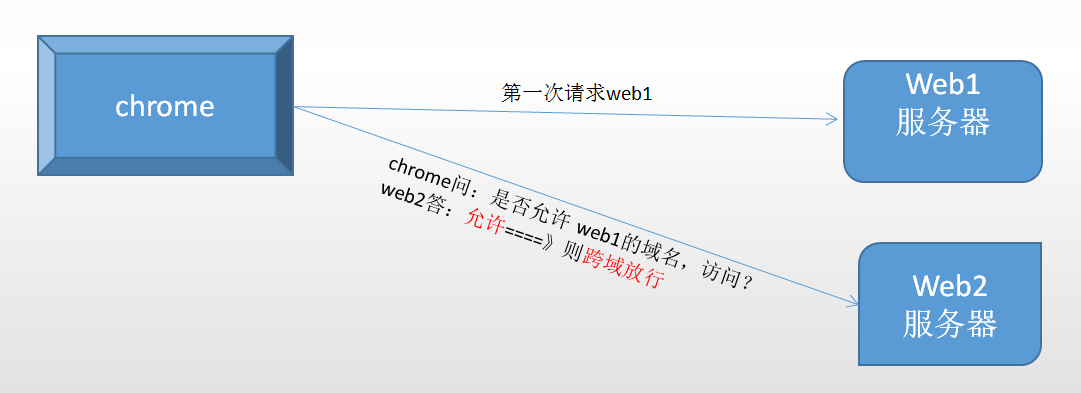
1.jsonp只能解决GET类的请求，其它类型的请求，script标签无法做到

2.使用jsonp的方式，对应的后台程序必须对结果进行改造。将返回值做一个函数式包装。这对业务开发有较大侵入性，增加开发人员负担

### cors方案的解决之道

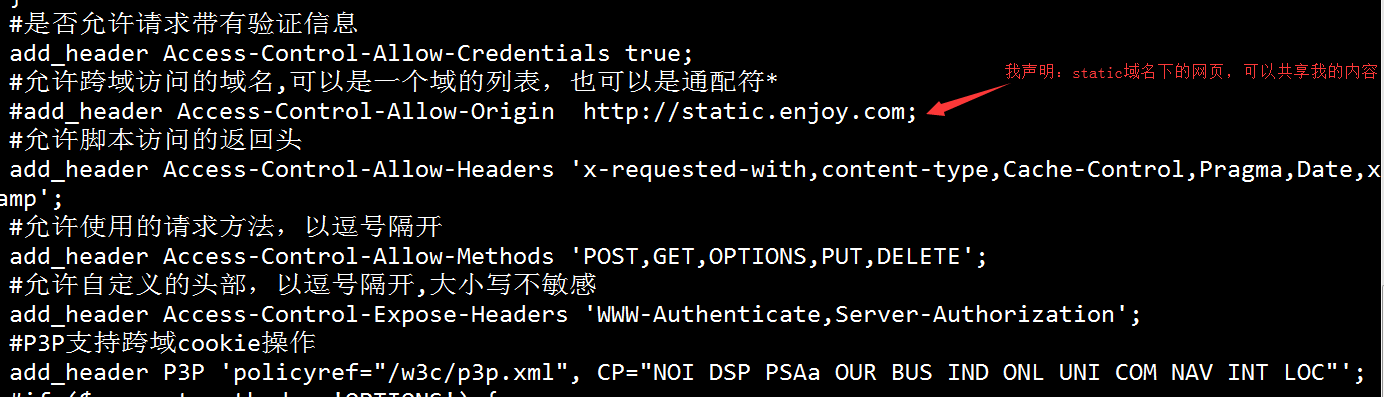
W3C制定跨域限制的本意，是防止页面领域安全混乱，即防止A公司不经B公司同意，使用ajax盗取B公司的服务内容。

出于这个本意，W3C改进了跨域的方案，即：如果B公司是同意将自己的内容分享给A公司的，跨域限制可放开，此方案即CORS方案，如下图：

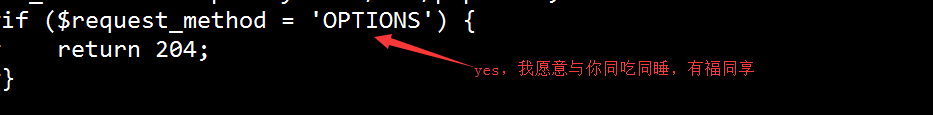


### nginx配置跨域操作

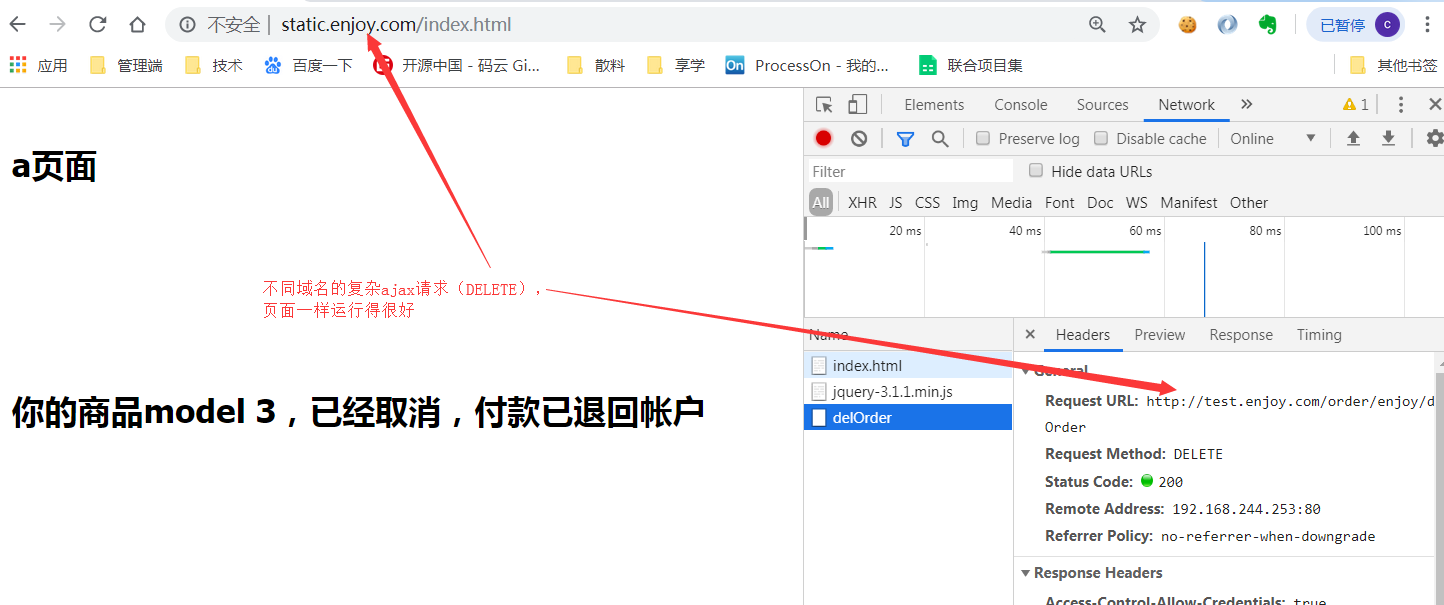
对于比较简单的http请求（GET、POST、HEAD类型），无须浏览器来问，nginx服务器直接在响应头部，加入同意跨域的信号即可



对于复杂的http请求（PUT、DELETE、含json格式数据），浏览器会在发请求前，先发一道OPTION请求来询问。我们在Nginx上直接配置对此询问的回答即可



有了上述Nginx的两道配置，跨域问题自然而解，对业务毫无侵入性。



## 防盗链

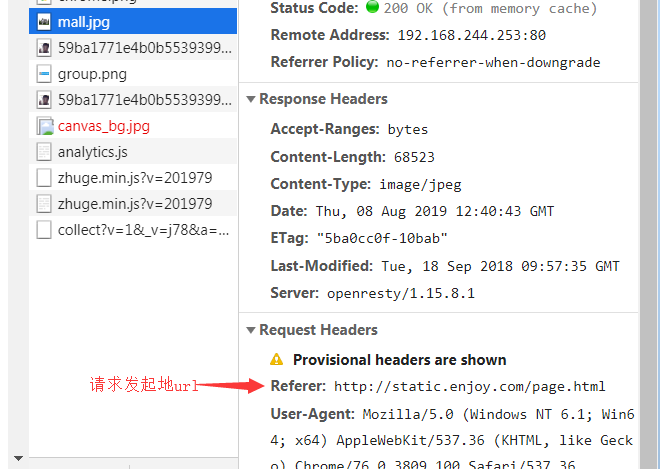
### 目标：

让资源只能在我的页面内显示，不能被其它页面直接引用



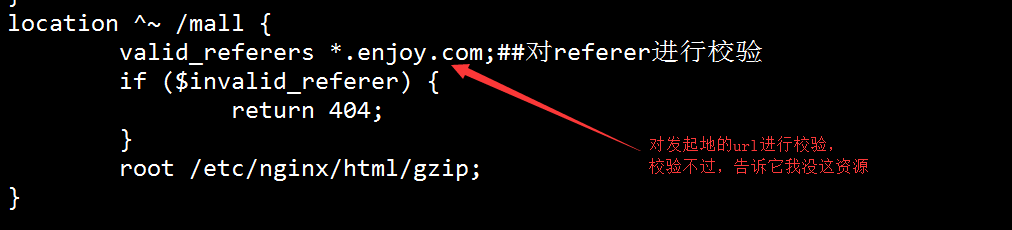
### 解决办法：

浏览器发起的任何请求，在其request头部，都会标注其请求发起地的URL，如下：



因此，在Nginx服务器上，只要校验此发起地url，就可以对应地拒绝响应它

### Nginx配置方法

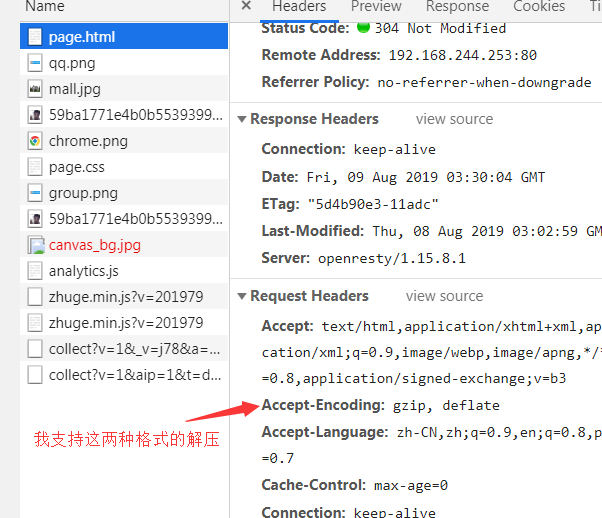


## 压缩

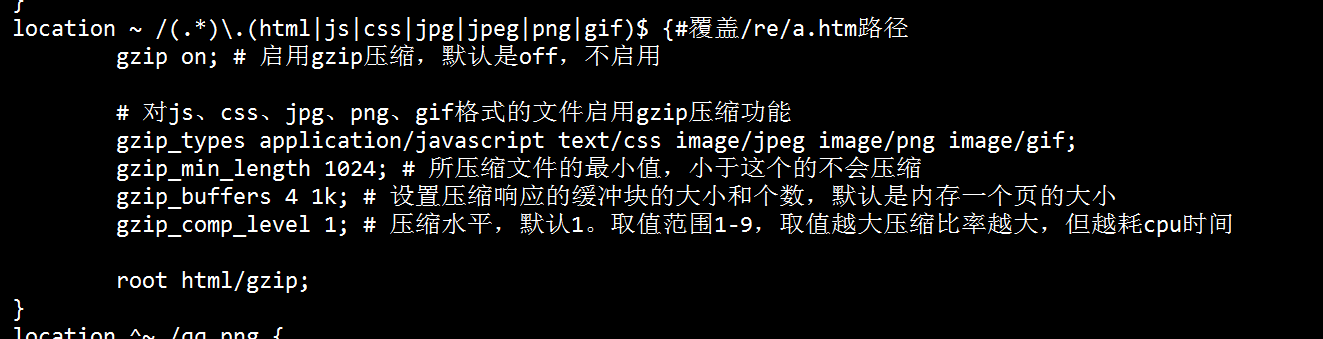
带宽资源很贵

--- /html/js/css压缩一下再传输，通常可减少50%的体积，何乐而不为

过程，浏览器在发送请求时，会附带自己支持的压缩方式：

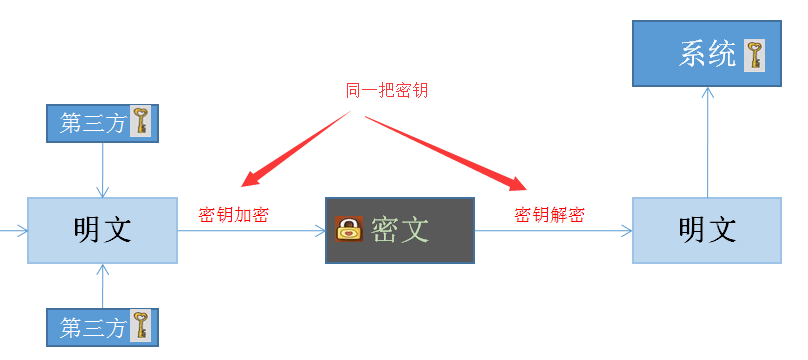


nginx配置



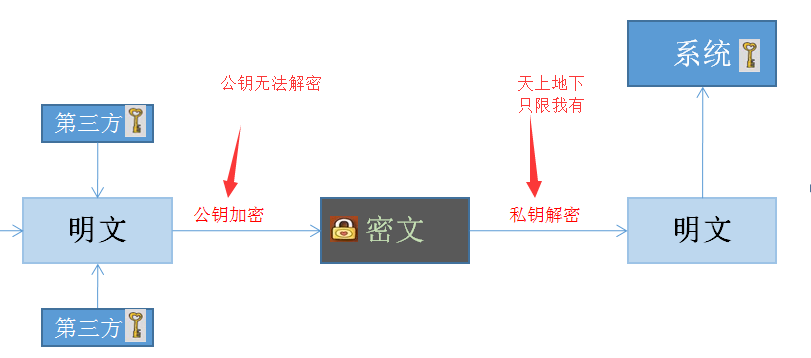
# https配置

## 对称加密



安全隐患：钥匙除我之外，还有多个人拥有。泄露风险较大，钥匙传递的过程风险较大

## 非对称加密



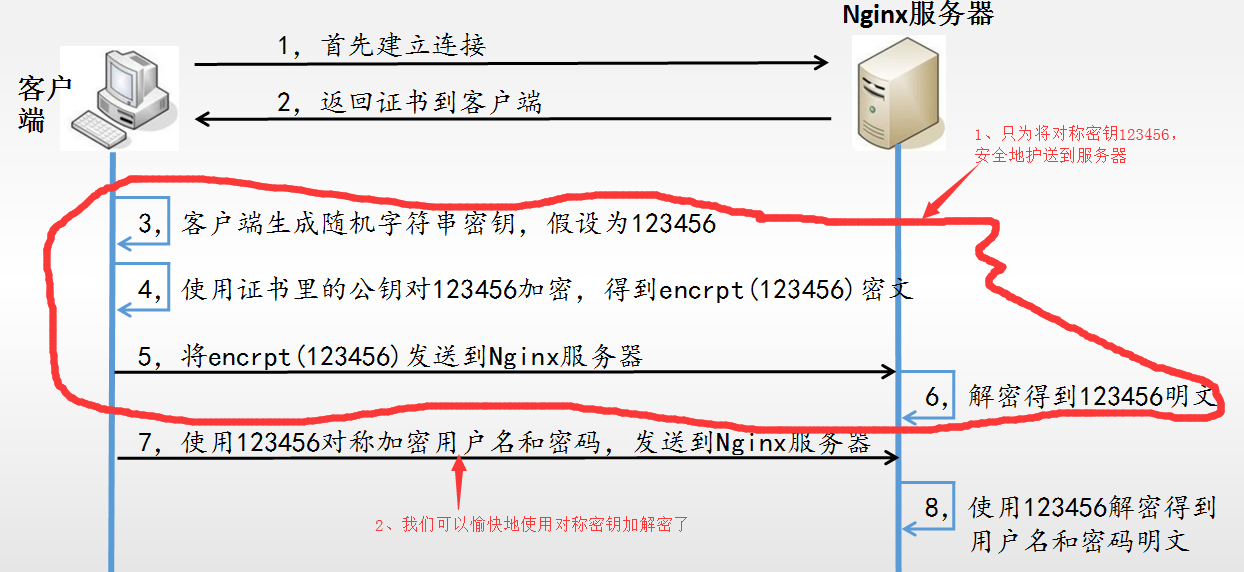
优缺点：私钥很安全。但是非对称算法开销很大，大批量应用于业务，会导致性能成本过高（太败家）。

## https加密方案

综合上述方案优缺点，各取所长，得到自己的方案

1、业务数据的加密使用对称加密，降低性能开销

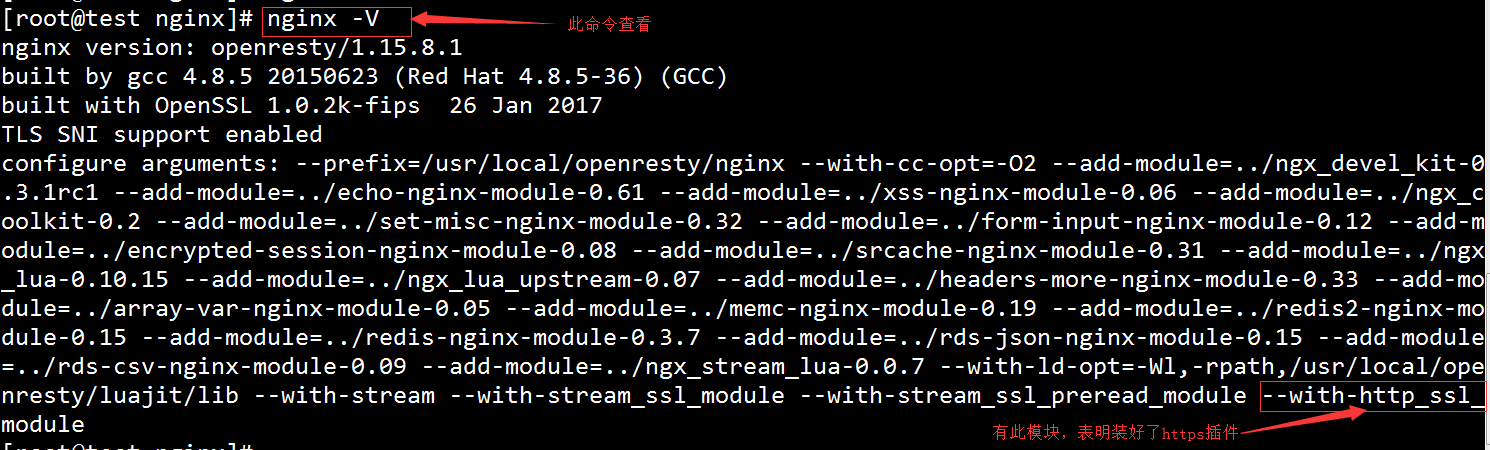
2、对称密钥，采用非对称加密，保驾护航



## Nginx配置https

#### 前提

查看nginx已经安装好了https模块（openresty默认是开启https模块的）：

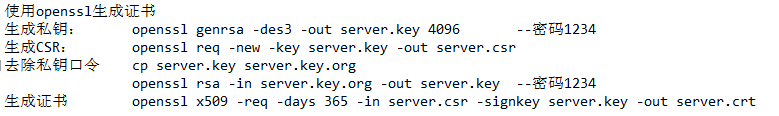


Nginx配置https只需要两个东东。一个是浏览器证书（内含公钥，供浏览器加密使用），一个是私钥（供自己解密使用）

server.crt和server.key可以自己去购买商业的。也可以自己使用程序生成一份（曾经的12306就使用自签的证书）

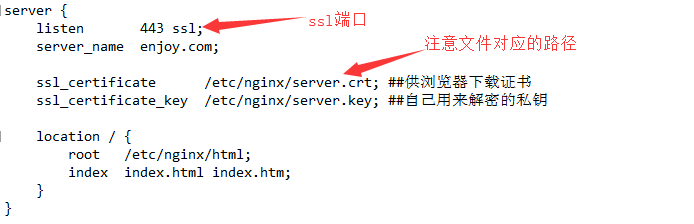
#### 自签证书

自签证书生成过程如下（前提是机器里装好了openssl程序，复制命令即可）：



#### Nginx配置

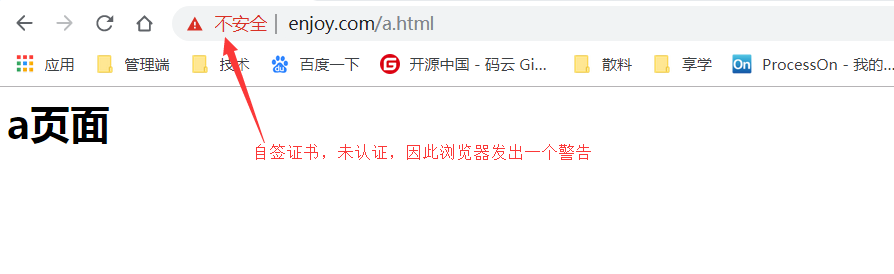
Nginx内的配置如下：



#### 校验

输入网址：https://enjoy.com/a.html

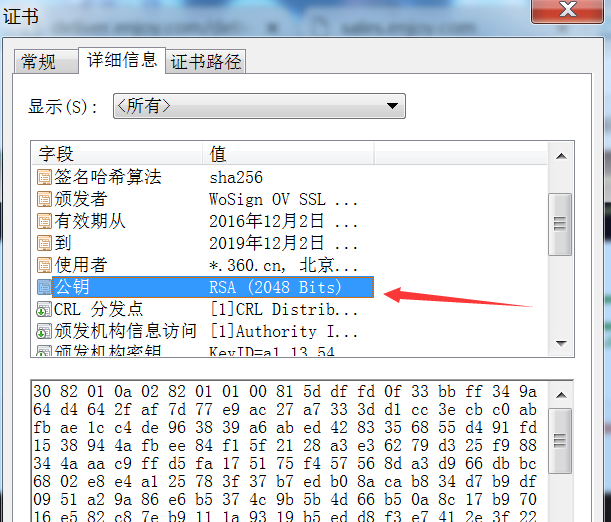
https方式显示页面如下：



查看证书



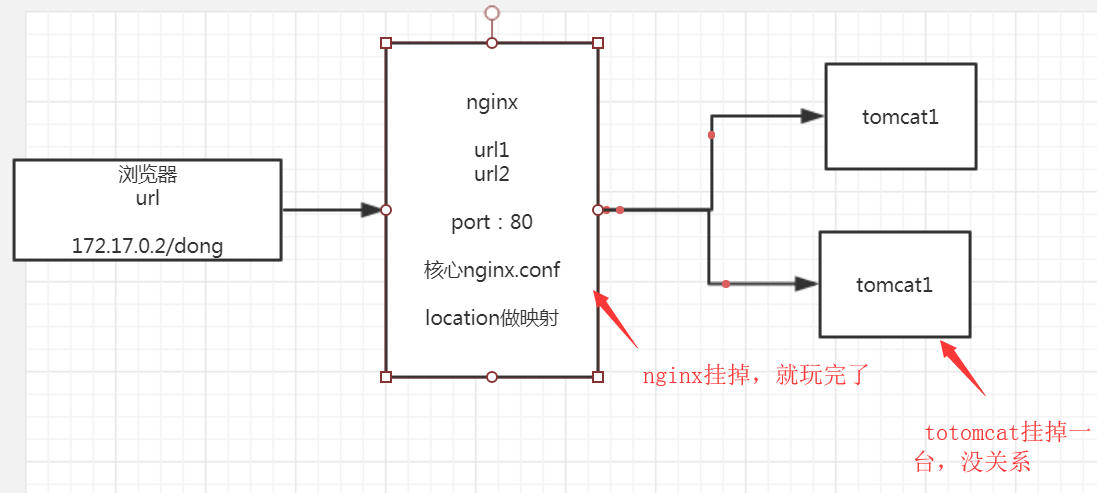
可看到证书只包含公钥字符串内容



# nginx高可用

## 传统的高可用思路

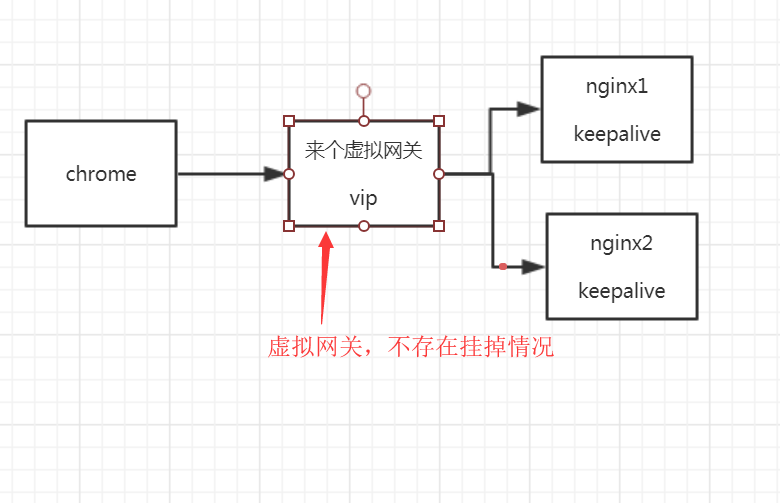
tomcat的高可用的思路，是在tomcat集群前面加一层负载服务nginx。如下图



这种做法，解决了tomcat的高可用问题。但是引入了前面的负载机器的高可用问题（Nginx如果挂了，玩完）

如果nginx沿用此思路，总会有一个最前端是单机的，存在宕机玩完的风险（鸡生蛋蛋生鸡无穷尽）

## lvs 思想解决高可用问题



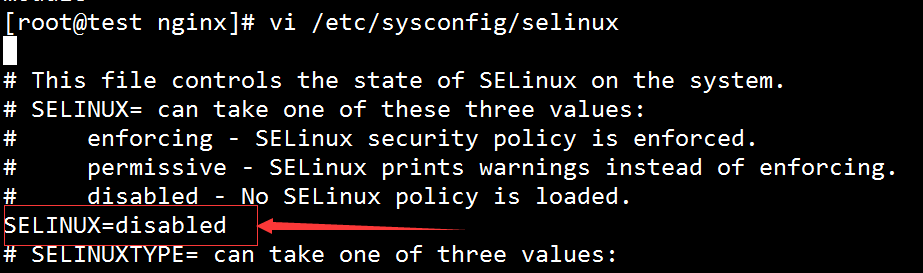
如上图，由服务器集群虚拟出来一台 虚拟网关vip（不真实存在，自然不存在宕机问题），

此vip由两台机器共同协商生成。当有一台机器宕机时，另一台机器一样能维持vip。这保证了，只要两台机器不同时宕机，vip就存在

## keepalived配置LVS过程

### 前提

1.关闭selinux，打开/etc/sysconfig/selinux设置其中值 🡪 SELINUX=disabled



2.安装必须的依赖包

yum -y install libnl libnl-devel libnfnetlink-devel

### keepalived安装

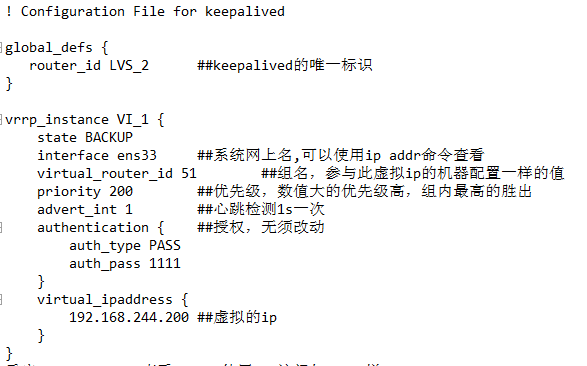
下载源码包--不能使用yum方式安装（有bug） --wget https://www.keepalived.org/software/keepalived-1.3.4.tar.gz

配置(指定安装目录和配置目录，否则文件太散乱) --./configure --prefix=/usr/local/keepalived --sysconf=/etc

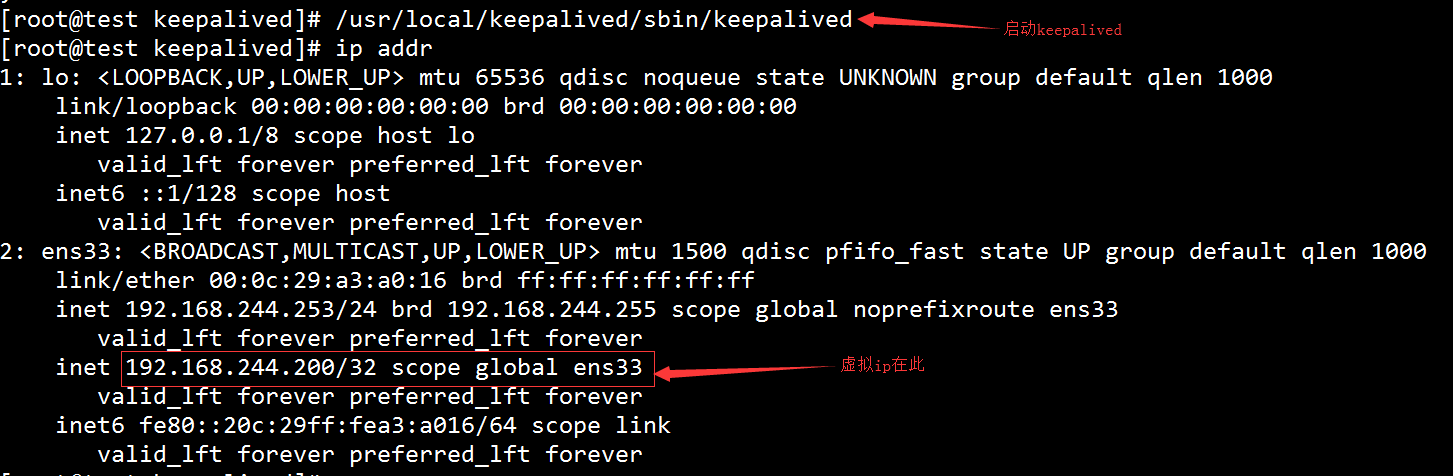
make && make install

### keepalived主机配置

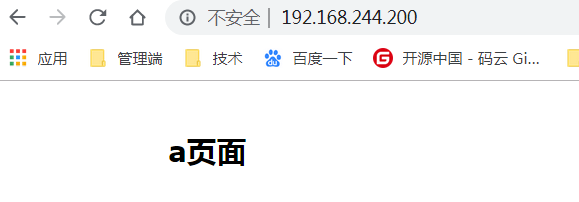
打开/etc/keepalived/keepalived.conf，只需要配置如下一段。（其它是多余配置，删除）



启动keepalived，查看机器ip地址，可发现多出一个244.200的ip

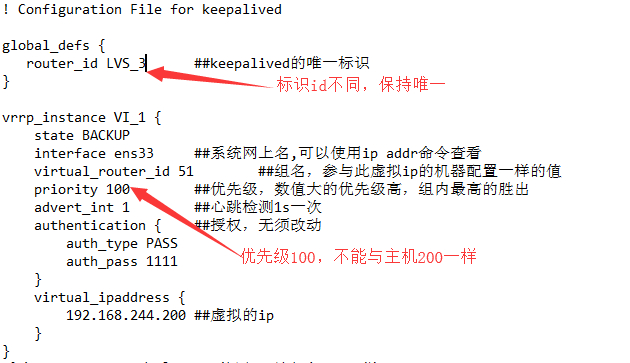


此时，使用原ip地址244.253能打开的页面，使用244.200也能打开



### keepalived从机配置

从机配置与主机过程完全一样，配置文件内以下标识id与优先级稍作变化即可



启动从机的keepalived后，可发现其ip地址无变化

### keepalived校验LVS效果

1、此时，杀掉主机上的keepalived，244.200的ip将从主机上消失。而出现的从机的ip中

2、再次启动主机的keepalived，244.200的ip将被主机重新夺回

3、此效果是单主单备方式。备机资源有一定的浪费。可以重复前面的动作，虚拟出第二个ip，将主从机优先级颠倒，从而利用起备机服务

### keepalived监控服务软件

以上操作中，keepalived很好的实现了LVS功能，即集群机器共同虚拟一个vip，并实现在集群中自动漂移。

但假如物理机状况良好，并不能保障其上运行的服务软件ok，因此需要借助keepalived来监控服务软件。

a、使用keepalived来监控nginx

编辑一个sh监控脚本，sh脚本：

#!/bin/bash

A=`ps -C nginx --no-header |wc -l` #统计nginx进程是否存在

if [ $A -eq 0 ];then #为0，表明nginx停止了

/usr/local/nginx/sbin/nginx #尝试重启nginx

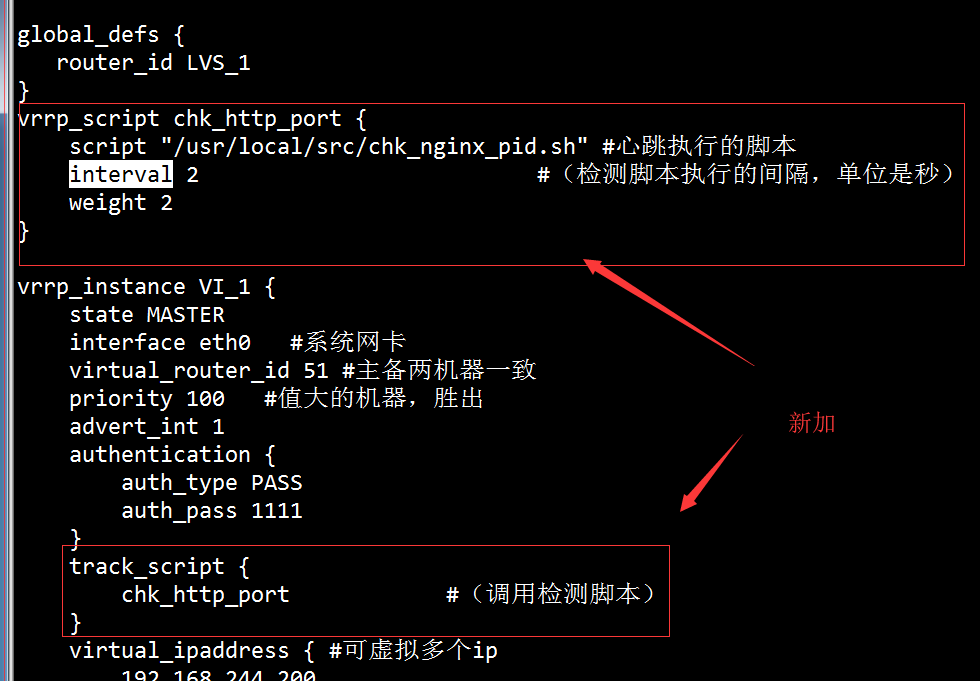
if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then #nginx重启失败，则keepalived自杀，进行VIP转移

killall keepalived #杀掉，vip就漫游到另一台机器

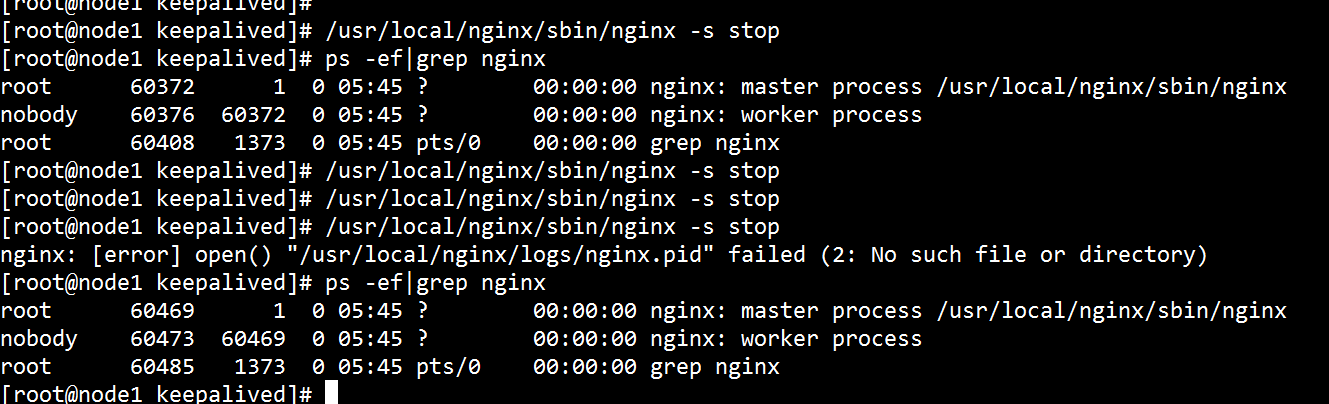
fi

fi

b、在配置文件中加入以下两处配置：



c、重启keepalived，测试监控效果，如下图操作：



nginx已变成不死鸟

# Nginx在mvvm模式中的使用：

