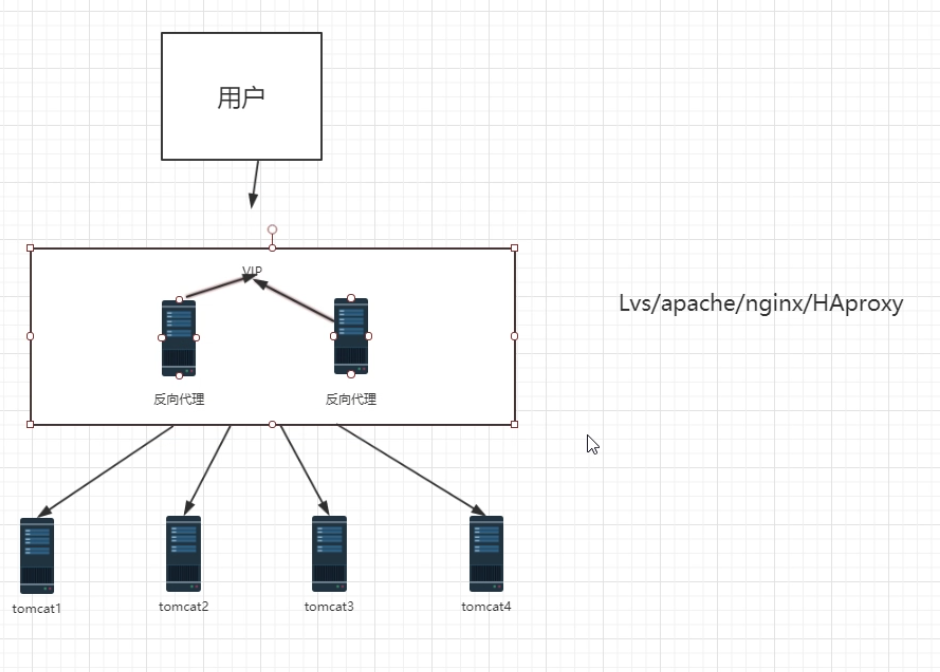
# 高性能反向代理服务器-Nginx第二次课程

授课老师：Mic

# 本次课程内容

1. Rewrite的使用
2. 缓存配置及Gzip配置
3. Nginx反向代理实战
4. 负载均衡配置说明
5. Nginx的进程模型
6. Nginx产线配置
7. Nginx+keepalived实现高可用



vip漂移 nginx中只有一个提供服务 另一个提供热备，让我想起了redis

# 课程笔记

## rewrite的使用

Rewrite通过ngx\_http\_rewrite\_module模块支持url重写、支持if判断，但不支持else

rewrite功能就是，使用nginx提供的全局变量或自己设置的变量，结合正则表达式和标志位实现url重写以及重定向

rewrite只能放在server{},location{},if{}中，并且只能对域名后边的除去传递的参数外的字符串起作用

常用指令

**If 空格 (条件) {设定条件进行重写}**

条件的语法：

1. “=” 来判断相等，用于字符比较
2. “~” 用正则来匹配（表示区分大小写），“~\*” 不区分大小写
3. “-f -d -e” 来判断是否为文件、目录、是否存在

**return 指令**

语法：return code;

停止处理并返回指定状态码给客户端。

if ($request\_uri ~ \*\.sh ){

return 403

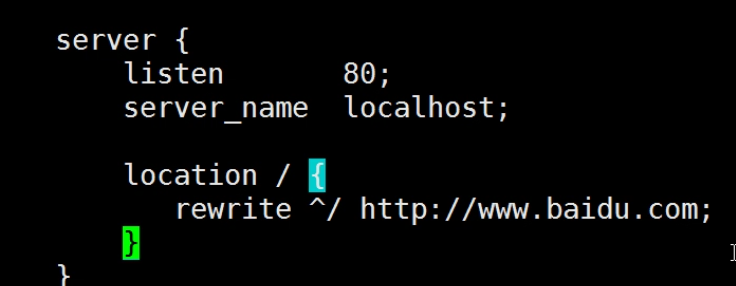
}

**set指令**

set variable value;

定义一个变量并复制，值可以是文本、变量或者文本变量混合体

**rewrite指令**

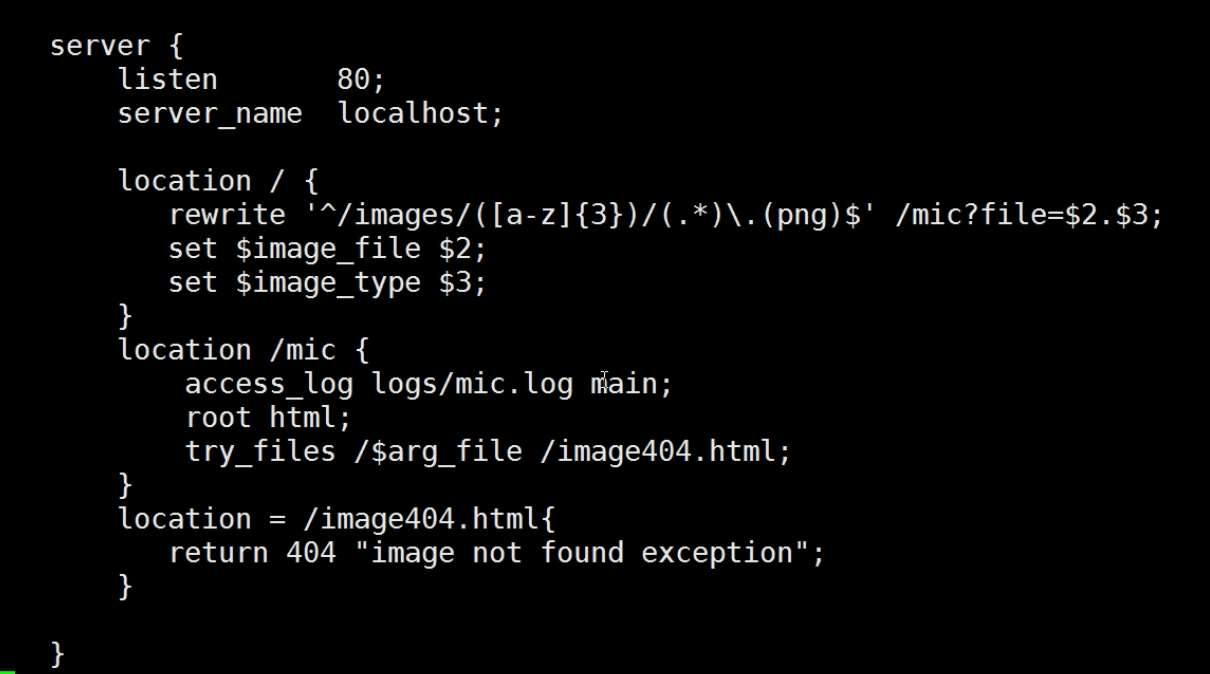
****

语法：rewrite regex replacement [flag]{last / break/ redirect 返回临时302/ permant 返回永久302}

last: 停止处理后续的rewrite指令集、 然后对当前重写的uri在rewrite指令集上重新查找

break; 停止处理后续的rewrite指令集 ,并不会重新查找

综合实例



如上配置对于： /images/ttt/test.png 会重写到/mic?file=test.png, 于是匹配到 location /mic ; 通过try\_files获取存在的文件进行返回。最后由于文件不存在所以直接

返回404错误

**rewrite匹配规则**

表面看rewrite和location功能有点像，都能实现跳转，主要区别在于rewrite是在同一域名内更改获取资源的路径，而location是对一类路径做控制访问或反向代理，可以proxy\_pass到其他机器。很多情况下rewrite也会写在location里，它们的执行顺序是：

* 执行server块的rewrite指令
* 执行location匹配
* 执行选定的location中的rewrite指令

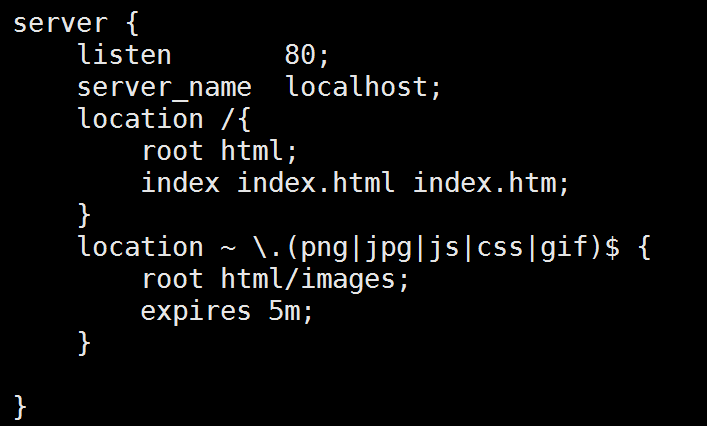
如果其中某步URI被重写，则重新循环执行1-3，直到找到真实存在的文件；循环超过10次，则返回500 Internal Server Error错误

## 浏览器本地缓存配置及动静分离

语法： expires 60s|m|h|d

操作步骤

* 在html目录下创建一个images文件，在该文件中放一张图片
* 修改index.html, 增加<img src=”图片”/>
* 修改nginx.conf配置。配置两个location实现动静分离，并且在静态文件中增加expires的缓存期限



## Gzip压缩策略

浏览器请求 -> 告诉服务端当前浏览器可以支持压缩类型->服务端会把内容根据浏览器所支持的压缩策略去进行压缩返回

->浏览器拿到数据以后解码； 常见的压缩方式：gzip（好像图片不失真）、deflate 、sdch

|  |  |
| --- | --- |
|  | Gzip on|off 是否开启gzip压缩  Gzip\_buffers 4 16k #设置系统获取几个单位的缓存用于存储gzip的压缩结果数据流。4 16k代表以16k为单位，安装原始数据大小以16k为单位的4倍申请内存。  Gzip\_comp\_level[1-9] 压缩级别， 级别越高，压缩越小，但是会占用CPU资源(压缩涉及计算)  Gzip\_disable #正则匹配UA 表示什么样的浏览器不进行gzip  Gzip\_min\_length #开始压缩的最小长度（小于多少就不做压缩）  Gzip\_http\_version 1.0|1.1 表示开始压缩的http协议版本  Gzip\_proxied （nginx 做前端代理时启用该选项，表示无论后端服务器的headers头返回什么信息，都无条件启用压缩）  Gzip\_type text/pliain,application/xml 对那些类型的文件做压缩 （conf/mime.conf）  Gzip\_vary on|off 是否传输gzip压缩标识 |

**注意点**

1. 图片、mp3这样的二进制文件，没必要做压缩处理，因为这类文件压缩比很小，压缩过程会耗费CPU资源
2. 太小的文件没必要压缩，因为压缩以后会增加一些头信息，反而导致文件变大
3. Nginx默认只对text/html进行压缩 ，如果要对html之外的内容进行压缩传输，我们需要手动来配置

## Nginx反向代理

一般都把文件配置出去 然后引用include



Proxy\_pass

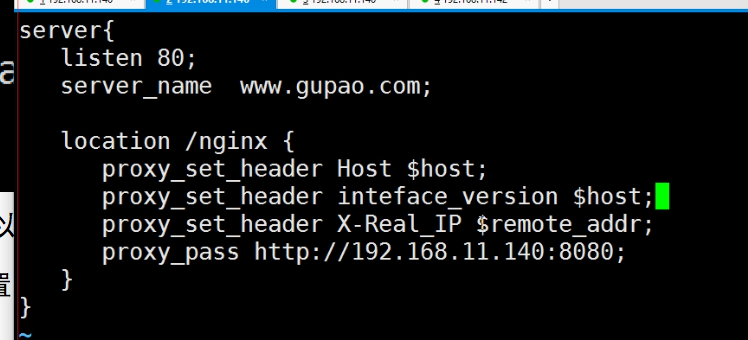
通过反向代理把请求转发到百度



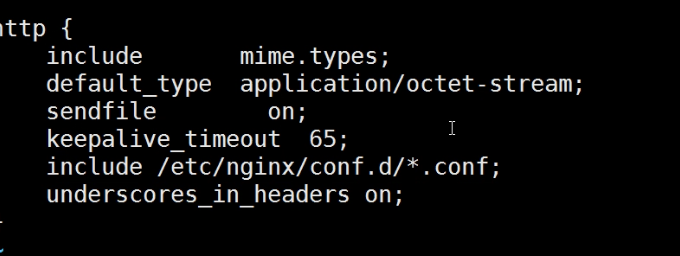
Proxy\_pass 既可以是ip地址，也可以是域名，同时还可以指定端口

Proxy\_pass指定的地址携带了URI，看我们前面的配置【/s】，那么这里的URI将会替换请求URI中匹配location参数部分；如上代码将会访问到http://www.baidu.com/s

interface\_version 需要下划线支持(现在版本是可以的)



之前下划线支持是要加underscore



## 负载均衡

*upstream*是Nginx的HTTP Upstream模块，这个模块通过一个简单的调度算法来实现客户端IP到后端服务器的负载均衡

### Upstream常用参数介绍

**语法：**server address [parameters]

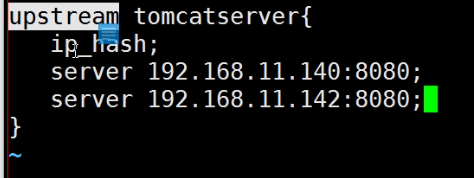
其中关键字server必选。  
address也必选，可以是主机名、域名、ip或unix socket，也可以指定端口号。  
parameters是可选参数，可以是如下参数：

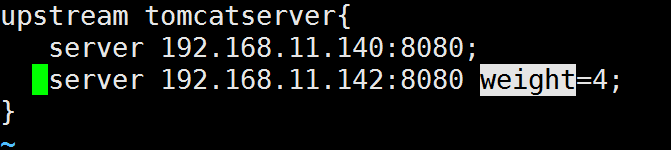
**down**：表示当前server已停用

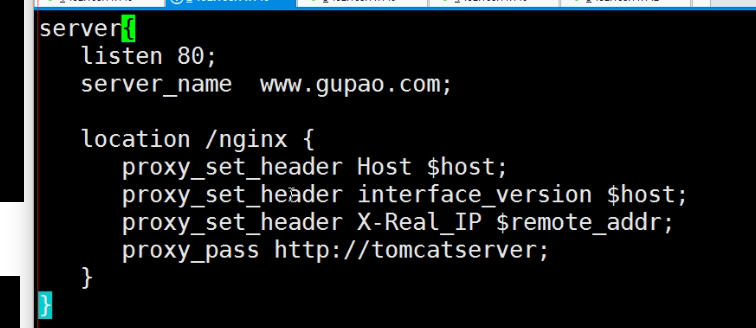
**backup**：表示当前server是备用服务器，只有其它非backup后端服务器都挂掉了或者很忙才会分配到请求

**weight**：表示当前server负载权重，权重越大被请求几率越大。默认是1

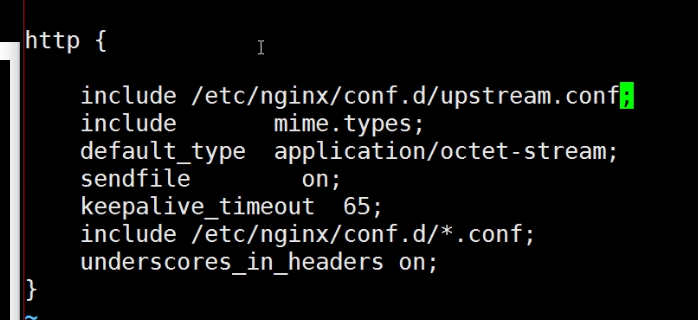
**max\_fails**和**fail\_timeout**一般会关联使用，如果某台server在fail\_timeout时间内出现了max\_fails次连接失败，那么Nginx会认为其已经挂掉了，从而在fail\_timeout时间内不再去请求它，fail\_timeout默认是10s，max\_fails默认是1，即默认情况是只要发生错误就认为服务器挂掉了，如果将max\_fails设置为0，则表示取消这项检查。







应该是放http段里面（把第一个include删除 因为下面include已经是\*.conf了）



ups支持的调度算法

ip\_hash 根据ip的hash值来做转发

默认是轮询机制

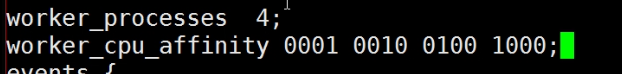
权重 weight=x

fair 根据服务器的响应时间来分配请求

url\_hash

# Q&A

Nginx默认没有利用cpu的多核优势 得配置



理论上5w的并发 也就是5w的连接数