

一、 Nginx 简介

1.1. 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。

1.2. 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

二、 Nginx 架构设计

2.1. 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。



Nginx 模块图

- **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。

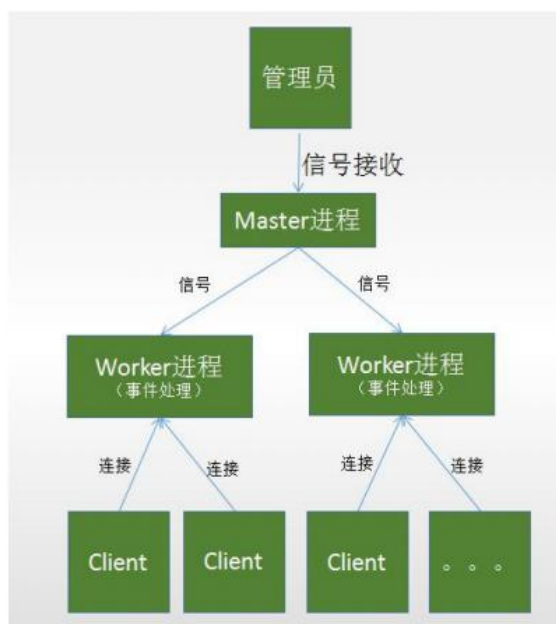
- **邮件服务模块**

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。

2.2. Nginx 多进程模型



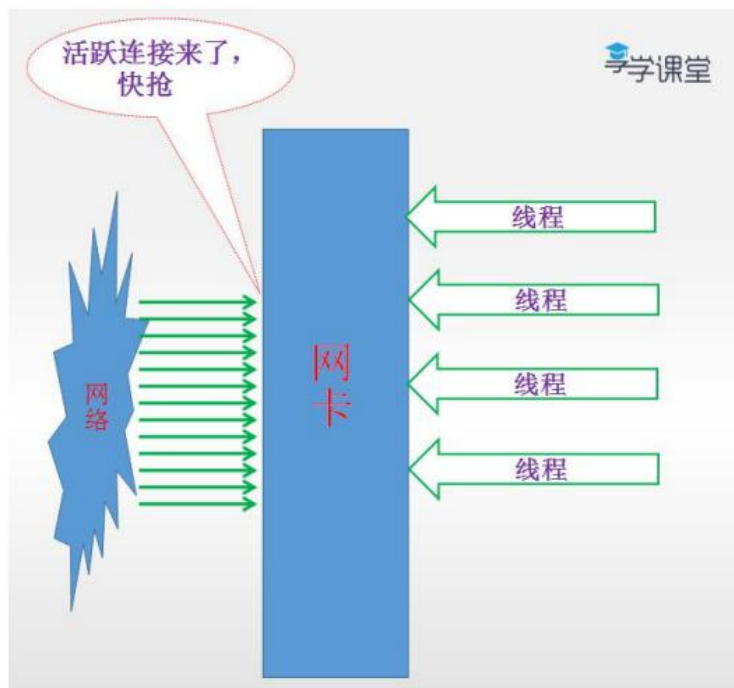
2.1、服务器每当收到一个客户端时，就有服务器主进程（master process）生成一个子进程（worker process）出来和客户端建立连接进行交互，直到连接断开，该子进程结束。

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受影响，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。

2.3. 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。

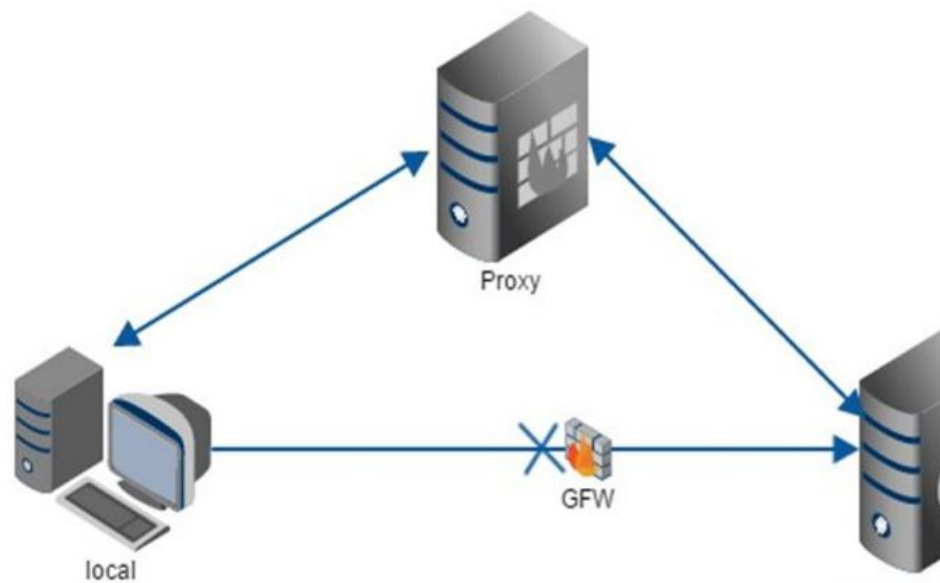
而 epoll 改进了收集连接的动作，提高效率。

epoll 的优点：

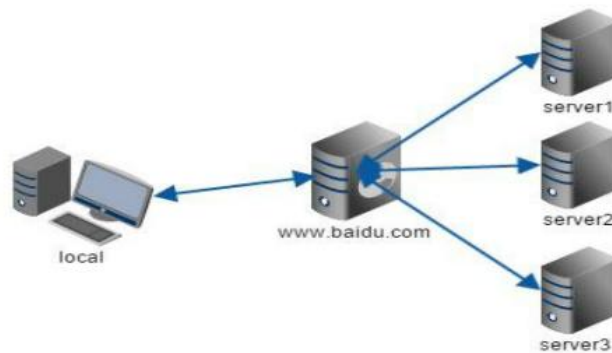
- ◇ 支持一个进程打开大数目的 socket 描述符(FD)
- ◇ IO 效率不随 FD 数目增加而线性下降
- ◇ 使用 mmap 加速内核与用户空间的消息传递

2.4. 正向代理与反向代理

4.1、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。



4.2、反向代理，服务端推出的一个代理招牌。



三、 nginx 安装配置：

3.1. 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 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

```

3.2. yum 方式:

```

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

```

3.3. 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

3.4. 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

```

四、 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。

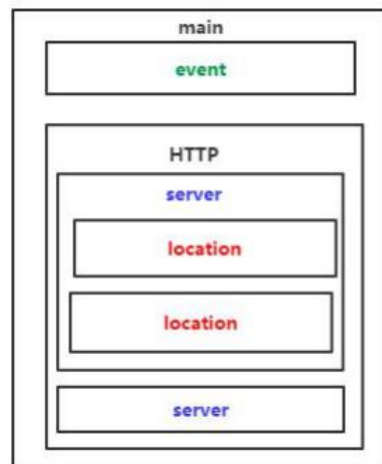
主进程主要完成如下工作：

1. 读取并验证配置信息；
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；

五、 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:用来定义全局错误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，而 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 I/O (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;
}
}

```

六、 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`

6.1. 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

2. `access_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];` # 设置访问日志
3. `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

```
4. http {
5.     include      mime.types;
6.     default_type  application/octet-stream;
7.
8.     log_format    main '$remote_addr - $remote_user [$time_local] "$request" '
9.                       '$status $body_bytes_sent "$http_referer" '
10.                      '"$http_user_agent" "$http_x_forwarded_for"';
11.     ##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。
12.     access_log    /var/logs/nginx-access.log buffer=32k gzip flush=1m;
13.     ...
14. }
```

作用域

access_log 指令的作用域分别有 http，server，location。

6.2. log_format 自定义格式

默认的日志格式

```
15. log_format    main '$remote_addr - $remote_user [$time_local] "$request" '
16.                '$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

6.3. 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

17. `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
```

6.4. 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/"
#备份文件名称
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
```

七、 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
```

八、 路由--Location 的使用

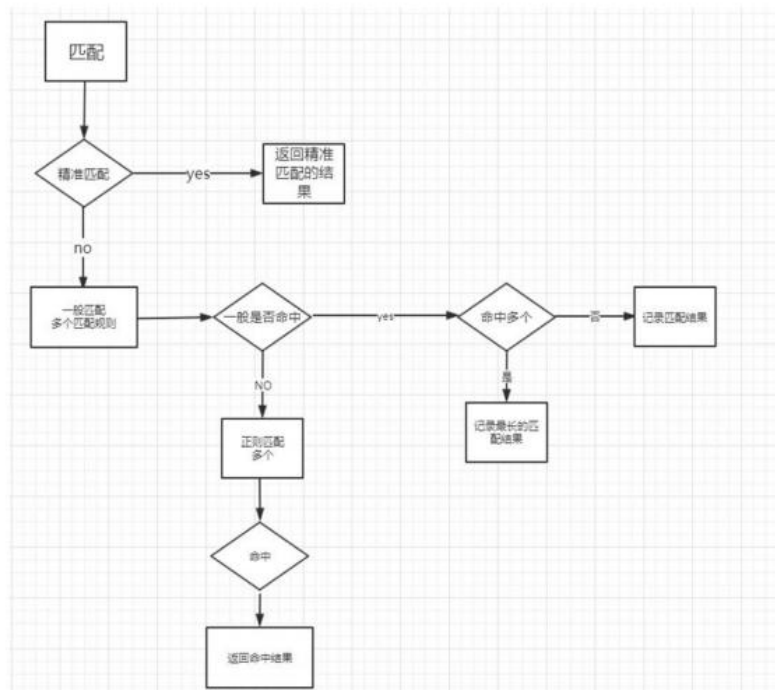
9.1. Location 语法规则

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。

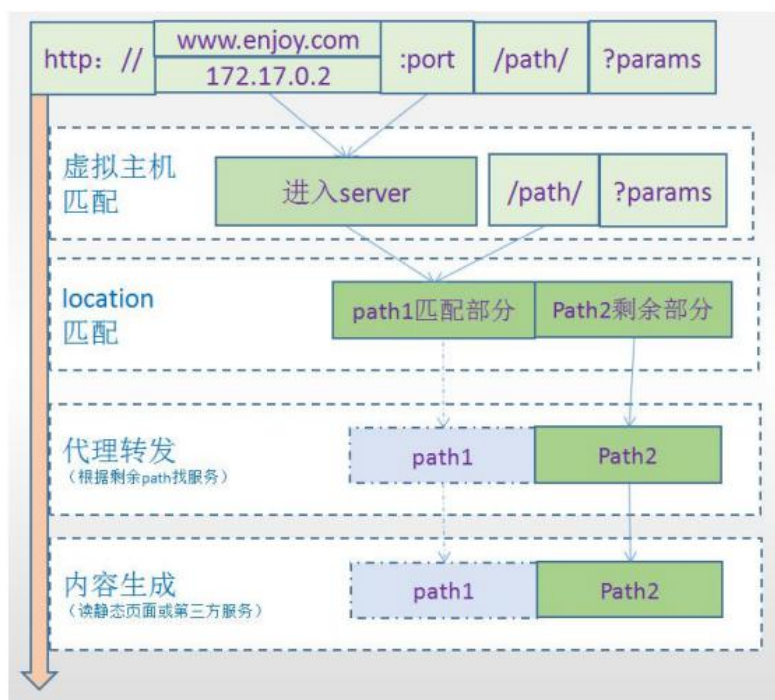
符号	含义
=	= 开头表示精确匹配
^^	^^开头表示 uri 以某个常规字符串开头，理解为匹配 url 路径即可（禁止正则匹配）。
~	~ 开头表示区分大小写的正则匹配
~*	~* 开头表示不区分大小写的正则匹配
!~和!~*	!~和!~*分别为区分大小写不匹配及不区分大小写不匹配的正则
/	用户所使用的代理（一般为浏览器）

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不是非正则，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

9.2. 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。

```
#无/, 访问路径: http://pxy.enjoy.com/mvc/index?id=2
location /mvc {
    #此处未关闭，传递整个路径/nginx/enjoy/getInfo到目标ip:port
    proxy_pass http://192.168.0.132:8088;
}
```

补过来

```
#有/, 访问路径: http://pxy.enjoy.com/nginx/mvc/index?id=2
location /nginx/mvc {#匹配路径/dynamic, 剩余路径/nginx/enjoy/getInfo
    proxy_pass http://192.168.0.132:8088/mvc;
}
```

剩余部分补过来

九、 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

9.1. 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

9.2. 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。

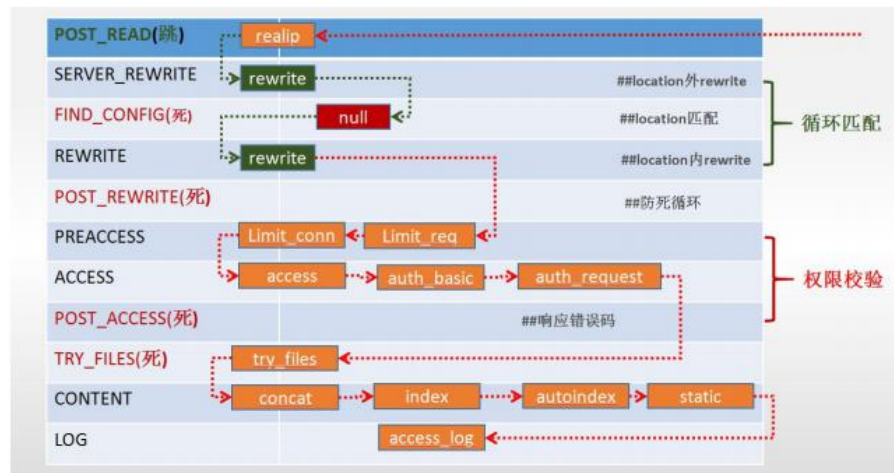
9.3. flag= redirect/permanent

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

9.4. 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

十、 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

十一、 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

11.1. 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。

11.2. 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

11.3. 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。

11.4. 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

```
location /order/enjoy {
    ##后台请求为: http://192.168.0.128:8383/enjoy/getPage
    ##调整后请求: http://test.enjoy.com/order/enjoy/getPage
    ##故代理需要关闭path1的传递
    proxy_pass http://order/enjoy;
}
```

十二、 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。

12.1. Openresty 的安装配置

12.1.1. 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
```

12.1.2. 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
## 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
```

12.1.3. 安装检测

```
nginx -V ##如下显示，则表示安装成功
```

```

[root@test nginx]# nginx -V
nginx version: openresty/1.15.8.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
built with OpenSSL 1.0.2k-fips 26 Jan 2017
TLS SNI support enabled
configure arguments: --prefix=/usr/local/openresty/nginx --with-cc-opt=-O2 --add-module=../ngx_devel_kit-0.3.1rc1 --add-module=../echo-nginx-module-0.61 --add-module=../xss-nginx-module-0.06 --add-module=../ngx_http_lua_module-0.10.15 --add-module=../set-misc-nginx-module-0.32 --add-module=../form-input-nginx-module-0.12 --add-module=../encrypted-session-nginx-module-0.08 --add-module=../srcache-nginx-module-0.31 --add-module=../ngx_lua_upstream-0.07 --add-module=../headers-more-nginx-module-0.33 --add-module=../array-var-nginx-module-0.05 --add-module=../memc-nginx-module-0.19 --add-module=../redis2-nginx-module-0.15 --add-module=../redis-nginx-module-0.3.7 --add-module=../rds-json-nginx-module-0.15 --add-module=../rds-csv-nginx-module-0.09 --add-module=../ngx_stream_lua-0.0.7 --with-ld-opt=-Wl,-rpath,/usr/local/openresty/luajit/lib --with-stream --with-stream_ssl_module --with-stream_ssl_preread_module --with-http_ssl_module

```

12.2. 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

12.3. 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

12.4. Lua 基础功能使用介绍

12.4.1. hello world

在 content 阶段，执行 lua 脚本，输出 hello, peter

```
location /hello {  
    ##ngx.say--输出内容print  
    content_by_lua 'ngx.say("Hello, Peter!")';  
}
```

在content阶段，去执行lua脚本

12.4.2. 执行 lua 脚本文件

```
location /args_read {  
    ##执行lua文件脚本  
    content_by_lua_file /etc/nginx/lua/lua_args.lua;  
}
```

脚本文件位置

12.4.3. 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：

```
location /args {  
    ##ngx.var--取请求参数，arg_a指参数a  
    content_by_lua_block {  
        ngx.say(ngx.var.arg_a)  
        ngx.say(ngx.var.arg_b)  
    }  
}
```

12.4.4. lua 取全量参数

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


```

--lua的注释
--key-value形式取得所有的url上的参数--get型参数
local arg = ngx.req.get_uri_args()
for k,v in pairs(arg) do
    ngx.say("[GET] ", k, " :", v)
end

--key-value形式取得所有post的参数
ngx.req.read_body()-- 解析 body 参数之前一定要先读取 body
local arg = ngx.req.get_post_args()
for k,v in pairs(arg) do
    ngx.say("[POST] ", k, " :", v)
end

```

读get和post参数

12.4.5. lua 取 request 中 header 信息

```

--读请求头信息
local headers = ngx.req.get_headers()
ngx.say("Host : ", headers.Host)
ngx.say("Host : ", headers["Host"])
ngx.say("-----")
for k,v in pairs(headers) do
    if type(v) == "table" then
        --table.concat是table操作, 意指将v内所有值合并
        ngx.say(k, " : ", table.concat(v, ","))
    else
        ngx.say(k, " : ", v)
    end
end
end

```

得到值合集
简单取值方式
遍历取值方式

12.4.6. 给 lua 脚本传参

使用端传参:

```

location /setfile {
    ##给lua脚本传递参数
    set_by_lua_file $val "/etc/nginx/lua/set.lua" $arg_a $arg_b;
    echo $val;
}

```

接收脚本返回结果
传入两个参数

脚本中借助 ngx.arg 取参

```

local a=tonumber(ngx.arg[1])
local b=tonumber(ngx.arg[2])
return a + b

```

取第一个参数
取第二个参数

12.4.7. 权限校验

一般校验动作，指定在 access 阶段执行脚本

```
location /access {  
    ##权限控制  
    access_by_lua_file "/etc/nginx/lua/access.lua";  
    echo "welcome $arg_name !";  
}
```

脚本处理

```
if ngx.var.arg_passwd == "123456"  
then  
    return  
else  
    ngx.exit(ngx.HTTP_FORBIDDEN)  
end
```

12.4.8. 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

```
location /filter {  
    echo 'hello Peter';  
    echo 'you are welcome!';  
    ##内容过滤  
    body_filter_by_lua_file "/etc/nginx/lua/filter.lua";  
}
```

脚本中的处理

```
--ngx.arg[1]是输出块内容  
local chunk = ngx.arg[1]  
if string.match(chunk, "hello") then  
    ngx.arg[2] = true -- 设置为true，表示输出结束 eof  
    return  
end
```

12.5. 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

12.5.1. Lua-resty-redis 连接 redis 用法

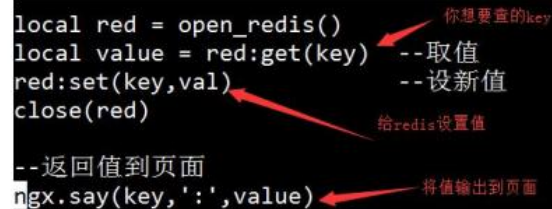
Lua-resty-redis 插件，对 Nginx 操作 redis 的支持十分强大，成熟的用法演示如下：
基础的引入、连接动作

```
local redis = require "resty.redis"
--打开redis连接
local function open_redis()
    local red = redis:new()
    red:set_timeout(1000) -- 超时时间1 second
    local res = red:connect('192.168.0.128',6379)
    if not res then
        return nil
    end
    res = red:auth(123456) --密码校验
    if not res then
        return nil
    end
    red.close = close
    return red
end
```



redis 操作动作

```
local red = open_redis()
local value = red:get(key)
red:set(key,val)
close(red)
--返回值到页面
ngx.say(key,':',value)
```



具体全量的程序，见源码配置包

12.5.2. Lua-resty-mysql 连接 mysql 数据库

引入模块、创建连接

```

local mysql = require "resty.mysql"
local cJSON = require "cjson"

--配置
local config = {
    host = "192.168.0.128",
    port = 3303,
    database = "enjoy",
    user = "root",
    password = "root"
}

--打开连接
local function open_mysql()
    local db, err = mysql:new()
    if not db then
        return nil
    end
    db:set_timeout(1000) -- 1 sec

    local ok, err, errno, sqlstate = db:connect(config)

    if not ok then
        return nil
    end
    db.close = close
    return db
end

```

引入模块

数据库配置，方便整体传入

创建连接

mysql 查询操作

```

local db = open_mysql()
local sql = "select * from t_account "
--设置中文编码
ngx.header['Content-Type']="text/html;charset=UTF-8"

local res, err, errno, sqlstate = db:query(sql)
close(db)
if not res then
    ngx.say(err)
    return {}
end

--json方式输出
ngx.say(cjson.encode(res))

```

查询sql

查询得到结果集

将结果集转换为json串，输出到页面

十三、 小功能合集

13.1. 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。

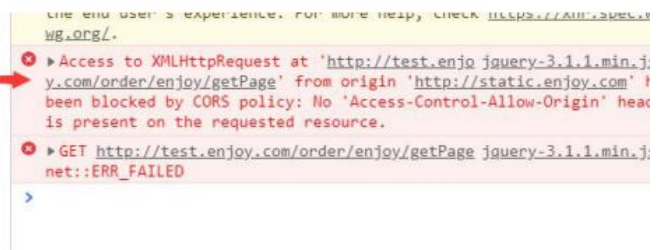
为了防止这种混乱，W3C 组织制定了浏览器安全规范，即 html 页面发起的 ajax 请求仅限于同域名后端范围，跨越域名的 ajax 请求不得执行，此谓跨域问题。

如下图：

```
//当前页面域名 static.enjoy.com
$.ajax({
  url:"http://test.enjoy.com/order/enjoy/getPage",
  type:"GET",
  //url:"http://test.enjoy.com/order/enjoy/delOrder",
  //type:"DELETE",
  async :false,
  success:function(data){
    $("#testcors").html(data);
  }
});
```

当前页面域名与ajax的目标域名不一致，浏览器将拒绝执行请求

浏览器控制台
将发出这样的错误



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：

```

<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script type="text/javascript">
$.ajax({
  url:"http://test.enjoy.com/order/enjoy/getPage",
  type:"GET",
  //url:"http://test.enjoy.com/order/enjoy/delOrder",
  //type:"DELETE",
  async :false,
  success:function(data){
    $("#testcors").html(data);
  }
});

```

在这里，跨域是允许的

在这里，跨域不允许

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

```

#是否允许请求带有验证信息
add_header Access-Control-Allow-Credentials true;
#允许跨域访问的域名,可以是一个域的列表,也可以是通配符*
#add_header Access-Control-Allow-Origin http://static.enjoy.com;
#允许脚本访问的返回头
add_header Access-Control-Allow-Headers 'x-requested-with,content-type,Cache-Control,Pragm
amp';
#允许使用的请求方法,以逗号隔开
add_header Access-Control-Allow-Methods 'POST,GET,OPTIONS,PUT,DELETE';
#允许自定义的头部,以逗号隔开,大小写不敏感
add_header Access-Control-Expose-Headers 'WWW-Authenticate,Server-Authorization';
#P3P支持跨域cookie操作
add_header P3P 'policyref="/w3c/p3p.xml", CP="NOI DSP PSaA OUR BUS IND ONL UNI COM NAV INT

```

我声明,static域名下的网页,可以

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

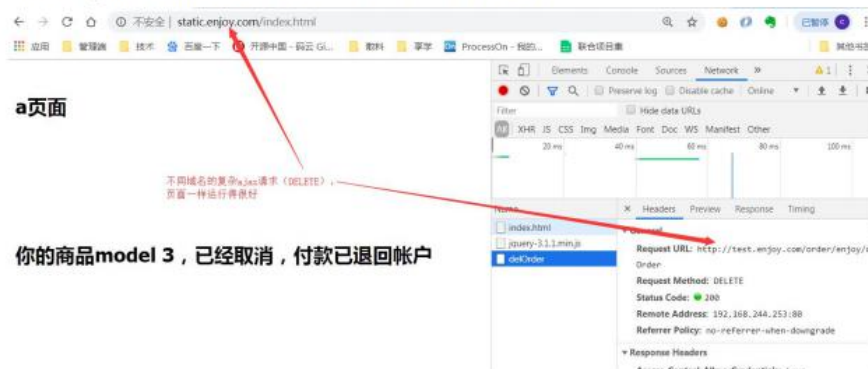
```

if ($request_method = 'OPTIONS') {
    return 204;
}

```

yes, 我愿意与你同吃同睡,有福同享

有了上述 Nginx 的两道配置,跨域问题自然和解,对业务毫无侵入性。



不同域名的静态资源 (DELETE)
页面一样运行得很好

你的商品model 3,已经取消,付款已退回帐户

13.2. 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

```

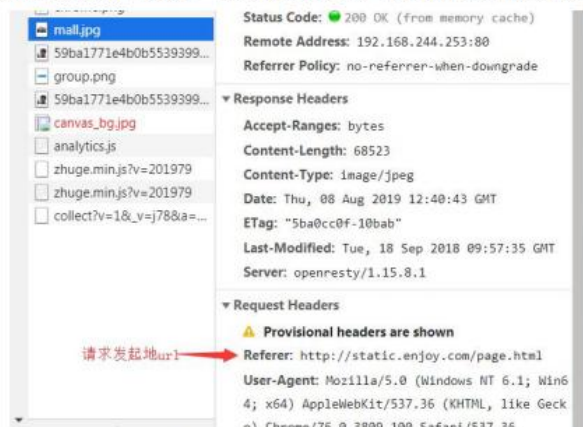
<img src='http://static.enjoy.com/qq.png' />
<img src='http://static.enjoy.com/chrome.png' />
<img src='http://static.enjoy.com/mall.jpg' />

```

老王在
让我的

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

```
location ^~ /mall {
    valid_referers *.enjoy.com;##对referer进行校验
    if ($invalid_referer) {
        return 404;
    }
    root /etc/nginx/html/gzip;
}
```

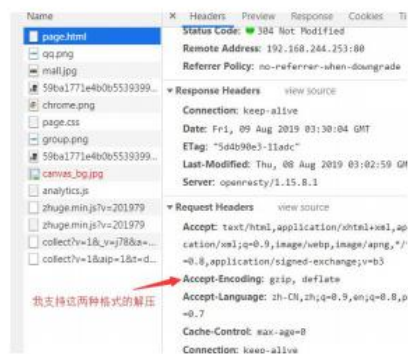
对发起地的url进行校验, 校验不过, 告诉它我没这

13.3. 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

```
location ~ /\.(\.html|js|css|jpg|jpeg|png|gif)$ {#覆盖/re/a.htm路径
    gzip on; # 启用gzip压缩. 默认是off. 不启用

    # 对js、css、jpg、png、gif格式的文件启用gzip压缩功能
    gzip_types application/javascript text/css image/jpeg image/png image/gif;
    gzip_min_length 1024; # 所压缩文件的最小值, 小于这个的不会压缩
    gzip_buffers 4 1k; # 设置压缩响应的缓冲块的大小和个数, 默认是内存一个页的大小
    gzip_comp_level 1; # 压缩水平, 默认1. 取值范围1-9, 取值越大压缩比率越大, 但越耗cpu时间

    root html/gzip;
}
```

十四、https 配置

14.1. 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

14.2. 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）。

14.3. 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



14.4. 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）：

```
[root@test nginx]# nginx -V
nginx version: openresty/1.15.8.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
built with OpenSSL 1.0.2k-fips 26 Jan 2017
TLS SNI support enabled
configure arguments: --prefix=/usr/local/openresty/nginx --w
.3.1rc1 --add-module=../echo-nginx-module-0.61 --add-module=
oolkit-0.2 --add-module=../set-misc-nginx-module-0.32 --add-
odule=../encrypted-session-nginx-module-0.08 --add-module=..
_lua-0.10.15 --add-module=../ngx_lua_upstream-0.07 --add-mod
dule=../array-var-nginx-module-0.05 --add-module=../memc-ngi
dule-0.15 --add-module=../redis-nginx-module-0.3.7 --add-mod
=../rds-csv-nginx-module-0.09 --add-module=../ngx_stream_lua
nresty/luajit/lib --with-stream --with-stream_ssl_module --w
module
```

此命令查看

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）：

```
使用openssl生成证书
生成私钥: openssl genrsa -des3 -out server.key 4096 --密码1234
生成CSR: openssl req -new -key server.key -out server.csr
去除私钥口令 cp server.key server.key.org
openssl rsa -in server.key.org -out server.key --密码1234
生成证书 openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Nginx 配置

Nginx 内的配置如下：

```
server {  
    listen      443 ssl;   
    server_name enjoy.com;  
  
    ssl_certificate      /etc/nginx/server.crt; ##供浏览器下载证书  
    ssl_certificate_key  /etc/nginx/server.key; ##自己用来解密的私钥  
  
    location / {  
        root   /etc/nginx/html;  
        index  index.html index.htm;  
    }  
}
```

ssl端口

注意文件对应的路径

校验

输入网址: <https://enjoy.com/a.html>

https 方式显示页面如下:



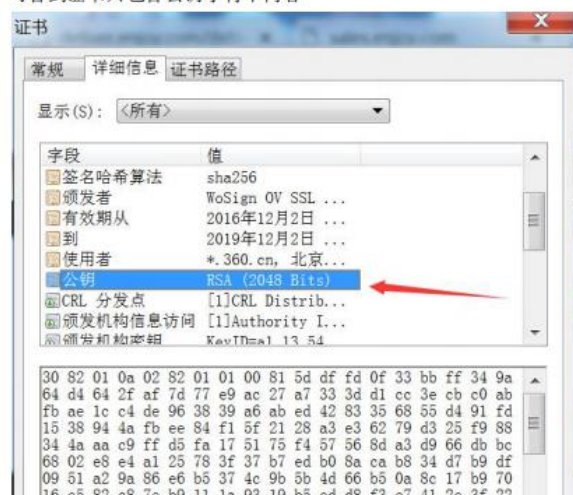
a页面

自签证书, 未认证, 因此浏览器发出一个警告

查看证书



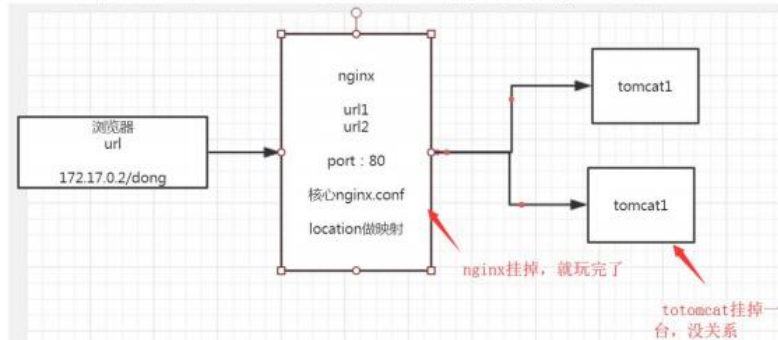
可看到证书只包含公钥字符串内容



十五、 nginx 高可用

15.1. 传统的高可用思路

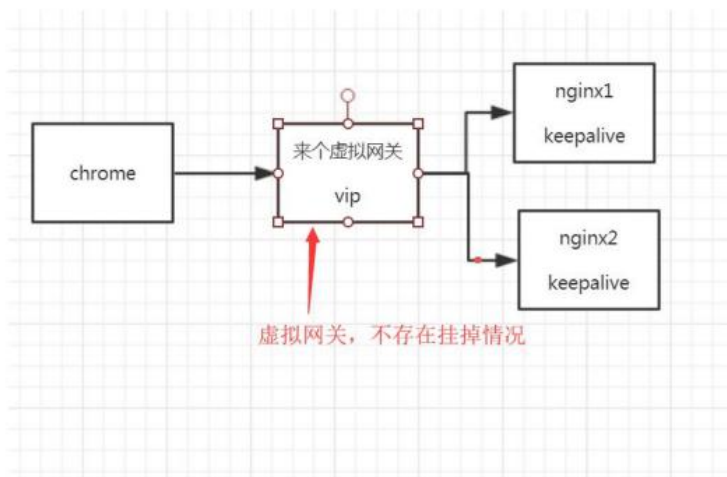
tomcat 的高可用的思路，是在 tomcat 集群前面加一层负载服务 nginx。如下图



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）

15.2. 1vs 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

15.3. keepalived 配置 LVS 过程

前提

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

```
[root@test nginx]# vi /etc/sysconfig/selinux
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#   enforcing - SELinux security policy is enforced.
#   permissive - SELinux prints warnings instead of enforcing.
#   disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three values:
```

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）


```

! Configuration File for keepalived

global_defs {
    router_id LVS_2      ##keepalived的唯一标识
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33      ##系统网上名,可以使用ip addr命令查看
    virtual_router_id 51  ##组名,参与此虚拟ip的机器配置一样的值
    priority 200          ##优先级,数值大的优先级高,组内最高的胜出
    advert_int 1          ##心跳检测1s一次
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.244.200 ##虚拟的ip
    }
}

```

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

```

[root@test keepalived]# /usr/local/keepalived/sbin/keepalived
[root@test keepalived]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc p
    link/ether 00:0c:29:a3:a0:16 brd ff:ff:ff:ff:ff:ff
    inet 192.168.244.253/24 brd 192.168.244.255 scope global
        valid_lft forever preferred_lft forever
    inet 192.168.244.200/32 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fea3:a016/64 scope link
        valid_lft forever preferred_lft forever

```

此时, 使用原 ip 地址 244.253 能打开的页面, 使用 244.200 也能打开



a页面

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
! Configuration File for keepalived

```
global_defs {
    router_id LVS_3  ##keepalived的唯一标识
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33  ##系统网上名,可以使用ip addr命令查看
    virtual_router_id 51  ##组名,参与此虚拟ip的机器配置一样的值
    priority 100  ##优先级,数值大的优先级高,组内最高的胜出
    advert_int 1  ##心跳检测1s一次
    authentication {
        auth_type PASS  ##授权,无须改动
        auth_pass 1111  优先级100,不能与主机200一样
    }
    virtual_ipaddress {
        192.168.244.200  ##虚拟的ip
    }
}
```

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：

```
global_defs {
    router_id LVS_1
}
vrrp_script chk_http_port {
    script "/usr/local/src/chk_nginx_pid.sh" #心跳执行的脚本
    interval 2                               #（检测脚本执行的间隔，单位是秒）
    weight 2
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0 #系统网卡
    virtual_router_id 51 #主备两机器一致
    priority 100 #值大的机器，胜出
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        chk_http_port #（调用检测脚本）
    }
    virtual_ipaddress { #可虚拟多个ip
        192.168.244.200
    }
}
```

新加

c、重启 keepalived，测试监控效果，如下图操作：

```
[root@node1 ~]# keepalived
[root@node1 ~]# /usr/local/nginx/sbin/nginx -s stop
[root@node1 ~]# ps -ef|grep nginx
root      60372      1  0 05:45 ?        00:00:00 nginx: master process /usr/local/nginx/sbin/nginx
nobody    60376  60372  0 05:45 ?        00:00:00 nginx: worker process
root      60408  1373  0 05:45 pts/0    00:00:00 grep nginx
[root@node1 ~]# /usr/local/nginx/sbin/nginx -s stop
[root@node1 ~]# /usr/local/nginx/sbin/nginx -s stop
[root@node1 ~]# /usr/local/nginx/sbin/nginx -s stop
nginx: [error] open() "/usr/local/nginx/logs/nginx.pid" failed (2: No such file or directory)
[root@node1 ~]# ps -ef|grep nginx
root      60469      1  0 05:45 ?        00:00:00 nginx: master process /usr/local/nginx/sbin/nginx
nobody    60473  60469  0 05:45 ?        00:00:00 nginx: worker process
root      60485  1373  0 05:45 pts/0    00:00:00 grep nginx
[root@node1 ~]#
```

nginx 已变成不死鸟

十六、 Nginx 在 mvvm 模式中的使用：

