nginx缓存 URL重写 和 读写分离

nginx的缓存功能

nginx有效减轻后端服务器压力 应该使用缓存

nginx中 缓存有两部分组成

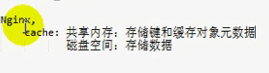
一部分是 有一段共享内存 ---- 内存区域主要用于存储查找的键 以及元数据

----查找在内存中完成 【共享内存 用来存放key】

另一部分是 磁盘空间 ---- 数据是存储在磁盘空间中

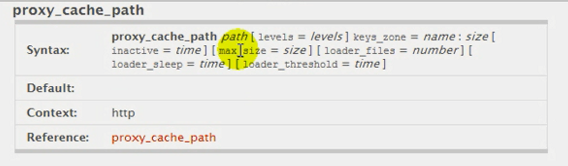
nginx仍然需要磁盘存数据 所以 使用固态硬盘很好

----- 对于缓存 磁盘IO的性能是至关重要的 【磁盘空间 用来存储value 因为value大 很多object】



由此 要使用缓存

要定义缓存空间 要使用、不能定义在server上下文



proxy\_cache\_path

 ---- levels 用于定义缓存目录的子目录级别的

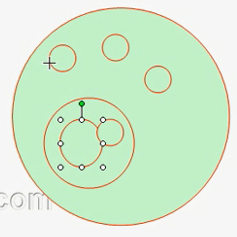
在缓存中存储数据 快速找一个数据

怎么找？按照某种过虑 存放

要缓存1000w 每一个要缓存100w

再分一个级别 每一个一级子目录 有10个二级子目录

很显然

 -----

每一个缓存对象都要命名 每一个对象都由自己的名 1000w的个名字 1000w次变化

一个目录缓存所有对象 命名很麻烦

二级子目录也复杂 10个二级子目录 每一个100w



 ---- 这个:分隔了几个数字 就表示是几级子目录

每一个数字的大小 表示这一级子目录的名称由几个字符组成

2:2 ---- 一级子目录和二级子目录都有两个字符

------ 最多有三级 每一级最多有2个字符

 三级目录

 name 每一个缓存都是共享内存组成 用来存储键的区域



 第一个内存区域的名字叫first  size大小是20M

这样 我的nginx的缓存 配置了磁盘路径

缓存目录级别是 

对应的内存区域 

我们在磁盘上 申请多大空间缓存对象？

缓存目录也不应该太大 可以定义一个大小  ----- 如果定义的大小太大 查找对象会特别慢

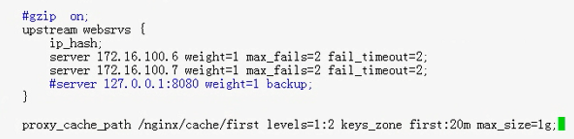


如果现在存储满了 但是 还有新的内容要缓存 所以 需要cache\_manager进程 根据最近最少使用算法 ----- cacahe\_manager就是之前除了worker master两个进程以外的进程

---- 所以 如果缓存的大小过大 查找麻烦 如果过小 缓存命中率会下降

---- 多大好 需要自己测试 ----- 只要提升 就让他增大

在server之外定义

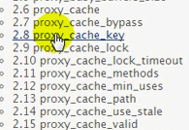


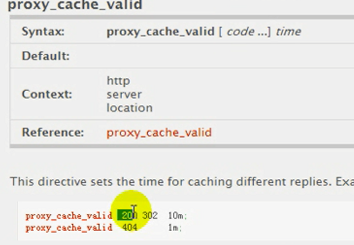
这个是有缓存功能 谁使用缓存功能 这个要在Location中指定

 ---- 明确说明不使用缓存

 ---- 使用 要指定要使用哪一个缓存

和cache相关的指令很多

 ----





301 302都是重定向

any表示任意请求结果都缓存

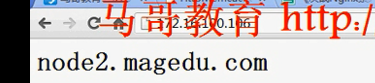
对于202的对象



创建目录



缓存功能就启用了

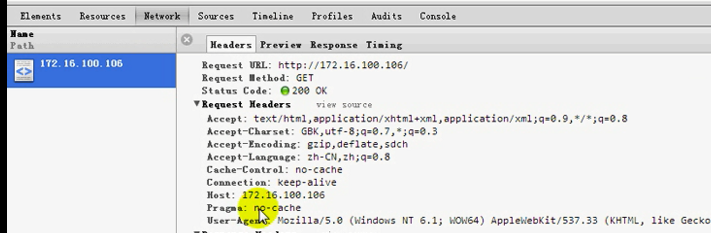


这样 没有负载均衡 因为都从缓存中走



Ctrl+F5 表示不使用浏览器自身的缓存 强行刷新

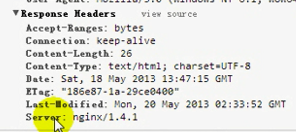
打开



 这个是no-cache

这个表示发送请求的时候 浏览器本身不使用缓存

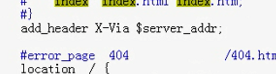
再看响应



Nginx自己返回

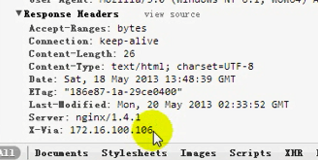
有一个指令 叫add\_header

加一个首部

重新修改对应的配置

这样每一次请求 都会添加一个含有X\_Via的报文

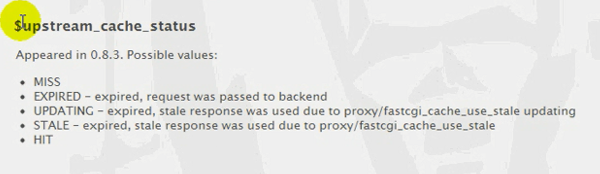
此时 crtl+f5

 这个有一个X-Via

到底缓存是否命中 仍然不知道

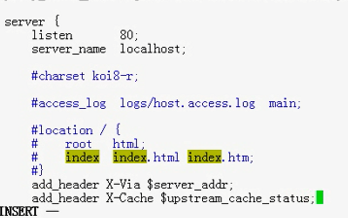
再加一个首部

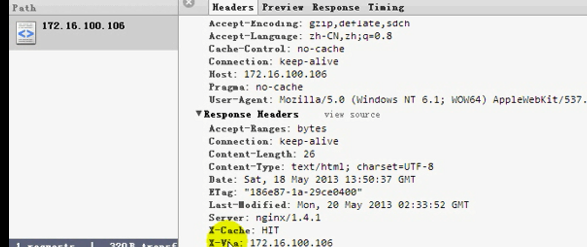
在upstream中有一个特殊的变量



看一下cache的状态

再加一个特殊的首部



说明缓存中有内容

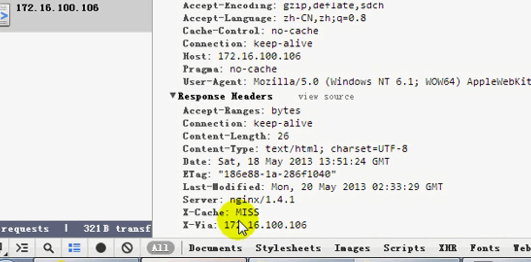
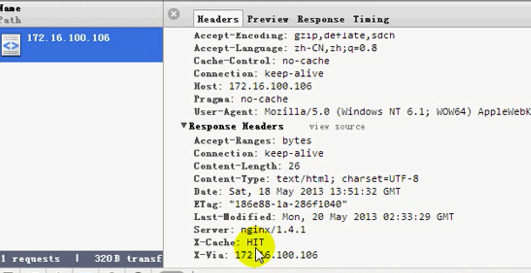
nginx停止掉

删除缓存的内容 

【使用磁盘IO做缓存 nginxIO模型使用的是AIO 所以 能力强】



删除 再start

重新修改配置文件

 --- 自定义字符串



当然 需要注意的是 nginx提供的缓存功能 ---- nginx每次都要写日志 ---为了快速存日志 ---- 为日志提供了缓冲区 ---- 也是一种缓存

为fastcgi也能提供缓存功能 ---不再奥数

------- 缓存是分级的

Nginx提供的缓存对所有用户都生效 ---- 共用cache

浏览器提供的缓存仅仅对当前浏览器生效 ----- 私有cache



Open\_log\_cache ---打开日志缓存 就是先把日志缓冲到内存中 然后到一定大小 再存放到磁盘中 ----降低磁盘IO

Open\_file\_cache ---web服务器 有很多文件要打开 所以 这个地方把文件

Fastcgi\_cacahe ---- php处理的结果存放到本地

需要注意的是 缓存要求有足够内存 很多场景下 打开文件缓存 和 日志缓存 会提高nginx的效率

Nginx在默认编译 就提供压缩功能



有很多指令



并非所有情况都要压缩

尤其是 要缓存的时候

还可以定义压缩比



压缩本身就要用到一些内存空间的 所以 有一个gzip\_buffers用来为压缩提供内存



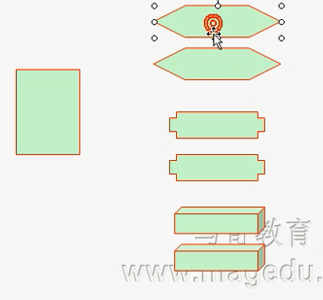
无论如何 启用压缩功能

 启动就可以了

如果对应的用户请求的动态服务特别多 --- 可能导致一台服务器无法响应 ----

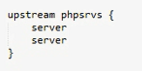
还有好几台静态服务器 专门用来提供图片服务的

还有一组新的服务器 用来存储用户上传的服务器



用户请求是php格式的内容 在最上面的web appserver代理

定义upstreams





Php也可以使用upstream

定义图片服务器组



其他资源怎办 可以定义一个最大的匹配



这样 动态的 静态的 +本地的都部署了

如果是静态资源 定义 本地都反向代理到一个staticfilesrvs

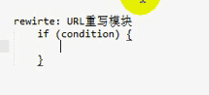


因此可以定义多个upstream组

另外 除了使用这种方式 来定义用户的资源模块

Rewrite

还有一个 url rewrie模块 ----- 有一个重要的指令 就是if指令



If通常使用在server和location中

条件该怎么写？

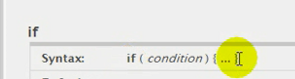
条件的匹配和location的匹配 也是类似的

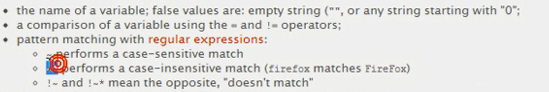
有一个测试 双目测试 和 单目测试

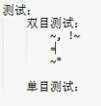
双目测试：~ = ~\*等等

找一下rewrite模块









如果nginx有很多内置变量

测试 请求报文是不是post方法

 如果你请求的uri中包含了/forum的意思

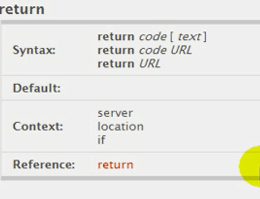
判断条件通常是为了完成url重写

还有一个就是break



Break就是退出当前判断

还有一个return 直接返回一个状态信息给客户端 一般是0-999



比如referrer http中 referrer 是当前页面从哪里过来

经常 对于我们的资源是盗链 避免盗链 如果是盗链 不让访问

还有一个模块 Referrer



定义哪些地方的引用是合法的引用

 ---- 从浏览器中输入的叫none 没有referer

 ---

 不能够被指定的referer匹配到 都是不合法的

403表示访问资源不存在

因此 我们的非法盗链不让访问

当别人非法盗链的时候 跳转到我们的服务器上面 打开主页

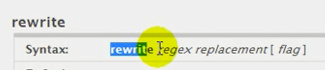
可以使用rewrite 或者 错误的图片给一个统一的提示

===== 如何使用rewrite指令 来完成url重写 ====

本来你访问的是一个路径 但是 给你访问的是另一个路径



完成把一个url替换成另外一个url

但是 rewrite支持正则表达式 就是把一类url转换成另一类的url  


把符合条件的url统统替换

或者 使用 rewrite 正则表达式 replacement



你的uri以download开头的/download/任意字符/media/任意字符.任意字符 都重定向到 上面的资源上



加入现在有一个location



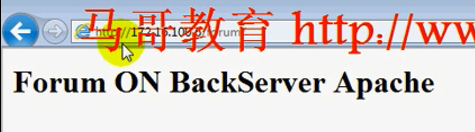
期望如果访问bbs路径 就访问到另一台服务器



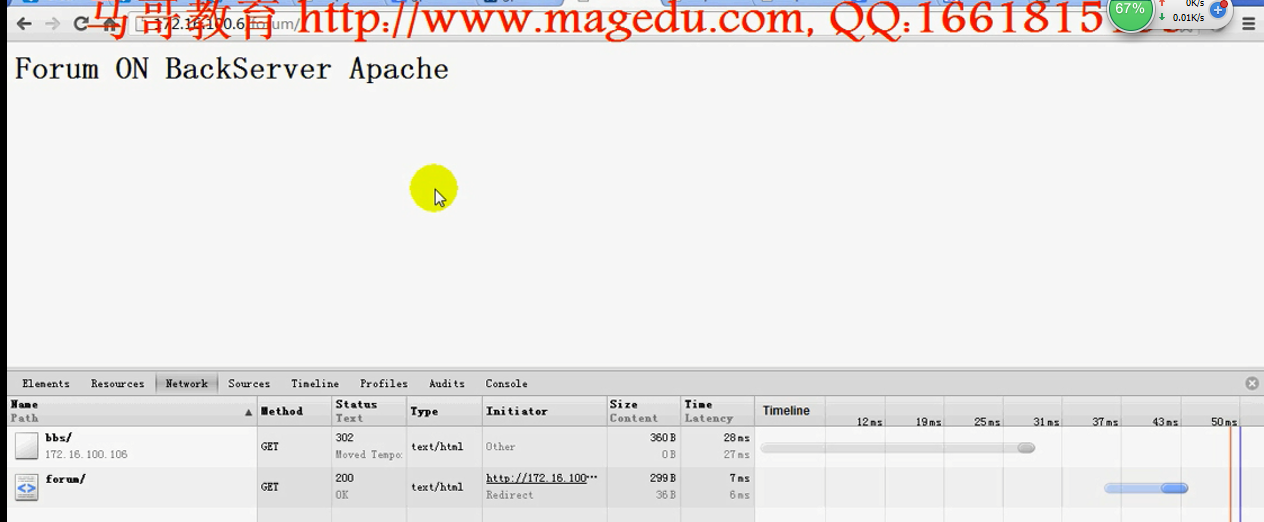
 ----- $1的原因就是跟随的参数都是不变的

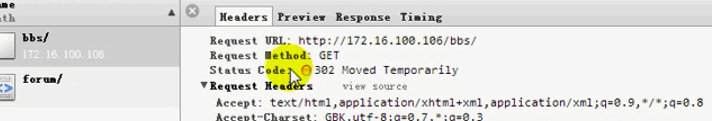
Rewrite很容易造成循环引用



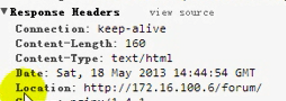


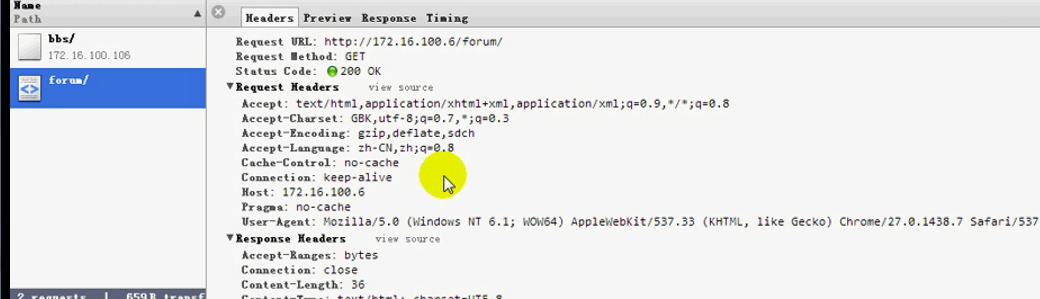
发现使用chrome可以使用F12分析执行过程





302的意思 就是临时重定向 资源挪到哪里了？响应首部 给出一个Location





301是永久重定向 302是临时重定向

自己主机内部实现的都是永久重定向

主机之间通畅是临时重定向



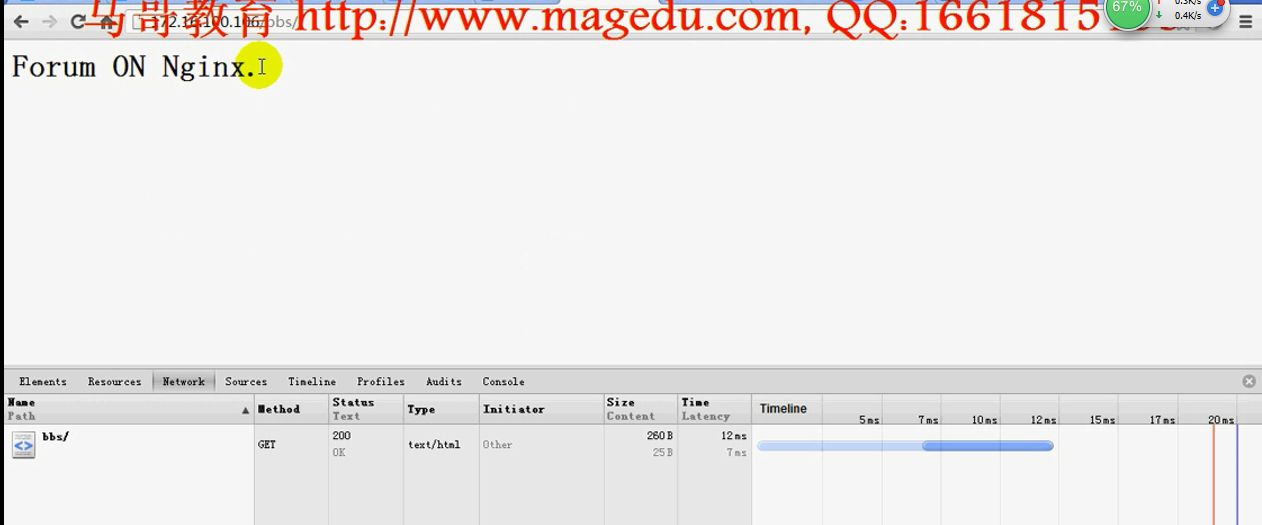
没有bbs目录

但是 假设之前都是使用bbs进行访问 所以 你这目录一更改 就会导致 你的访问路径要更改

但是 你的用户使用的都是bbs 所以 这样会造成不便

此时 重写







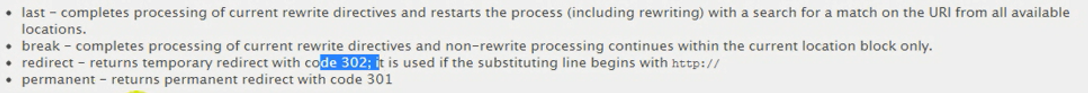
这个就直接定向到forum 隐式重定向

nginx各指令中 重定向的时候

【我现在理解的是 如果是本地转发 location 符号 uri {里面的uri是这个外面的uri的父级磁盘目录} ----- 因为 nginx中的uri即是uri 又是对应的磁盘路径 但是rewrite就可以随便写

如果是远程转发 区别不大】

 flag是四种标志



一般情况下 常用的都是last

last ---- 本次重写完成之后 重启下一轮检查

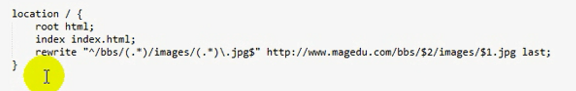


使用<http://www.magedu.com/bbs/index.html> ->重写为



重定向的时候 会对发起第二次请求 这时候 还是要经过nginx的 注意 这个请求 还是能够符合  所以 再次匹配 仍然是rewrite 再次被检查 不满足rewrite的条件

有一种特殊的场景



访问的是本机



替换后的内容可以被rewrite匹配到 这样就又重写了 这样有换成原来的 这样 就是循环

遇到这种情况 会来回循环10次 就会停止循环

这种情况 只能使用第二种break



这两个少

大部分使用last 出现循环重定向的时候 要使用break

前端有一个分发器 后面跟了好几个web server

如果bbs 用户上传附件 但是 下一次访问 就负载不到了

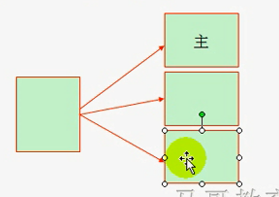
第一种 就是持久链接 ----- 但是 第二次访问 第二天 就不可以了【持久链接对那种非时间过长的 比如购物车这样的session有效 这种附件 由于要长期甚至永久存在 所以 不OK ----- 也不能使用memcached 因为缓存是存放不重要内容的 是内存中 不能持久保存 所以 必须把缓存中的文件落地】

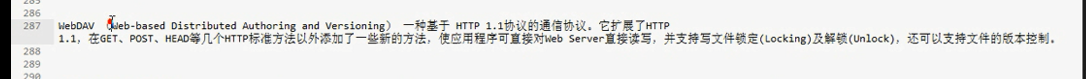
第二种就是同步找一个主服务器



但是访问的服务器未必是主服务器

所以 可以读写分离 服务器如果是写操作 就定到最上面的服务器 --- 一直作为主服务器



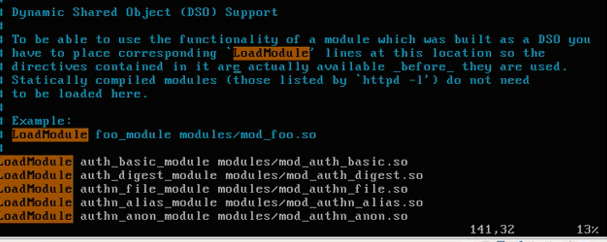


WEBDAV ----- 类似于支持web服务器的读写操作

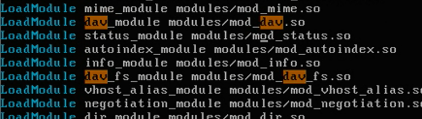
WebDAV为HTTP1.1的新的服务 put

比如两个服务器

让100.7支持上传操作 ---- 修改上面的httpd配置







一个叫dav module 一个叫dav\_fs\_module 有了这两个dav module 就可以进行dav操作

找到directory



加一项dav on



重启web 服务

现在尝试上传  

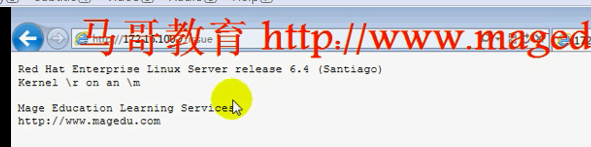

说你没有权限 因为/var/www/是root

但是 运行的是root”:root

修改资源权限



访问



这里面使用的方法就是一个put方法

期望只有2可以上传

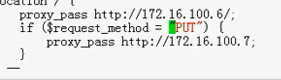
但是 如果做了负载均衡 这两个web服务器都可以传服务器

所以 这个时候 修改Nginx





这里面 就是 $request\_method =

 ---- 空格的问题

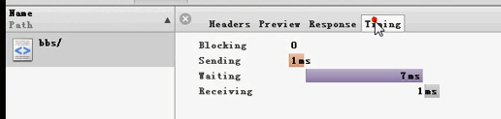




这样就读写分离了

然后就把100.7上面的内容 同步到100.6中去 这样 文件就完成了同步

可以定义两个upstream 一个是读组 一个写组 这样就是一组读服务器 和 一组写服务器



这里面Timing选项卡也比较重要 Sending是向服务器发送请求花费1ms

等待服务器响应 花费7ms

服务器返回响应 1ms

Headers选项卡也很重要

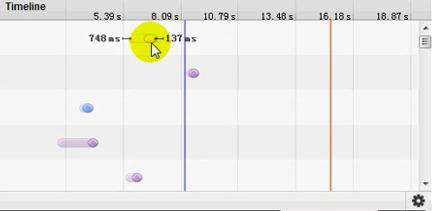


这个也是可以分析资源的

浏览器是多线程 ---- 有些资源是并行获取的

修改一个站点





当然是多线程的 很多资源是并行获取的 在IE浏览器没有 要安装一个叫HttpWatch 有免费版的 也有专业版的 --- 不再演示了

同时httpWatch比chrome更强大 甚至可以录制页面资源的加载过程