**Nginx学习笔记**

Nginx：是一个高性能的HTTP和反向代理的服务器，也是一个IMAP/POP3/STMP代理服务器。

官网：http://nginx.org

# 1 安装nginx

|  |  |
| --- | --- |
| 安装依赖包 | yum install pcre pcre-devel |
| 下载一致性hash模块 | wget https://github.com/replay/ngx\_http\_consistent\_hash/archive/master.zip |
| 解压 | unzip ngx\_http\_consistent\_hash-master.zip |
| 下载nginx源码 | wget http://nginx.org/download/nginx-1.10.1.tar.gz |
| 解压 | tar -zxvf nginx-1.10.1.tar.gz |
| 进入目录 | cd nginx-1.10.1 |
| 检查编译环境 | ./configure --prefix=/usr/local/nginx --with-http\_stub\_status\_module --add-module=/home/vison/ngx\_http\_consistent\_hash-master |
| 编译安装 | make && make install |

启动Nginx：**/usr/local/nginx/sbin/nginx**

如果有其他服务占用80端口，会导致启动失败，先查看哪个服务占有80端口，然后关闭进程即可

netstat -antp

kill -9 进程id

# 2 Nginx信号控制

信号类型:

|  |  |
| --- | --- |
| TERM, INT | Quick shutdown |
| QUIT | Graceful shutdown 优雅的关闭进程,即等请求结束后再关闭 |
| HUP | Configuration reload ,Start the new worker processes with a new configuration Gracefully shutdown the old worker processes  改变配置文件,平滑的重读配置文件 |
| USR1 | Reopen the log files 重读日志,在日志按月/日分割时有用 |
| USR2 | Upgrade Executable on the fly 平滑的升级 |
| WINCH | Gracefully shutdown the worker processes 优雅关闭旧的进程(配合USR2来进行升级) |

使用方法：

获取Nginx的主进程号：**ps -aux | grep nginx**

发送信号：**kill -HUP 进程号**

另外不用查看进程号方式

**kill -HUP `cat /etc/nginx/logs/nginx.pid`**

其中nginx.pid记录的是Nginx运行的pid

也可以用**nginx -s stop, quit, reopen, reload**

nginx -s stop 等价于 kill -INT pid

nginx -s quit 等价于 kill -QUIT pid

nginx -s reload等价于 kill -HUP pid

nginx -s reopen等价于 kill -USER1 pid

# 3 nginx配置

## 3.1全局区

工作子进程数

|  |
| --- |
| worker\_processes 1; |

一般设置为CPU数\*核数，设置大的话各worker会争夺CUP资源。

最大连接数

|  |
| --- |
| events {  worker\_connections 1024;  } |

每个worker process的最大连接数，根据电脑硬件配置设定大概数值

## 3.2 虚拟主机区

http服务器，里面包含虚拟主机，支持3种类型虚拟主机，基于IP、基于域名、基于端口的虚拟主机。

|  |
| --- |
| http{ // http服务器的主要字段  Server1 { // 虚拟主机段  Location / { // 定位,把特殊的路径或文件再次定位  } // 例如php单独处理  }  Server2 {  }  .  .  .  ServerN {  }  } |

**(1) 基于IP的虚拟主机**

|  |
| --- |
| server {  listen 80; // 监听端口  server\_name 192.168.8.104; // IP地址  location / {  root /var/www/html; // 根目录  index index.html; // 索引文件  }  } |

**(2) 基于域名的虚拟主机**

|  |
| --- |
| server {  listen 80; // 监听端口  server\_name vs.com; // 域名  location / {  root /var/www/html; // 根目录  index index.html; // 索引文件  }  } |

如果域名不属于你的话，你又想访问，需要修改hosts，把域名记录在hosts文件上即可。

**(3) 基于端口的虚拟主机**

|  |
| --- |
| server {  listen 8080; // 监听端口  server\_name 192.168.8.104; // IP  location / {  root /var/www/html; // 根目录  index index.html; // 索引文件  }  } |

配置完毕后测试一下配置是否正确：**/usr/local/nginx/sbin/nginx -t**

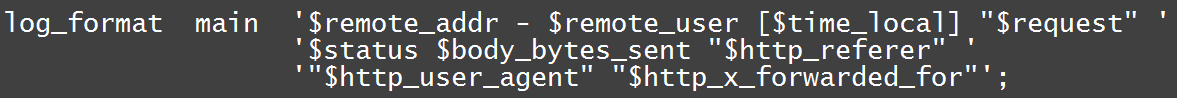
# 4. 日志管理

nginx支持给每个虚拟机独立的保存管理日志。

## 4.1 开启日志

记得取消注释掉main格式的语句

|  |
| --- |
| access\_log /usr/local/nginx/logs/access.log main;  保存路径 日志格式 |



上图为log\_format输出格式，具体可设置的参数格式如下表，可以自定义日志输出方式。

|  |  |  |
| --- | --- | --- |
| 参数 | 说明 | 示例 |
| $remote\_addr | 客户端地址 | 211.28.65.253 |
| $remote\_user | 客户端用户名称 | -- |
| $time\_local | 访问时间和时区 | 18/Jul/2012:17:00:01 +0800 |
| $request | 请求的URI和HTTP协议 | "GET /article-10000.html HTTP/1.1" |
| $http\_host | 请求地址，即浏览器中你输入的地址（IP或域名） | [www.it300.com 192.168.100.100](http://www.it300.com/) |
| $status | HTTP请求状态 | 200 |
| $body\_bytes\_sent | 发送给客户端文件内容大小 | 1547 |
| $http\_referer | url跳转来源，可以统计用户从那个入口过来 | <https://www.baidu.com/> |
| $http\_user\_agent | 用户终端浏览器等信息 | "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; SV1; GTB7.0; .NET4.0C; |
| $http\_x\_forwarded\_for | 在经过代理时，代理把源头请求IP加在此头信息中，即用户请求的原始IP | 192.168.8.101 |
| $upstream\_status | upstream状态 | 200 |
| $ssl\_protocol | SSL协议版本 | TLSv1 |
| $ssl\_cipher | 交换数据中的算法 | RC4-SHA |
| $upstream\_addr | 后台upstream的地址，即真正提供服务的主机地址 | 10.10.10.100:80 |
| $request\_time | 整个请求的总时间 | 0.205 |
| $upstream\_response\_time | 请求过程中，upstream响应时间 | 0.002 |

自定义格式：

|  |
| --- |
| log\_format mymain '[$remote\_addr], [$remote\_user], [$time\_local], [$request], '  '[$status], [$body\_bytes\_sent], [$http\_referer], '  '[$http\_user\_agent], [$http\_x\_forwarded\_for], '  '[$upstream\_addr], [$upstream\_status], [$upstream\_response\_time], [$request\_time]\n'; |

## 4.2 日志定时切割分类保存

需要用到shell脚本和定时任务相关知识。

**(1) shell脚本**

|  |
| --- |
| #!/bin/bash  # 保存路径  base\_path='/data/nginx/logs'  # 按年月日命名文件夹  path\_name=$(date -d yesterday +"%Y-%m-%d")  # 创建文件夹  mkdir -p $base\_path/$path\_name  # 文件名最后追加时和分  hm=$(date -d yesterday +"%H:%M")  # 测试输出文件名  #echo $base\_path/$path\_name/access\_$hm.log  # 移动日志文件并改名  mv /usr/local/nginx/logs/access.log $base\_path/$path\_name/access\_$hm.log  # 新建文件  touch /usr/local/nginx/logs/access.log  # nginx重读日志  /usr/local/nginx/sbin/nginx -s reopen  # kill -USR1 `/usr/local/nginx/logs/nginx.pid` |

**(2) 添加定时任务**

|  |  |
| --- | --- |
| #分 时 日  月 周  执行的命令 (共6列) | |
| 特殊字符 | 代表意义 |
| \*(星号) | 代表任何时刻都接受的意思。 |
| ,(逗号) | 代表分隔时段的意思。例如要执行的任务是3:00与6:00时：  0 3,6 \* \* \* command |
| -(减号) | 代表一段时间范围内。例如在8点到12点之间的每小时的20分都进行一项工作：  20 8-12 \* \* \* command |
| /n(斜线) | n代表数字，每隔n单位间隔，例如每五分钟进行一次：  \*/5 \* \* \* \* command |

编辑定时任务：**crontab –e**

测试每两分钟切割日记

\*/2 \* \* \* \* sh /data/nginx/runlogs.sh

一般建议在凌晨2～4点之间切割日记，因为这时系统负载小，例如设置每天凌晨3点切割日志

|  |
| --- |
| 0 3 \* \* \* sh /data/nginx/runlogs.sh |

## 4.3 查看nginx错误记录

当nginx运行出现错误时，可以查看错误日志，查看最近五行错误记录

**tail 5 /usr/local/nginx/logs/error.log**

# 5 location定位

location有“定位的意思”，根据URI进行不同的定位。

在虚拟机设置是必须的，location把网站的不同定位到不同的处理方式上。

比如碰到\*.php文件时去调用PHP解释器。

## 5.1 语法

|  |
| --- |
| 匹配修饰符 匹配模式  location [=|~|~\*|^~] patt {  } |

其中修饰符可以不写。

分为3种大类型：

location = patt {} [精准匹配]

location ~ patt{} [正则匹配]

location patt{} [普通匹配]

如果所有location都没有匹配，最终继承nginx默认目录的index.html

如果location没有root选项，也会继承nginx默认目录的index.html

## 5.2 匹配优先级

**(1)精准匹配和普通匹配**

如果精准匹配和普通匹配同时都命中uri，优先选择精准匹配。

|  |
| --- |
| # 精准匹配  location = / {  root /var/www/html/;  index index.htm index.html;  }  # 普通匹配  location / {  root /usr/local/nginx/html;  index index.html index.htm;  } |

在配置文件里既有精准匹配，又有普通匹配时，首先是进行精准匹配，例如在浏览器输入http://192.168.8.104，精准匹配中”/”，得到index页为index.htm，内部跳转uri为<http://192.168.8.104/index.htm>

此时精准匹配不到，而普通匹配命中，最终结果访问了/usr/local/nginx/html/index.htm

**(2) 正则匹配和普通匹配**

如果正则匹配和普通匹配同时都命中uri，优先选择正则匹配。

|  |
| --- |
| # 正则匹配  location ~ image {  root /var/www/html;  index index.html;  }  # 一般匹配  location / {  root /usr/local/nginx/html;  index index.html index.htm;  } |

如果我们访问 <http://192.168.8.104/image/logo.png>

普通匹配”/”与”/image/logo.png”匹配

正则匹配”image”与”image/logo.png”

正则表达式的成果发挥作用，最终访问/var/www/image/logo.png

**(3) 普通匹配和普通匹配**

哪个普通匹配更长，就会使用哪个普通匹配成果。

|  |
| --- |
| # 正则匹配  location / {  root /usr/local/nginx/html;  index index.html index.htm;  }  # 正则匹配  location /foo {  root /var/www/html;  index index.html index.htm;  } |

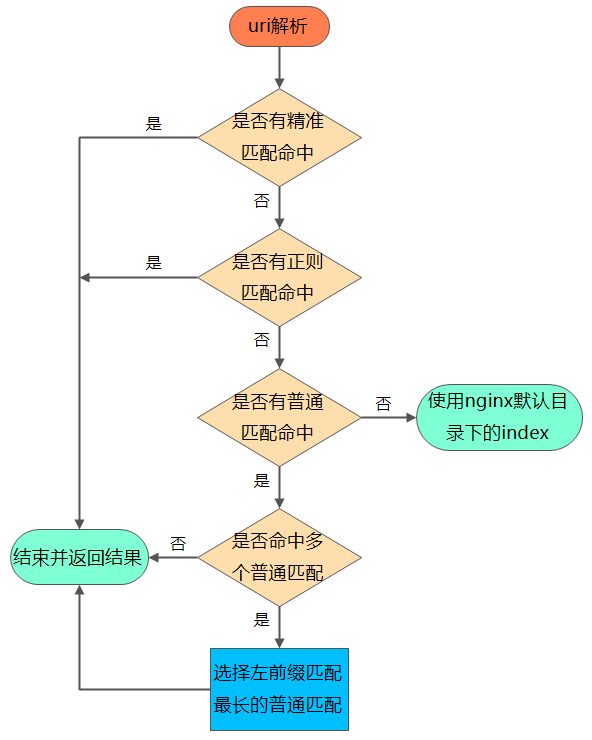
访问<http://192.168.8.104/foo>/1.html，

普通匹配”/”与”/foo/1.html”匹配

普通匹配”/foo”与”/foo/1.html”也匹配

普通匹配”/foo”发挥作用，因为”/foo”左前缀匹配更长，最终访问/var/www/html/index.html

## 5.3 location匹配命中流程图



# 6 nginx与php整合

安装php，注意编译安装是让php作为独立进程，即在configure是加入enable-fpm选项。

安装后启动php软件：**/usr/local/php/sbin/php-fpm**

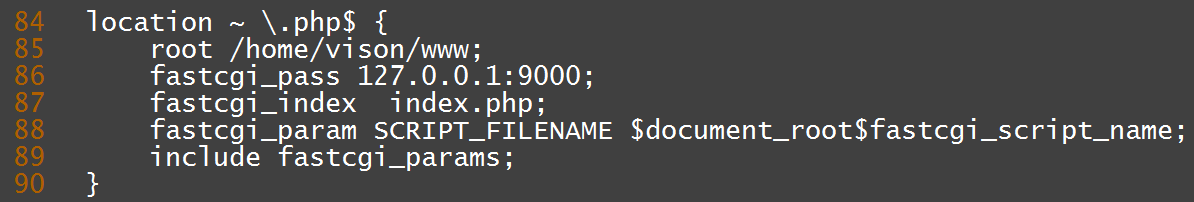
设置nginx配置，使其支持php

**vim /etc/nginx/nginx.conf**

在http{}里添加下面内容

|  |
| --- |
| server{  listen 80;  server\_name 192.168.8.104;  #access\_log logs/log.log;  location ~ \.php$ {  root /home/vison/www;  fastcgi\_pass 127.0.0.1:9000;  fastcgi\_index index.php;  fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  include fastcgi\_params;  }  } |

配置结果如下图：



最后重启nginx: **/usr/local/nginx/sbin/nginx -s reload**

# 7 rewrite重写

查看nginx使用的系统变量，**cat /etc/nginx/fastcgi\_params**

有些版本文件名是fastcgi.conf

## 7.1 重写指令

if (条件){} 设定条件

set 设置变量

return 返回状态码

break 跳出rewrite

rewrite 重写

## 7.2条件判断使用的符号

条件判断使用的符号：

”=”用于比较字符串。

”~”正则匹配，区分大小写。

”~\*” 正则匹配，不区分大小写

”-f –d -e” 判断目录、文件是否存在。

实例1：使用”=”号，禁止某ip地址访问

|  |
| --- |
| # 禁止某ip访问  if ($remote\_addr = 192.168.8.102) {  return 403;  } |

实例2：使用正则匹配”~”，判断浏览器，然后重写

|  |
| --- |
| # 判断是IE浏览器访问则重写  if ($http\_user\_agent ~\* msie) {  rewrite ^.\*$ /ie.htm;  break; #(不break会循环重定向)  } |

实例3：使用-e，判断访问的文件是否存在

|  |
| --- |
| # 判断访问的文件或路径是否存在  if (!-e $document\_root$fastcgi\_script\_name) {  rewrite ^.\*$ /404.html;  break;  } |

访问<http://192.168.8.104/none.html>

实例4：使用set变量

set 是设置变量用的，可以用来达到多条件判断时作标志用，达到apache下的 rewrite\_condition的效果

|  |
| --- |
| # 判断如果IE浏览访问则重写，这里用set作为例子，实际一般不这样用  if ($http\_user\_agent ~\* msie) {  set $isie 1;  }  if ($fastcgi\_script\_name = ie.html) {  set $isie 0;  }  if ($isie 1) {  rewrite ^.\*$ ie.html;  #这里不用break  } |

## 7.3 重写和重定向的区别

观察访问日志, 日志中显示的访问路径依然是GET /none.html HTTP/1.1，因为服务器内部的rewrite和302跳转不一样。跳转的话URL都变了，变成重新http请求404.html，而内部rewrite，上下文没变，fastcgi\_script\_name 仍然是none.html因此 会循环重定向。

# 8 gzip压缩

意义：节省传输带宽和时间，但是会消耗CPU资源，利大于弊，特别是大网站上使用效果明显。

原理：

浏览器请求的header显示支持有三种压缩模式，gzip、deflate、sdch

服务器回应：把内容用gzip方式压缩为二进制文件发送给浏览器，浏览器再把压缩的文件进行解压后显示结果。

## 8.1 nginx 配置gzip压缩的常用参数

|  |  |
| --- | --- |
| gzip on|off; | 是否开启gzip |
| gzip\_buffers 32 4K| 16 8K | 缓冲(压缩在内存中缓冲几块? 每块多大?) |
| gzip\_comp\_level [1-9] | 推荐6 压缩级别(级别越高,压的越小,越浪费CPU计算资源) |
| gzip\_disable | 正则匹配UA 什么样的Uri不进行gzip |
| gzip\_min\_length 8000 | 开始压缩的最小长度(再小就不要压缩了,意义不在) |
| gzip\_http\_version 1.0|1.1 | 开始压缩的http协议版本(可以不设置,目前几乎全是1.1协议) |
| gzip\_proxied | 设置请求者代理服务器,该如何缓存内容 |
| gzip\_types text/plain application/xml | 通过查看nginx安装目录下的mime.types文件得到相应的类型。对哪些类型的文件用压缩 如txt,xml,html ,css |
| gzip\_vary on|off | 是否传输gzip压缩标志 |

**注意:**

对于图片/mp3这样的二进制文件不必压缩，用正则匹配。

因为压缩率比较小，比如100->80字节，而且压缩也是耗费CPU资源，

根据网站具体情况设置开始压缩的最小文件。

用于测试的简单配置：

|  |
| --- |
| gzip on;  gzip\_buffers 32 4K;  gzip\_comp\_level 6;  gzip\_min\_length 600;  gzip\_types text/css text/xml application/javascript text/plain;  #gzip\_vary off;  #gzip\_disable /^.\*\.(jpg|gif|png|bmp|mp3|rar|zip)$/i; |

## 8.2 查看nginx支持文件类型

压缩和不压缩哪些文件类型从mime.type文件来查找

**cat /etc/nginx/mime.types**

## 8.3 测试压缩文件

使用到curl测试工具，安装：**yum install curl**

**curl -I -H "Accept-Encoding: gzip, deflate" "http://192.168.8.104"**



# 9 nginx的缓存设置

nginx的缓存可以提高网站性能，特别是大流量网站上，设置缓存图片、静态文件，可以节省带宽和服务器压力。

## 9.1 原理

服务器响应文件内容时，同时响应2个标签值：ETag标签(内容的签名，修改内容后也会跟着改变)和Last\_Modified，浏览器下次去请求时，头信息发送这两个标签，服务器检测文件有没有发生变化，如果没有变化，直接把头信息返回ETag、Last\_Modified给浏览器，浏览器知道内容没有改变，于是直接调用本地缓存。这个过程，也请求了服务器，但是只传输head信息，内容很小。

服务器把文件缓存在用户的浏览器端，缓存后浏览器在文件过期时间内直接调用本地缓存文件，提高用户体验。

另外304 也是一种很好的缓存手段。

**适用文件：**

对于变化周期较短的，如静态htm、js、css比较适于用这个方式。

对于不常改变的图片，如jpg、png等也适用于这个方式。

## 9.2 设置过期时间

nginx中设置过期时间,非常简单，在location或if段里写。

|  |
| --- |
| expires 30s; # 缓存时间30秒  expires 30m; # 缓存时间30分钟  expires 2h; # 缓存时间3小时  expires 30d; # 缓存时间30天 |

例如：

|  |
| --- |
| location ~\* \.(jpg|gif|png|bmp)$ {  root /home/vison/www; # 图片文件在服务器的位置  expires 7d;  }  location ~\* \.(js|css)$ {  root /home/vison/www; # js、css文件在服务器的位置  expires 8h;  } |

# 10 nginx反向代理

## 10.1 原理



当用户访问静态文件时，nginx自己处理，当用户访问动态文件php时，nginx自己不处理php的相关请求，而是把php的相关请求转发给apache来处理，apache处理结果返回给nginx，nginx在把结果返回给用户。

这个过程叫反向代理，俗称”动静分离”

NGINX+APACHE+PHP会更好，因为可以充分利用NGINX的“反向代理”技术。将静态文件由NGINX处理，动态文件（PHP）由APACHE处理，这是最高效的处理方式。

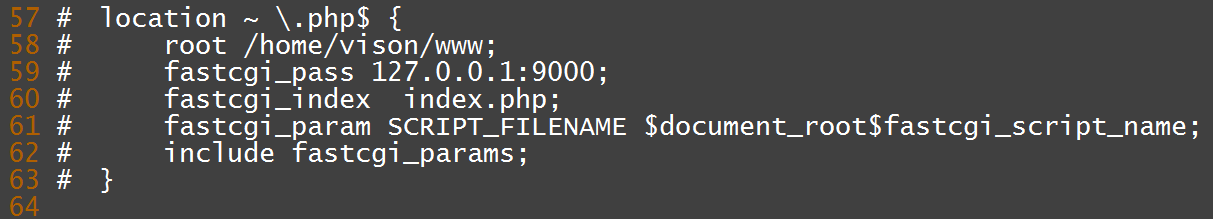
但是，一般访问量不大的网站不需要这么做，因为没有高并发的情况下，这样做并不能体现非常大的优势。

## 10.2 反向代理实现

**(1)设置nginx配置文件**

**vim /etc/nginx/nginx.conf**

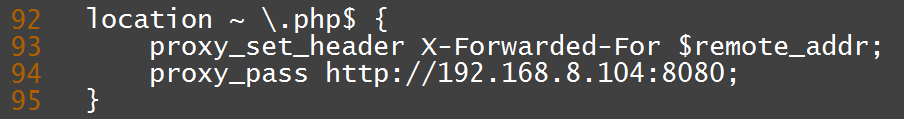
删除或取消location定位php文件的设置



添加php文件代理地址(单个代理服务器)

|  |
| --- |
| location ~ \.php$ {  proxy\_set\_header X-Forwarded-For $remote\_addr;  proxy\_pass http://192.168.8.104:8080;  } |

修改结果：

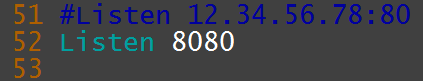


**(2)设置apache配置文件**

把监听80端口改为监听8080端口

**vim /usr/local/apache/httpd/httpd.conf**

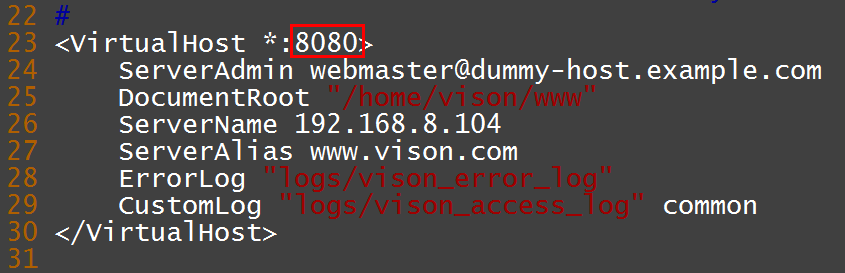
修改如下图：



设置虚拟机的端口也改为8080

**vim httpd/extra/httpd-vhosts.conf**

修改如下图：



**(3)测试反向代理**

在网站根目录写一个简单的test.php文件测试，内容带有访问静态图片文件和动态php函数，内容如下：

|  |
| --- |
| <?php echo rand(1,100),'<br>'; ?>  <image src="images/1.jpg"/> |

使用浏览器访问<http://192.168.8.104/test.php>



查看nginx访问日志

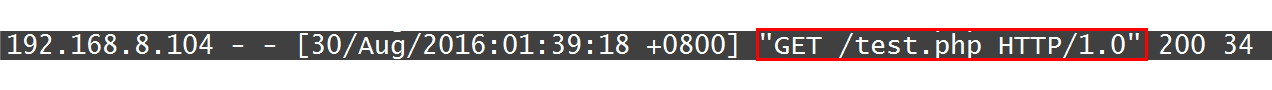
**tail –n 2 /var/log/nginx/access.log**



由日志得到图片1.jpg静态文件是由nginx响应的。

查看apache访问日志

**tail –n 2 /usr/local/apache/logs/ vison\_access\_log**



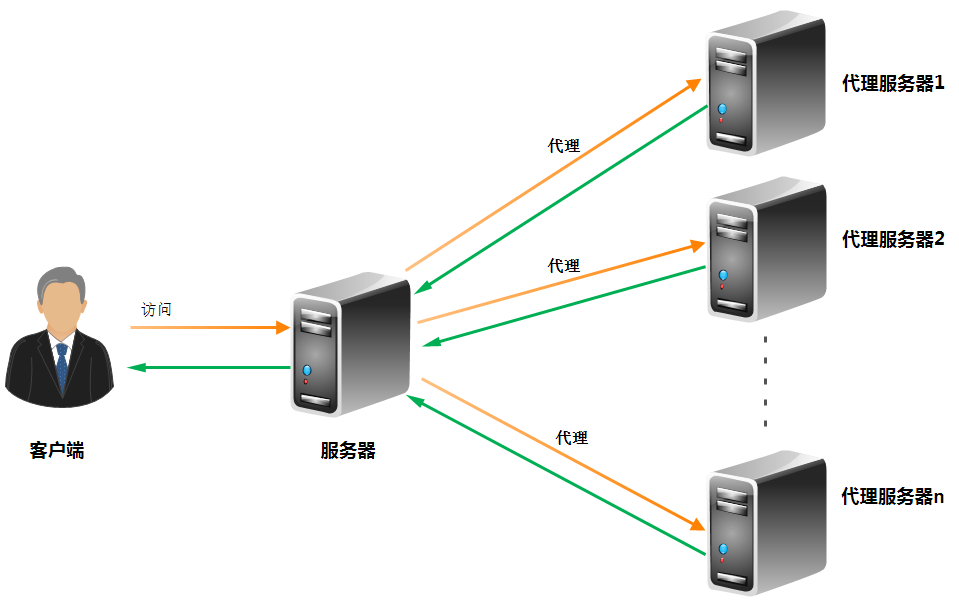
由日志得到动态文件test.php是有apache执行的。

说明反向代理成功。

# 11 集群和负载均衡

反向代理后端如果有多台服务器，自然可形成负载均衡

## 11.1 集群



服务器集群就是指将很多服务器集中起来一起进行同一种服务，在客户端看来就像是只有一个服务器。集群可以利用多个计算机进行并行计算从而获得很高的计算速度，提高用户体验。也可以用多个计算机做备份，从而使得任何一个机器坏了整个系统还是能正常运行。

## 11.2 负载均衡

一个任务分摊到多台服务器上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。

例如当网站访问量大时，通过负载均衡把任务分配给多台服务中的一台去执行，相当于多台服务器共同去完成一向任务，比单台服务器完成一向任务时相应更快，提高用户体验。

## 11.3 集群和负载均衡实现

当有多台服务器代理时，把多台服务器用upstream绑定在一起并起个组名，然后proxy\_pass指向该组即可。

默认的均衡的算法很简单，就是针对后端服务器的轮循顺序逐个请求。

也有其他负载均衡算法，如一致性哈希，需要安装第3方模块。

首先在网站根目录下新建一个test.html文件，添加下面内容：

|  |
| --- |
| <html>  <head>  <title>test</title>  </head>  <body>  <h3>简单测试集群和负载均衡</h3>  <img src="images/5.jpg"/>  </body>  </html> |

**(1)设置nginx配置文件，模拟多台虚拟主机**

**vim /etc/nginx/nginx.conf**

添加三个不同端口的虚拟主机，并把他们组成一个图片服务器组。

|  |
| --- |
| server {  listen 81;  server\_name 192.168.8.104;  root /home/vison/www;  access\_log /var/log/nginx/access\_81.log;  }  server {  listen 82;  server\_name 192.168.8.104;  root /home/vison/www;  access\_log /var/log/nginx/access\_82.log;  }  server {  listen 83;  server\_name 192.168.8.104;  root /home/vison/www;  access\_log /var/log/nginx/access\_83.log;  }  upstream imgserver {  server 192.168.8.104:81 weight=1 max\_fails=2 fail\_timeout=3;  server 192.168.8.104:82 weight=1 max\_fails=2 fail\_timeout=3;  server 192.168.8.104:83 weight=1 max\_fails=2 fail\_timeout=3;  } |

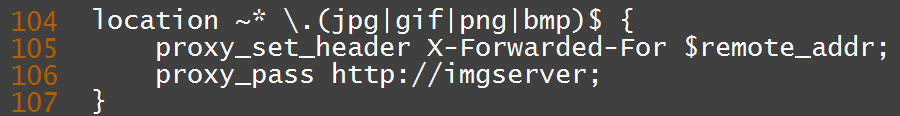
修改结果如下图：



**(2)设置location定位，如果请求的是图片则转发给代理服务器**

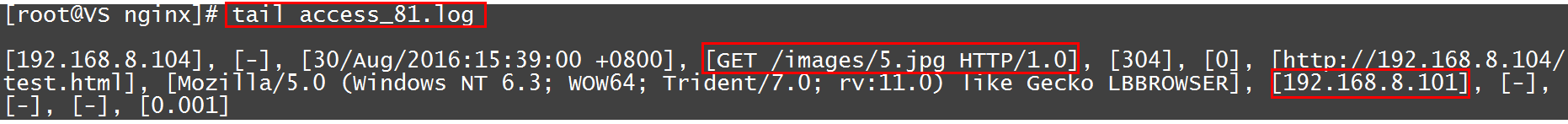
|  |
| --- |
| location ~\* \.(jpg|gif|png|bmp)$ {  proxy\_set\_header X-Forwarded-For $remote\_addr;  proxy\_pass http://imgserver;  } |

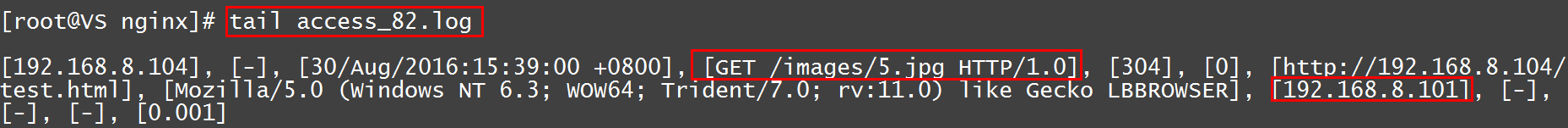
设置结果如下图：

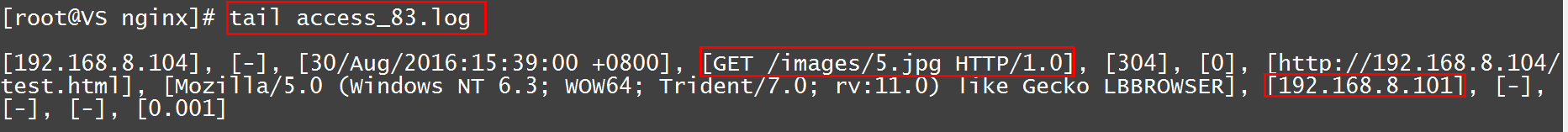


**(3)查看虚拟主机的访问日志**

在浏览器访问<http://192.168.8.104/test.html>，连续刷新三次，然后查看三个虚拟主机的访问日志。







如上图所示的结果，说明集群与负载均衡已经实现，不过负载均衡是简单在三个服务器的轮循顺序策略。

# 12 nginx连接memcached缓存

## 12.1 nginx连接一个memcached服务器测试

**(1) 配置nginx.conf**

|  |
| --- |
| location / {  set $memcached\_key "$uri"; #$uri作为memcached的key  memcached\_pass 127.0.0.1:11211; # 去该缓存服务器找key  error\_page 404 /cb.php; # 没有找到把key转交给cb.php处理  }  location ~ \.php$ {  root /home/vison/www;  fastcgi\_pass 127.0.0.1:9000;  fastcgi\_index index.php;  fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  include fastcgi\_params;  } |

**(2) 在网站根目录下新建一个cb.php，添加下面内容**

|  |
| --- |
| **<?php**  *// $uri作为key，包括/* $uri = $\_SERVER[**'REQUEST\_URI'**];  *// 获取用户uid* $uid = *substr*($uri,5,*strpos*($uri,**'.'**)-5);  *// 连接数据库* $mysqli = **new** mysqli(**'192.168.8.102'**, **'root'**, **'123456'**,**'test'**); **if** (*mysqli\_connect\_errno*()){  **die**(**'Unable to connect!'**). *mysqli\_connect\_error*();  **return**; } $sql = **'set names utf8'**; $mysqli->query($sql);  *// 获取和显示结果* $sql = **'selct \* from user where id='**.$uid; **if** ($v = getUser($mysqli,$sql)){  *print\_r*($v); }**else**{  **echo 'not fond key = '**,$uri; }   $mysqli->close();  **function** getUser($mysqli, $sql){ *// 获取用户信息并写入缓存* **if** ($result = $mysqli->query($sql)){  **if**($row = $result->fetch\_assoc()){  sql2memcache($\_SERVER[**'REQUEST\_URI'**], $row);  $result->free();  **return** $row;  }  $result->free();  }**else**{  **echo 'error: "'**,$sql,**'"'**;  }  **return null**; }  *// php-memcache* **function** sql2memcache($key,$val){ *// 把数据库的数据添加到缓存中* $mem = **new** memcache();  $mem->addServer(**'192.168.8.102'**,11211);   **if**($mem->add($key,$val,0)){  **echo 'add "'**, $key, **'" to memcached ok!<br>'**;  }  $mem->close(); } |

**(3) 在数据库test下新建一个用来测试的user表，并插入几条内容**

|  |
| --- |
| use test;  create table user( id int, name char(20), gender char(2), age tinyint );  insert into user values (1,'刘备','男',35);  insert into user values (2,'关羽','男',28);  insert into user values (3,'张飞','男',29); |

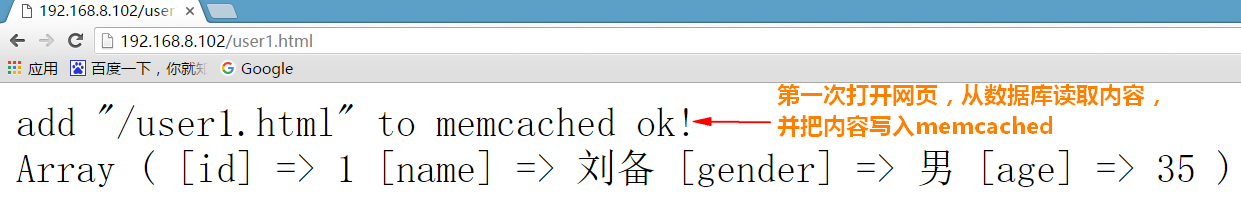
**(4) 启动nginx、php-fpm、memcached**

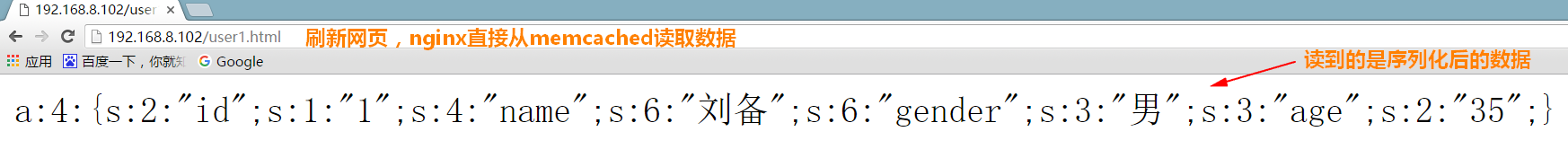
|  |
| --- |
| systemctl start nginx  /usr/local/php/sbin/php-fpm  memcached -m 8 -u nobody -p 11211 -vv |

**(5) 测试数据是从memcached读取还是从数据库读取**

第一次打开网址[http://192.168.8.102/user1.html](http://192.168.8.102/user1.html，nginx)

nginx拿到key="/user1.html"去memcached找，没找到对应的key，nginx把key转交给cb.php处理，cb.php根据uid号读取数据，然后再把数据写入memcached。





## 12.2 nginx安装第三方一致性hash模块

查看旧的nginx编译的模块，nginx -V

在原来的nginx模块下增加一个选项--add-module=模块路径

**(1) 重新安装nginx，添加一致性hash模块**

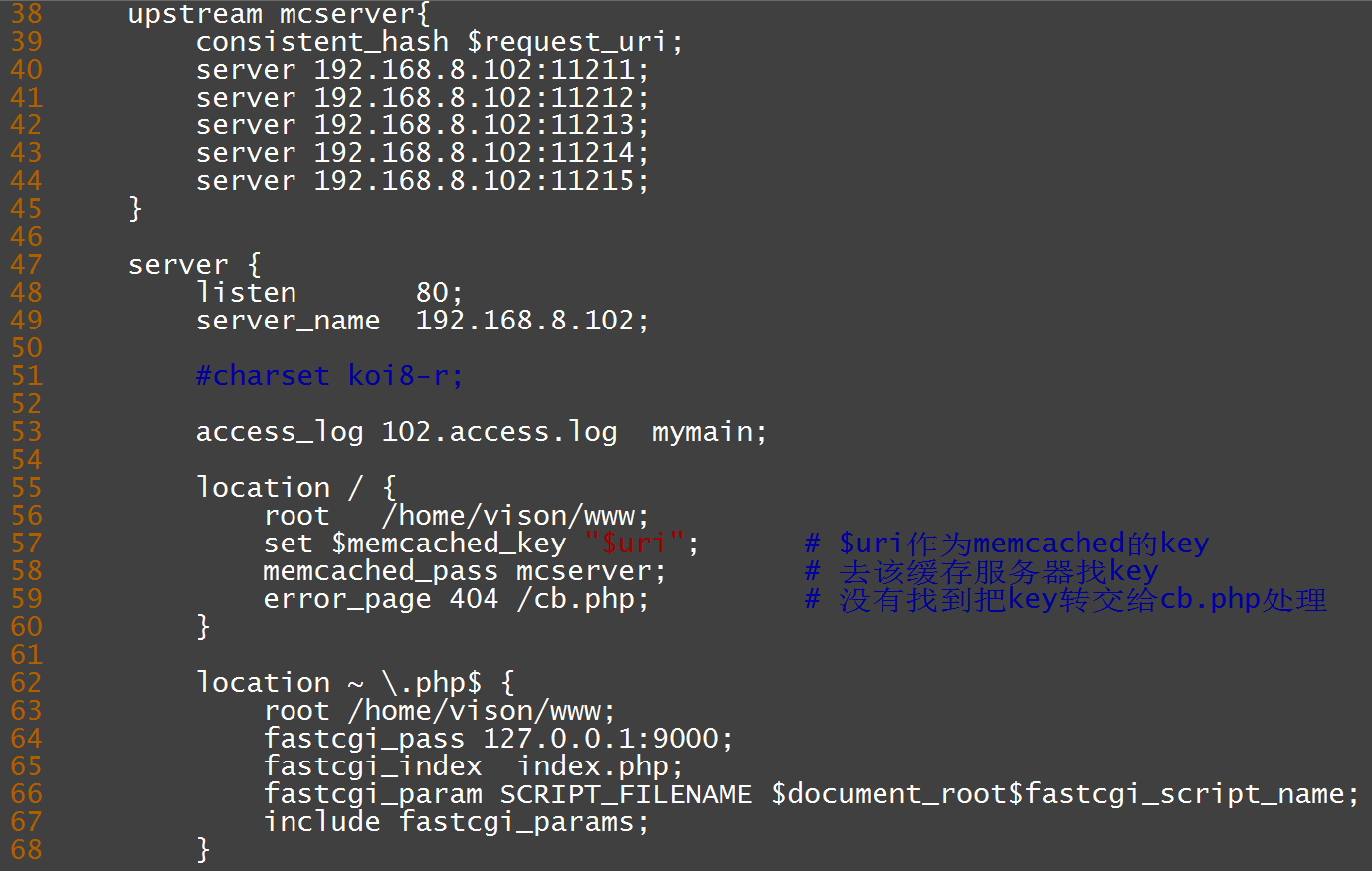
|  |  |
| --- | --- |
| 下载一致性hash模块 | wget <https://github.com/replay/ngx_http_consistent_hash/archive/master.zip> |
| 解压 | unzip ngx\_http\_consistent\_hash-master.zip |
| 下载nginx源码 | wget <http://nginx.org/download/nginx-1.10.1.tar.gz> |
| 解压 | tar -zxvf nginx-1.10.1.tar.gz |
| 进入目录 | cd nginx-1.10.1 |
| 检查编译环境 | ./configure --prefix=/usr/local/nginx --add-module=/home/vison/ngx\_http\_consistent\_hash-master |
| 编译安装 | make && make install |

**(2) 修改配置nginx.conf**

**vim /usr/local/nginx/conf/nginx.conf**

修改内容如下：

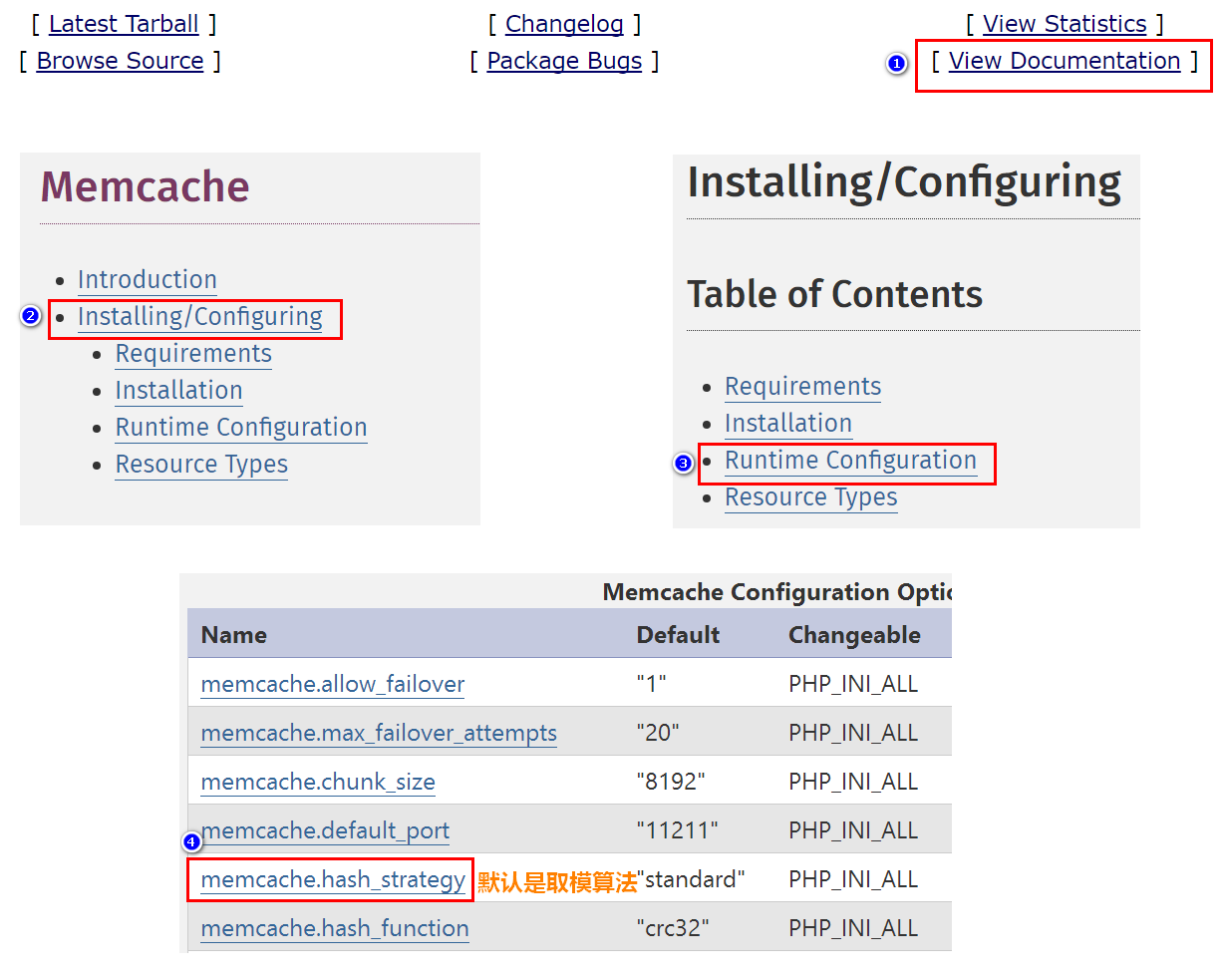
|  |
| --- |
| upstream mcserver{  consistent\_hash $request\_uri;  server 192.168.8.102:11211;  server 192.168.8.102:11212;  server 192.168.8.102:11213;  server 192.168.8.102:11214;  server 192.168.8.102:11215;  }  location / {  root /home/vison/www;  set $memcached\_key $request\_uri; # $uri作为memcached的key  memcached\_pass mcserver; # 去该缓存服务器找key  error\_page 404 /cb.php; # 没有找到把key转交给cb.php处理  }  location ~ \.php$ {  root /home/vison/www;  fastcgi\_pass 127.0.0.1:9000;  fastcgi\_index index.php;  fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  include fastcgi\_params;  } |



**(3) 修改配置php.ini**

注意：推荐使用的是php-memcache扩展，而php-memcached扩展的hash值和nginx的hash值不知为何不一致，设置CRC算法也没用。

查看php-memcache官方文档来开启一致性hash算法。



**vim /usr/local/php/lib/php.ini**

开启一致性hash算法，默认是取模算法。

|  |
| --- |
| 对于php-memcache  memcache.hash\_strategy=consistent    对于php-memcached  memcached.sess\_consistent\_hash=on |

**(4) 编辑cb.php文件**

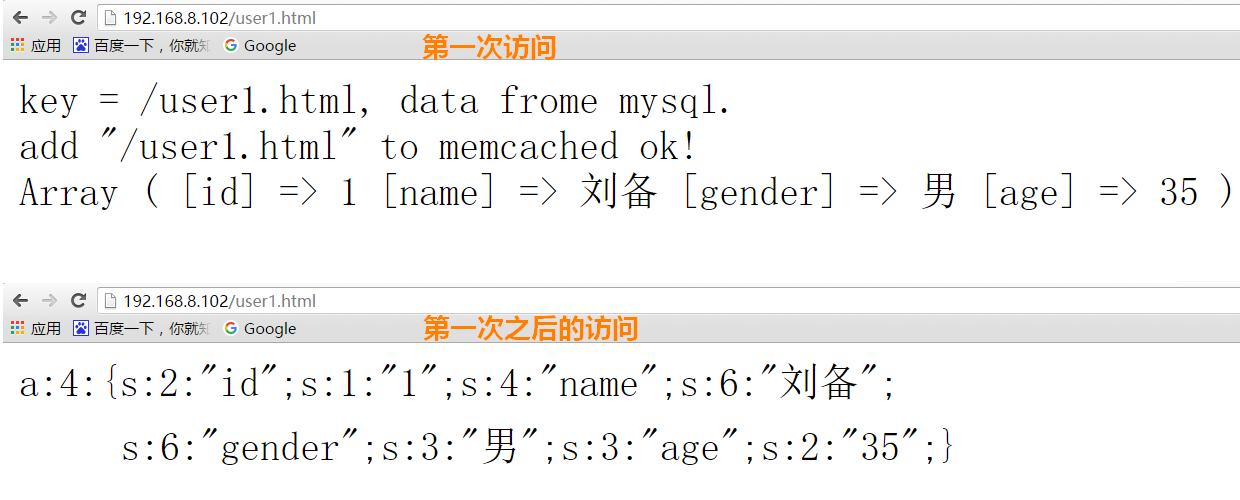
|  |
| --- |
| **<?php**  *// $uri作为key，包括/* $uri = $\_SERVER[**'REQUEST\_URI'**];  **echo 'key = '**,$uri,**', data frome mysql.<br>'**;  *// 获取用户uid* $uid = *substr*($uri,5,*strpos*($uri,**'.'**)-5);  *// 连接数据库* $mysqli = **new** mysqli(**'192.168.8.102'**, **'root'**, **'123456'**,**'test'**); **if** (*mysqli\_connect\_errno*()){  **die**(**'Unable to connect!'**). *mysqli\_connect\_error*();*//注意mysqli\_connect\_error()新特性* **return**; } $sql = **'set names utf8'**; $mysqli->query($sql);  *// 获取和显示结果* $sql = **'selct \* from user where id='**.$uid; **if** ($v = getUser($mysqli,$sql)){  *print\_r*($v); }**else**{  **echo 'not fond key = '**,$uri; }  $mysqli->close();   **function** getUser($mysqli, $sql){ *// 获取用户信息并写入缓存* **if** ($result = $mysqli->query($sql)){  **if**($row = $result->fetch\_assoc()){  sql2memcache($\_SERVER[**'REQUEST\_URI'**], $row);  **return** $row;  }  $result->free();  }**else**{  **echo 'error: "'**,$sql,**'"'**;  }  **return null**; }  *// php-memcache* **function** sql2memcache($key,$val){ *// 把数据库的数据添加到缓存中* $mem = **new** memcache();  $mem->addServer(**'192.168.8.102'**,11211);  $mem->addServer(**'192.168.8.102'**,11212);  $mem->addServer(**'192.168.8.102'**,11213);  $mem->addServer(**'192.168.8.102'**,11214);  $mem->addServer(**'192.168.8.102'**,11215);   **if**($mem->add($key,$val,0)){  **echo 'add "'**, $key, **'" to memcached ok!<br>'**;  }  $mem->close(); }   *// php-memcached的hash值和nginx的hash值不一致，未解决 /\*function sql2memcache($key,$val){ // 把数据库的数据添加到缓存中  $mem = new memcached();   $mem->addServer('192.168.8.102',11211);  $mem->addServer('192.168.8.102',11212);  $mem->addServer('192.168.8.102',11213);  $mem->addServer('192.168.8.102',11214);  $mem->addServer('192.168.8.102',11215);   $mem->setOption(Memcached::OPT\_HASH, Memcached::HASH\_CRC); // $mem->setOption(Memcached::OPT\_LIBKETAMA\_COMPATIBLE,true); // $mem->setOption(Memcached::OPT\_DISTRIBUTION,Memcached::DISTRIBUTION\_CONSISTENT);   if($mem->add($key,$val,0)){  echo 'add "', $key, '" to memcached ok!<br>';  }  var\_dump($mem->getServerByKey($\_SERVER['REQUEST\_URI']));   $mem->quit(); } \*/* |

**(5) 启动nginx、php-fpm、memcached**

|  |
| --- |
| systemctl start nginx  /usr/local/php/sbin/php-fpm  memcached -m 8 -u nobody -p 11211 -vv  memcached -m 8 -u nobody -p 11212 -vv  memcached -m 8 -u nobody -p 11213 -vv |

**(6)测试**

用浏览器打开网页<http://192.168.8.102/user1.html>，打开一次再刷新一次，看是否从缓存读取数据。



# 13 高性能的服务器架设思路

对于高性能网站，如何支撑？

**(1) 对于开发人员**

减少请求量，把HTML和CSS文件合并，多张背景图片合并为一张，减少mysql查询等。

**(2) 对于运维人员**

使用nginx的expires，利用浏览器缓存，减少查询。

**(3) 利用CDN来响应请求**

**(4) 最终剩下不可避免的请求，利用服务器集群+负载均衡来支撑**

整体思想：把不可避免的响应请求工作平均分给每台服务器，理想状态是每台服务器的性能都被充分利用。

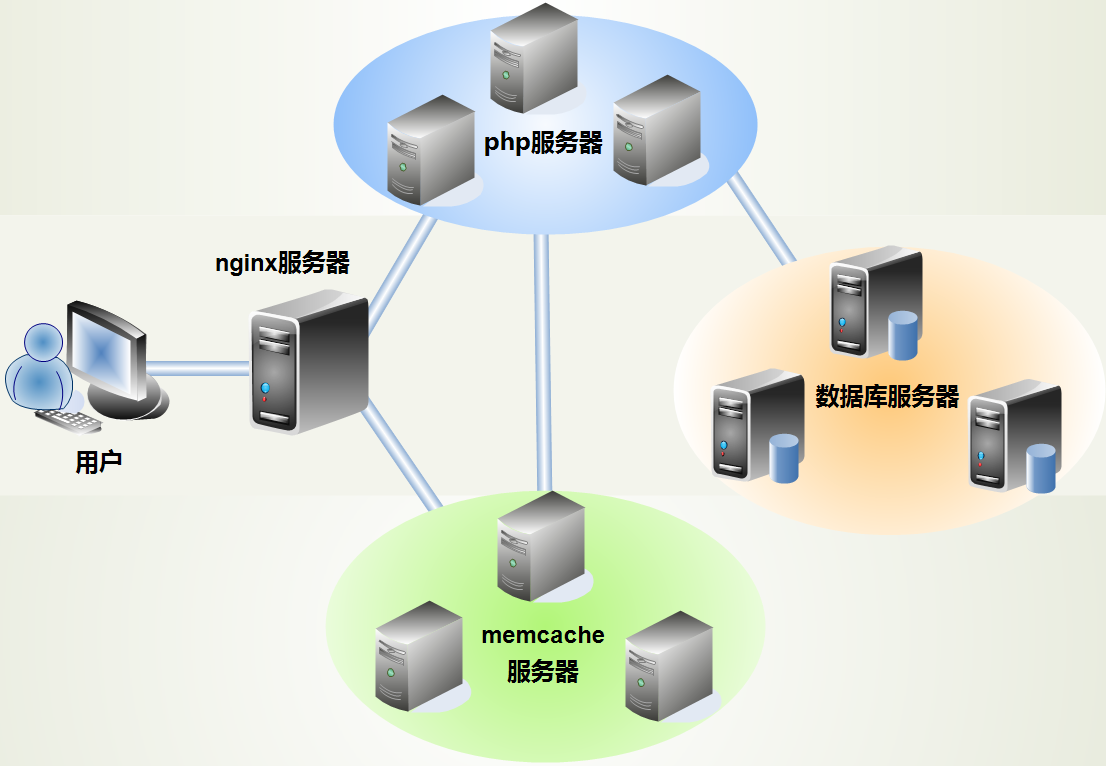
**服务器分类**

IO密集型服务器：数据库存储，使用读写速度比较快的固态硬盘，cpu不用那么强大。

计算密集型服务器：要求CPU运算强大，例如用来压缩视频。

进程密集型服务器：内存要求比较大，例如memcached缓存、几十个php-fpm独立进程

# 14 服务器集群优化和测试



## 14.1 搭建测试nginx环境

**(1) 添加nginx的status选项**

|  |  |
| --- | --- |
| 进入源文件目录 | cd nginx-1.10.1 |
| 检查编译环境 | ./configure --prefix=/usr/local/nginx --with-http\_stub\_status\_module --add-module=/home/vison/ngx\_http\_consistent\_hash-master |
| 编译和安装 | make && make install |

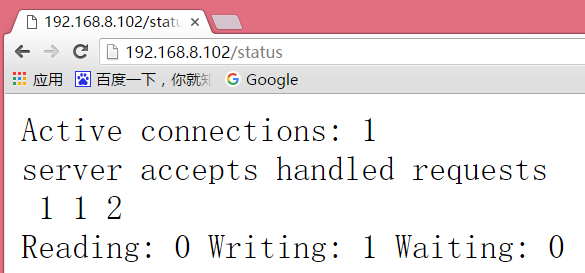
**(2) 配置nginx.conf**

添加location如下：

|  |
| --- |
| location /status{  stub\_status on;  access\_log off; #关闭log  allow 192.168.8.101; # 只允许指定IP访问  deny all;  } |

**(3) 验证nginx的status有没有打开**

在浏览器输入http://192.168.8.102/status



**(4) 使用apache的压力测试插件ab对nginx压力测试**

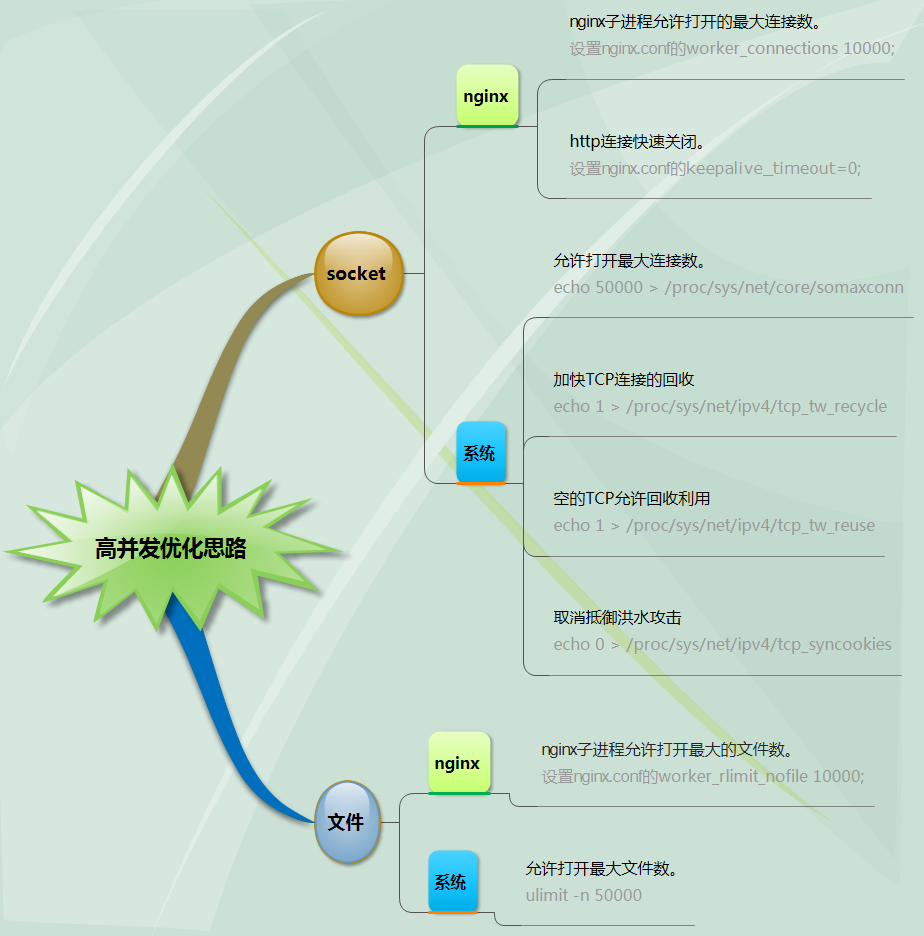
**/usr/local/apache/bin/ab -c 1000 -n 100000 http://192.168.8.102/index.html**

## 14.2 优化nginx和系统参数

高并发优化思路：

同时能建立多少个socket连接

同时能打开和读取多少个文件，然后沿着socket返回。

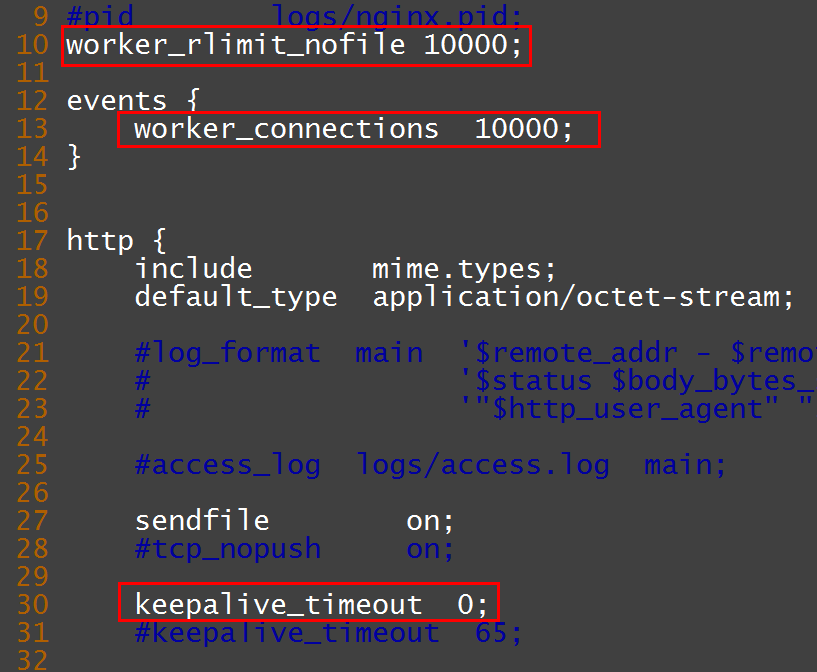


**(1) nginx优化选项**

打开nginx配置文件**vim /usr/local/nginx/conf/nginx.conf**，修改下面选项：

|  |
| --- |
| worker\_rlimit\_nofile 10000;  worker\_connections 10000;  keepalive\_timeout 0; |

示例如下图：

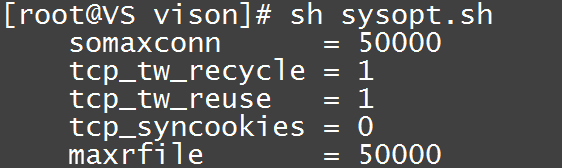


**(2) 修改系统参数**

写一个脚本文件sysopt.sh

|  |
| --- |
| #!/bin/sh  echo 50000 > /proc/sys/net/core/somaxconn  echo 1 > /proc/sys/net/ipv4/tcp\_tw\_recycle  echo 1 > /proc/sys/net/ipv4/tcp\_tw\_reuse  echo 0 > /proc/sys/net/ipv4/tcp\_syncookies  ulimit -n 50000  #查看设置结果  awk '{printf " somaxconn = %s\n",$0}' /proc/sys/net/core/somaxconn  awk '{printf " tcp\_tw\_recycle = %s\n",$0}' /proc/sys/net/ipv4/tcp\_tw\_recycle  awk '{printf " tcp\_tw\_reuse = %s\n",$0}' /proc/sys/net/ipv4/tcp\_tw\_reuse  awk '{printf " tcp\_syncookies = %s\n",$0}' /proc/sys/net/ipv4/tcp\_syncookies  ulimit -n | awk '{printf " maxrfile = %s\n",$0}' |

运行脚本文件：**sh sysopt.sh**

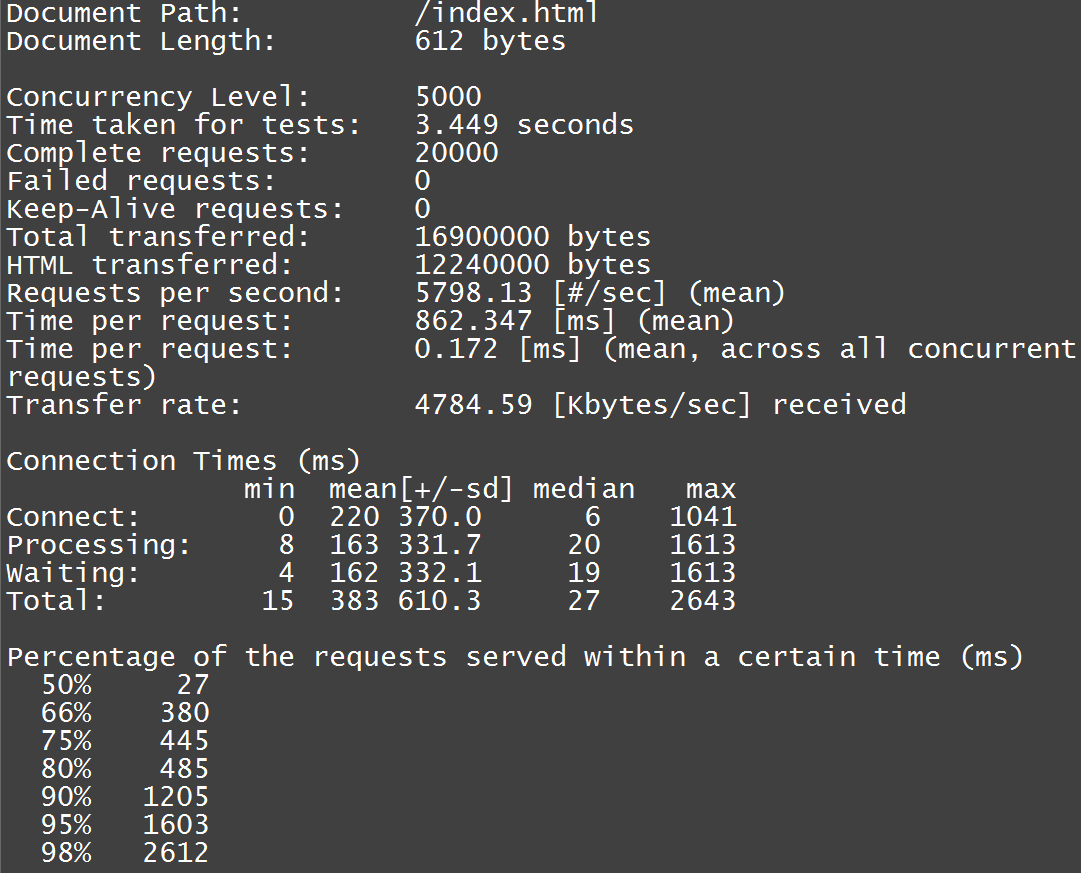


**(3) nginx单独压力测试**

启动nginx： **/usr/local/nginx/sbin/nginx**

启动ab压力测试工具：**/usr/local/apache/bin/ab -c 5000 -n 20000 -k http://192.168.8.102/index.html**

下面是虚拟机作为服务器的压力结果：



## 14.3 搭建php服务器

这里用一台虚拟机虚拟3个php服务器。

**(1) 安装独立进程运行的php软件，并且有连接mysql和memcache选项**

详细安装过程在《Centos系统环境搭建》的7.3和7.6节。

**(2) 配置php-fpm.conf**

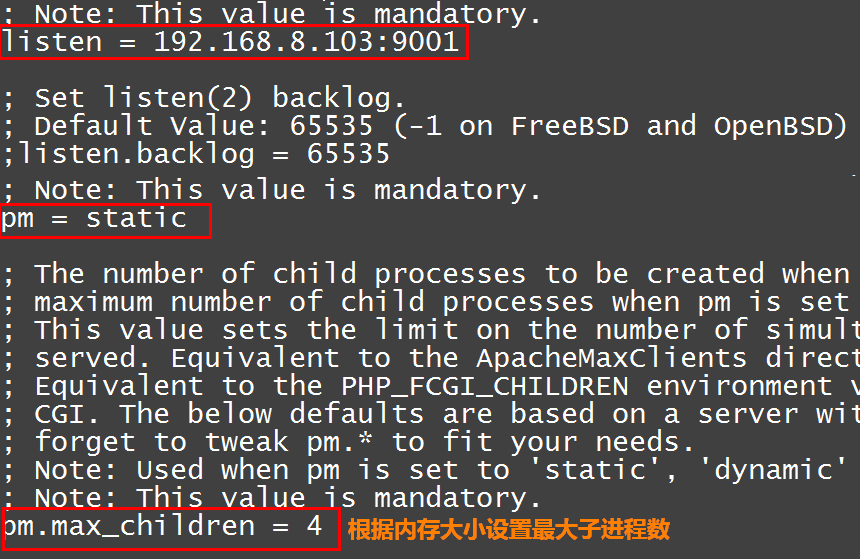
复制3分php-fpm.conf，名称分别为php-fpm-9001.conf、php-fpm-9002.conf、php-fpm-9003.conf

|  |
| --- |
| cd /usr/local/php/etc  cp phpfpm.conf php-fpm-9001.conf  cp phpfpm.conf php-fpm-9002.conf  cp phpfpm.conf php-fpm-9003.conf |

分别配置监听端口、子进程个数、子进程类型

**vim /usr/local/php/etc/php-fpm-9001.conf**

|  |
| --- |
| ;设置监听端口，注意使用本机ip，不使用localhost或127.0.0.1  listen = 192.168.8.103:9001  ;高并发时各个子进程会一直忙，不回收子进程，减少创建和回收子进程的过程，设置php子进程为静态，即使空闲不退出。  pm = static  ; 根据内存大小设置最大子进程数  pm.max\_children = 4 |



同理修改php-fpm-9002.conf、php-fpm-9003.conf配置文件。

**(3) 启动php-fpm软件**

因为启动虚拟了3个php进程，新建一个脚本文件来同时启动3个php进程。

**vim /usr/local/php/sbin/php.sh**

脚本内容如下：

|  |
| --- |
| #!/bin/bash  /usr/local/php/sbin/php-fpm -y /usr/local/php/etc/php-fpm-9001.conf  /usr/local/php/sbin/php-fpm -y /usr/local/php/etc/php-fpm-9002.conf  /usr/local/php/sbin/php-fpm -y /usr/local/php/etc/php-fpm-9003.conf |

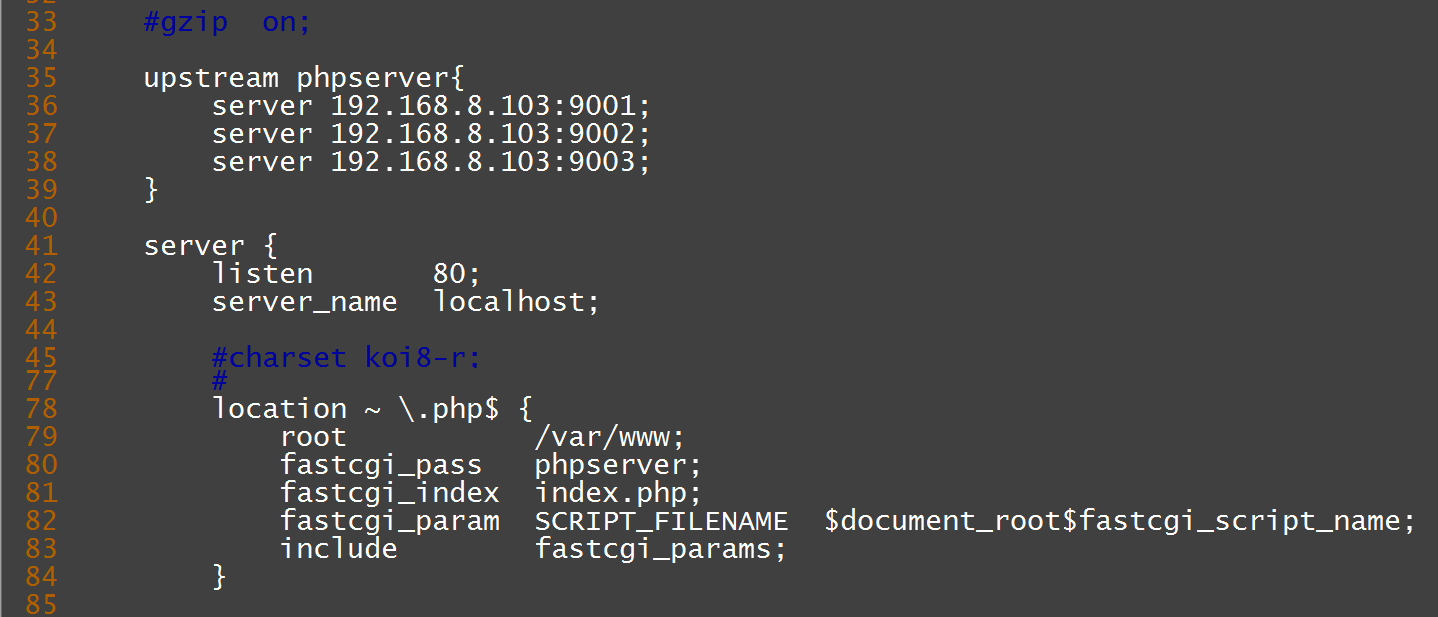
启动php进程：sh /usr/local/php/sbin/php.sh

**(4) 配置nginx服务器的nginx.conf文件**

**vim /usr/local/nginx/conf/nginx.conf**

|  |
| --- |
| #集群php服务器  upstream phpserver{  server 192.168.8.103:9001;  server 192.168.8.103:9002;  server 192.168.8.103:9003;  }  #php代理服务器  location ~ \.php$ {  root /var/www;  fastcgi\_pass phpserver;  fastcgi\_index index.php;  fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;  include fastcgi\_params;  } |

示例图如下：



重启nginx：**/usr/local/nginx/sbin/nginx -s reload**

**(5) 测试php多个子进程是否同时工作**

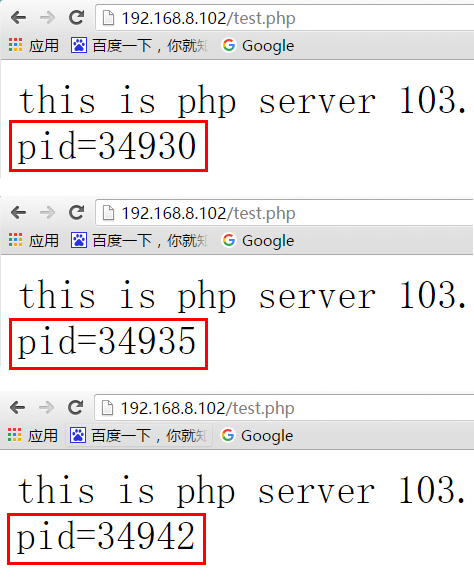
新建一个测试文件test.php，

**vim /var/www/test.php**

添加下面内容：

|  |
| --- |
| <?php  echo 'this is php server 103.<br>';  echo 'pid=',posix\_getpid(); // php子进程id号 |

在浏览器输入<http://192.168.8.102/test.php>，不断刷新网页，查看php子进程是否都在工作。如下图所示：



## 14.4 搭建memcached服务器

这里用一台虚拟机虚拟3个memcached服务器。

**(1) 重新安装nginx，添加一致性hash模块**

安装详细过程在12.2节。

**(2) 安装memcached缓存**

详细安装过程在《Centos系统环境搭建》的7.4节。

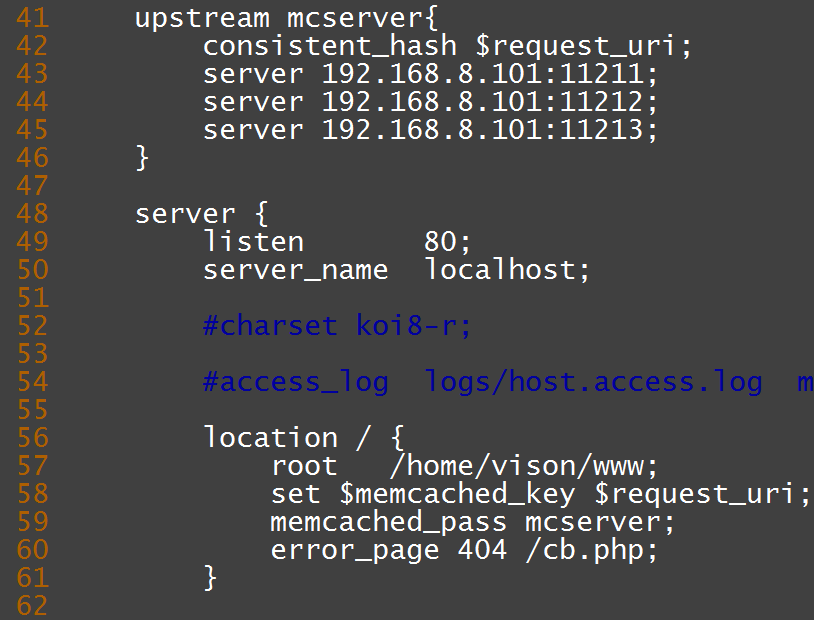
**(3) 在nginx服务器配置nginx.conf**

**vim /usr/local/nginx/conf/nginx.conf**

修改内容如下：

|  |
| --- |
| upstream mcserver{  consistent\_hash $request\_uri;  server 192.168.8.102:11211;  server 192.168.8.102:11212;  server 192.168.8.102:11213;  }  location / {  root /home/vison/www;  set $memcached\_key $request\_uri;  memcached\_pass mcserver;  error\_page 404 /cb.php;  } |

示图如下：



## 14.5 集群性能测试

**(1) 在php服务器下新建文件cb.php**

**vim /var/www/cb.php**

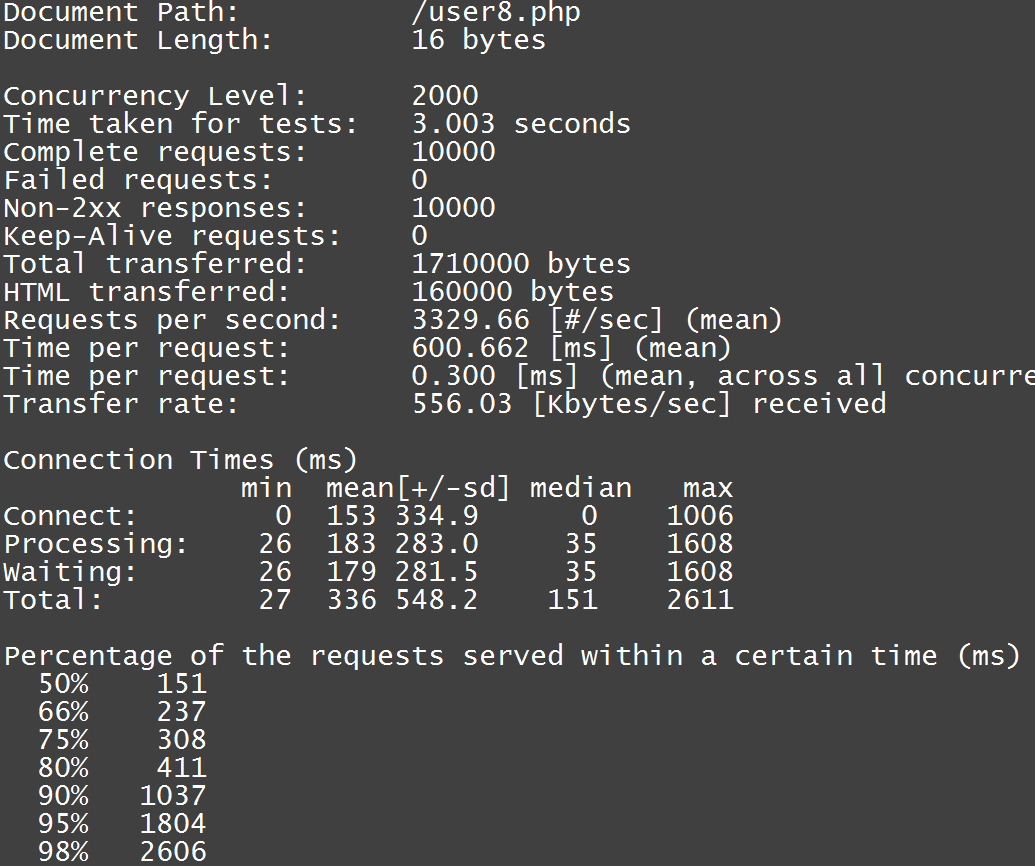
添加内容如下：

|  |
| --- |
| <?***php*** *// $uri作为key，包括/* $uri = $\_SERVER[**'REQUEST\_URI'**];    **echo 'key = '**,$uri,**', data frome mysql.<br>'**;    *// 获取用户uid* $uid = *substr*($uri,5,*strpos*($uri,**'.'**)-5);    *// 连接数据库* $mysqli = **new** mysqli(**'192.168.8.102'**, **'root'**, **'123456'**,**'test'**);  **if** (*mysqli\_connect\_errno*()){  **die**(**'Unable to connect!'**). *mysqli\_connect\_error*();  **return**;  }  $sql = **"set names utf8"**;  $mysqli->query($sql);    $sql = **"select** *\** **from user where id="**.$uid;  *print\_r*(getUser($mysqli,$sql));   $mysqli->close();   **function** getUser($mysqli, $sql){ *// 获取用户信息并写入缓存* **if** ($result = $mysqli->query($sql)){  **if**($row = $result->fetch\_assoc()){  sql2memcache($\_SERVER[**'REQUEST\_URI'**], $row);  $result->free();  **return** $row;  }  $result->free();  }**else**{  **echo 'error: "'**,$sql,**'"'**;  }  **return null**; }  *// memcache* **function** sql2memcache($key,$val){ *// 把数据库的数据添加到缓存中* $mem = **new** memcache();  $mem->addServer(**'192.168.8.101'**,11211);  $mem->addServer(**'192.168.8.101'**,11212);  $mem->addServer(**'192.168.8.101'**,11213);   **if**($mem->add($key,$val,0)){  **echo 'add "'**, $key, **'" to memcached ok!<br>'**;  }  $mem->close(); } |

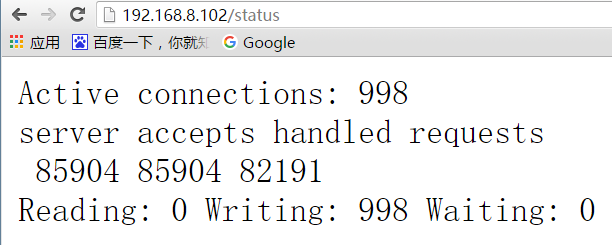
**(2) 测试**

由于使用虚拟机，各性能都会有很大影响。

**./ab -c 2000 -n 10000 http://192.168.8.102/user8.php**



使用浏览器查看nginx服务器状态http://192.168.8.102/status



遇到错误，首先查看error.log，查看错误原因，常见错误是文件不存在、上游服务器集群连接错误或文件不存在，检测ip和端口是否对应。

查看网络蜘蛛：http://baidu.com/robots.txt

https://www.taobao.com/robots.txt