



# 课程目标

掌握nginx+tomcat反向代理的使用方法。

掌握nginx作为负载均衡器的使用方法。

掌握nginx实现web缓存方法。

# nginx介绍

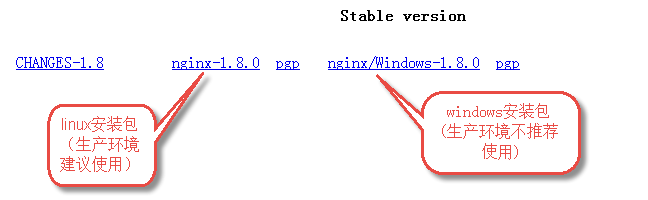
## 什么是nginx

Nginx是一款高性能的http 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器。官方测试nginx能够支支撑5万并发链接，并且cpu、内存等资源消耗却非常低，运行非常稳定。

# nginx安装

## 下载

进入http://nginx.org/en/download.html 下载nginx1.8.0版本（当前最新稳定版本）。





## 安装

### nginx安装环境

nginx是C语言开发，建议在linux上运行，本教程使用Centos6.5作为安装环境。

* gcc

安装nginx需要先将官网下载的源码进行编译，编译依赖gcc环境，如果没有gcc环境，需要安装gcc：yum install gcc-c++

* PCRE

PCRE(Perl Compatible Regular Expressions)是一个Perl库，包括 perl 兼容的正则表达式库。nginx的http模块使用pcre来解析正则表达式，所以需要在linux上安装pcre库。

**yum install -y pcre pcre-devel**

注：pcre-devel是使用pcre开发的一个二次开发库。nginx也需要此库。

* zlib

zlib库提供了很多种压缩和解压缩的方式，nginx使用zlib对http包的内容进行gzip，所以需要在linux上安装zlib库。

**yum install -y zlib zlib-devel**

* openssl

OpenSSL 是一个强大的安全套接字层密码库，囊括主要的密码算法、常用的密钥和证书封装管理功能及SSL协议，并提供丰富的应用程序供测试或其它目的使用。

nginx不仅支持http协议，还支持https（即在ssl协议上传输http），所以需要在linux安装openssl库。

**yum install -y openssl openssl-devel**

### 编译安装

将nginx-1.8.0.tar.gz拷贝至linux服务器。

解压：

tar -zxvf nginx-1.8.0.tar.gz

cd nginx-1.8.0

1. configure

./configure --help查询详细参数（参考本教程附录部分：nginx编译参数）

参数设置如下：

./configure \

--prefix=/usr/local/nginx \

--pid-path=/var/run/nginx/nginx.pid \

--lock-path=/var/lock/nginx.lock \

--error-log-path=/var/log/nginx/error.log \

--http-log-path=/var/log/nginx/access.log \

--with-http\_gzip\_static\_module \

--http-client-body-temp-path=/var/temp/nginx/client \

--http-proxy-temp-path=/var/temp/nginx/proxy \

--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \

--http-uwsgi-temp-path=/var/temp/nginx/uwsgi \

--http-scgi-temp-path=/var/temp/nginx/scgi

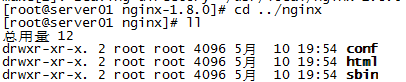
**注意：上边将临时文件目录指定为/var/temp/nginx，需要在/var下创建temp及nginx目录**

1. 编译安装

make

make install

安装成功查看安装目录 ：



## 启动nginx

cd /usr/local/nginx/sbin/

./nginx

查询nginx进程：



15098是nginx主进程的进程id，15099是nginx工作进程的进程id

**注意：执行./nginx启动nginx，这里可以-c指定加载的nginx配置文件，如下：**

**./nginx -c /usr/local/nginx/conf/nginx.conf**

**如果不指定-c，nginx在启动时默认加载conf/nginx.conf文件，此文件的地址也可以在编译安装nginx时指定./configure的参数（--conf-path= 指向配置文件（nginx.conf））**

## 停止nginx

方式1，快速停止：

cd /usr/local/nginx/sbin

./nginx -s stop

此方式相当于先查出nginx进程id再使用kill命令强制杀掉进程。

方式2，完整停止(建议使用)：

cd /usr/local/nginx/sbin

./nginx -s quit

此方式停止步骤是待nginx进程处理任务完毕进行停止。

## 重启nginx

方式1，先停止再启动（建议使用）：

对nginx进行重启相当于先停止nginx再启动nginx，即先执行停止命令再执行启动命令。

如下：

./nginx -s quit

./nginx

方式2，重新加载配置文件：

当nginx的配置文件nginx.conf修改后，要想让配置生效需要重启nginx，使用-s reload不用先停止nginx再启动nginx即可将配置信息在nginx中生效，如下：

./nginx -s reload

## 安装测试

nginx安装成功，启动nginx，即可访问虚拟机上的nginx：



到这说明nginx基本上安装成功。

# Nginx的rpm软件包安装

## 安装包在位置

D:\讲课内容--\新巴巴运动网\nginx高并发解决\nginx安装包



## 此种安装方式不用安装gcc等编译工具

## 安装命令如下

rpm –ivh nginx

# 配置虚拟主机

## 什么是虚拟主机

虚拟主机是一种特殊的软硬件技术，它可以将网络上的每一台计算机分成多个虚拟主机，每个虚拟主机可以独立对外提供www服务，这样就可以实现一台主机对外提供多个web服务，每个虚拟主机之间是独立的，互不影响的。

如下图：

一台物理服务器

www.itcast.cn

open.itcast.cn

......

通过nginx可以实现虚拟主机的配置，nginx支持三种类型的虚拟主机配置，1、基于ip的虚拟主机， 2、基于域名的虚拟主机 3、基于端口的虚拟主机

## 基于域名的虚拟主机配置

### 需求

两个域名指向同一台服务器，用户访问不同的域名显示不同的网页内容。

两个域名是aaa.test.com和bbb.test.com

一台服务器还使用虚拟机192.168.101.3代替

### 创建工程目录

创建/usr/local/aaa\_html，此目录为aaa.test.com域名访问的目录

创建/usr/local/bbb\_html，此目录为bbb.test.com域名访问的目录

目录中的内容使用nginx自带的html文件，将/usr/local/nginx/html中的内容拷贝分别拷贝到上边两个目录中，并且将aaa\_html目录中的index.html内容改为：“Welcome to aaa nginx!”

将bbb\_html目录中的index.html内容改为“Welcome to bbb nginx!”

### 虚拟主机配置

修改/usr/local/nginx/conf/nginx.conf文件，添加两个虚拟主机，如下：

#配置虚拟主机aaa.test.com

server {

#监听的ip和端口，配置本机ip和端口

listen 192.168.101.3:80;

#虚拟主机名称是aaa.test.com，请求域名aaa.test.com的url将由此server配置解析

server\_name aaa.test.com;

#所有的请求都以/开始，所有的请求都可以匹配此location

location / {

#使用root指令指定虚拟主机目录即网页存放目录

#比如访问http://ip/test.html将找到/usr/local/aaa\_html/test.html

#比如访问http://ip/item/test.html将找到/usr/local/aaa\_html/item/test.html

root /usr/local/aaa\_html;

#指定欢迎页面，按从左到右顺序查找

index index.html index.htm;

}

}

#配置虚拟主机bbb.test.com

server {

listen 192.168.101.3:80;

server\_name bbb.test.com;

location / {

root /usr/local/bbb\_html;

index index.html index.htm;

}

}

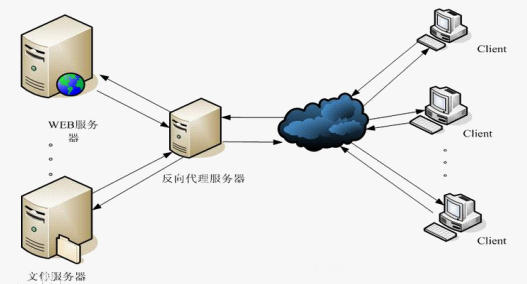
# nginx反向代理

## 什么是反向代理

通常的代理服务器，只用于代理内部网络对Internet的连接请求，客户机必须指定代理服务器,并将本来要直接发送到Web服务器上的http请求发送到代理服务器中由代理服务器向Internet上的web服务器发起请求，最终达到客户机上网的目的。

而反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

如下图：



## nginx+tomcat反向代理

### 需求

两个tomcat服务通过nginx反向代理，本例子使用三台虚拟机进行测试，

nginx服务器：192.168.101.3

tomcat1服务器：192.168.101.5

tomcat2服务器：192.168.101.6

如下图：

nginx 192.168.101.3

tomcat1 192.168.101.5

tomcat2 192.168.101.6

请求aaa.test.com

请求bbb.test.com

aaa.test.com由tomcat1提供服务

bbb.test.com由tomcat2提供服务

### 启动tomcat

tomcat使用apache-tomcat-7.0.57版本，在192.168.101.5和192.168.101.6虚拟机上启动tomcat。

### nginx反向代理配置

根据上边的需求在nginx.conf文件中配置反向代理，如下：

#配置一个代理即tomcat1服务器

upstream tomcat\_server1 {

server 192.168.101.5:8080;

}

#配置一个代理即tomcat2服务器

upstream tomcat\_server2 {

server 192.168.101.6:8080;

}

#配置一个虚拟主机

server {

listen 80;

server\_name aaa.test.com;

location / {

#域名aaa.test.com的请求全部转发到tomcat\_server1即tomcat1服务上

proxy\_pass http://tomcat\_server1;

#欢迎页面，按照从左到右的顺序查找页面

index index.jsp index.html index.htm;

}

}

server {

listen 80;

server\_name bbb.test.com;

location / {

#域名bbb.test.com的请求全部转发到tomcat\_server2即tomcat2服务上

proxy\_pass http://tomcat\_server2;

index index.jsp index.html index.htm;

}

}

### 测试

分别修改两个tomcat下的webapps/ROOT/index.jsp的内容，使用tomcat1和tomcat2两个服务首页显示不同的内容，如下：

tomcat1下的index.jsp修改后：



tomcat2下的index.jsp修改后：



分别访问aaa.test.com、bbb.test.com测试反向代理。

请求访问aaa.test.com通过nginx代理访问tomcat1，请求访问bbb.test.com通过nginx代理访问tomcat2。

# 负载均衡

## 什么是负载均衡

负载均衡 建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

负载均衡，英文名称为Load Balance，其意思就是分摊到多个操作单元上进行执行，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。

## nginx实现负载均衡

### 需求

nginx作为负载均衡服务器，用户请求先到达nginx，再由nginx根据负载配置将请求转发至 tomcat服务器。

nginx负载均衡服务器：192.168.101.3

tomcat1服务器：192.168.101.5

tomcat2服务器：192.168.101.6

nginx 192.168.101.3

负载均衡服务器

tomcat1 192.168.101.5

tomcat2 192.168.101.6

请求aaa.test.com

请求aaa.test.com

请求经过负载均衡至tomcat1

请求经过负载均衡至tomcat2

### 配置

根据上边的需求在nginx.conf文件中配置负载均衡，如下：

upstream tomcat\_server\_pool{

server 192.168.101.5:8080 weight=10;

server 192.168.101.6:8080 weight=10;

}

server {

listen 80;

server\_name aaa.test.com;

location / {

proxy\_pass http://tomcat\_server\_pool;

index index.jsp index.html index.htm;

}

}

### 测试

请求aaa.test.com，通过nginx负载均衡，将请求转发到tomcat服务器。

通过观察tomcat的访问日志或tomcat访问页面即可知道当前请求由哪个tomcat服务器受理。

# nginx负载均衡高可用

## 什么是负载均衡高可用

nginx作为负载均衡器，所有请求都到了nginx，可见nginx处于非常重点的位置，如果nginx服务器宕机后端web服务将无法提供服务，影响严重。

为了屏蔽负载均衡服务器的宕机，需要建立一个备份机。主服务器和备份机上都运行高可用（High Availability）监控程序，通过传送诸如“I am alive”这样的信息来监控对方的运行状况。当备份机不能在一定的时间内收到这样的信息时，它就接管主服务器的服务IP并继续提供负载均衡服务；当备份管理器又从主管理器收到“I am alive”这样的信息时，它就释放服务IP地址，这样的主服务器就开始再次提供负载均衡服务。

## keepalived+nginx实现主备

### 什么是keepalived

keepalived是集群管理中保证集群高可用的一个服务软件，用来防止单点故障。

Keepalived的作用是检测web服务器的状态，如果有一台web服务器死机，或工作出现故障，Keepalived将检测到，并将有故障的web服务器从系统中剔除，当web服务器工作正常后Keepalived自动将web服务器加入到服务器群中，这些工作全部自动完成，不需要人工干涉，需要人工做的只是修复故障的web服务器。

### keepalived工作原理

keepalived是以VRRP协议为实现基础的，VRRP全称Virtual Router Redundancy Protocol，即虚拟路由冗余协议。

虚拟路由冗余协议，可以认为是实现路由器高可用的协议，即将N台提供相同功能的路由器组成一个路由器组，这个组里面有一个master和多个backup，master上面有一个对外提供服务的vip（该路由器所在局域网内其他机器的默认路由为该vip），master会发组播，当backup收不到VRRP包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master。这样的话就可以保证路由器的高可用了。

keepalived主要有三个模块，分别是core、check和VRRP。core模块为keepalived的核心，负责主进程的启动、维护以及全局配置文件的加载和解析。check负责健康检查，包括常见的各种检查方式。VRRP模块是来实现VRRP协议的。

详细参考：Keepalived权威指南中文.pdf

### keepalived+nginx实现主备过程

#### 初始状态

VIP虚ip：192.168.101.100

nginx负载均衡服务器（主）

192.168.101.3

keepalived

nginx负载均衡服务器（备）

192.168.101.3

keepalived

心跳

tomat服务器群

#### 主机宕机

VIP虚ip：192.168.101.100

nginx负载均衡服务器（主）

192.168.101.3

keepalived

nginx负载均衡服务器（备）

192.168.101.3

keepalived

心跳

tomat服务器群

#### 主机恢复

VIP虚ip：192.168.101.100

nginx负载均衡服务器（主）

192.168.101.3

keepalived

nginx负载均衡服务器（备）

192.168.101.3

keepalived

心跳

tomat服务器群

### 高可用环境

两台nginx，一主一备：192.168.101.3和192.168.101.4

两台tomcat服务器：192.168.101.5、192.168.101.6

### 安装keepalived

分别在主备nginx上安装keepalived

D:\讲课内容--\新巴巴运动网\nginx高并发解决\高可用\keepalived\安装包

安装命令：

查看openssl的版本：要求必须是openssl-1.0.1e以上才行、如果版本已经符合、不用再安装openssl

（非必须）Rpm –Uvh openssl-1.0.1e-30.el6.8.i686.rpm

安装keepalived软件

rpm –ivh keepalived-1.2.13-5.el6\_6.i686.rpm

### 配置keepalived

#### 主nginx

修改主nginx下/etc/keepalived/keepalived.conf文件

! Configuration File for keepalived

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from XXX@XXX.com #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_instance VI\_1 {

state MASTER #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 100 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

#### 备nginx

修改备nginx下/etc/keepalived/keepalived.conf文件

**配置备nginx时需要注意：需要修改state为BACKUP , priority比MASTER低，virtual\_router\_id和master的值一致**

! Configuration File for keepalived

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from XXX@XXX.com #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_instance VI\_1 {

state BACKUP #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 99 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

### 测试

主备nginx都启动keepalived及nginx。

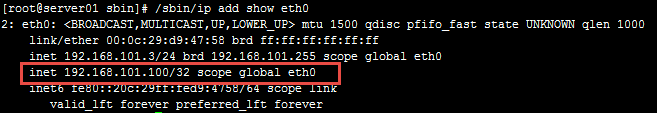
service keepalived start

./nginx

#### 初始状态

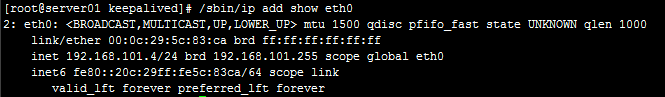
查看主nginx的eth0设置：

vip绑定在主nginx的eth0上。



查看备nginx的eth0设置：

vip没有绑定在备nginx的eth0上。

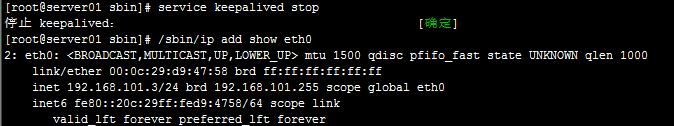


访问ccc.test.com，可以访问。

#### 主机宕机

将主nginx的keepalived停止或将主nginx关机(相当于模拟宕机)，查看主nginx的eth0：

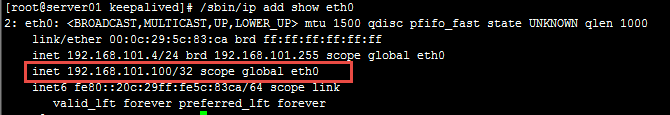
eth0没有绑定vip



注意这里模拟的是停止 keepalived进程没有模拟宕机，所以还要将nginx进程也停止表示主nginx服务无法提供。

查看备nginx的eth0：

vip已经漂移到备nginx。



访问ccc.test.com，可以访问。

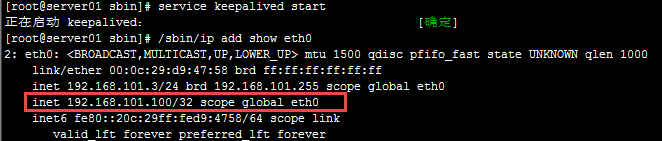
#### 主机恢复

将主nginx的keepalived和nginx都启动。

查看主nginx的eth0：

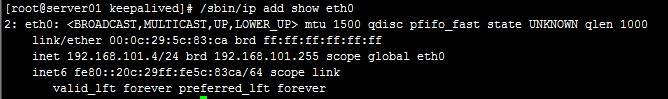
查看备nginx的eth0：

vip漂移到主nginx。



查看备nginx的eth0：

eth0没有绑定vip



访问：ccc.test.com，正常访问。

**注意：主nginx恢复时一定要将nginx也启动（通常nginx启动要加在开机启动中），否则即使vip漂移到主nginx也无法访问。**

### 解决nginx进程和keepalived不同时存在问题

#### 问题描述

keepalived是通过检测keepalived进程是否存在判断服务器是否宕机，如果keepalived进程在但是nginx进程不在了那么keepalived是不会做主备切换，所以我们需要写个脚本来监控nginx进程是否存在，如果nginx不存在就将keepalived进程杀掉。

#### nginx进程检测脚本

在主nginx上需要编写nginx进程检测脚本（check\_nginx.sh），判断nginx进程是否存在，如果nginx不存在就将keepalived进程杀掉，check\_nginx.sh内容如下：

#!/bin/bash

# 如果进程中没有nginx则将keepalived进程kill掉

A=`ps -C nginx --no-header |wc -l` ## 查看是否有 nginx进程 把值赋给变量A

if [ $A -eq 0 ];then ## 如果没有进程值得为 零

service keepalived stop ## 则结束 keepalived 进程

fi

将check\_nginx.sh拷贝至/etc/keepalived下，

脚本测试：

将nginx停止，将keepalived启动，执行脚本：sh /etc/keepalived/check\_nginx.sh



从执行可以看出自动将keepalived进程kill掉了。

#### 修改keepalived.conf

修改主nginx的keepalived.conf，添加脚本定义检测：

注意下边红色标识地方：

#全局配置

global\_defs {

notification\_email { #指定keepalived在发生切换时需要发送email到的对象，一行一个

XXX@XXX.com

}

notification\_email\_from miaoruntu@itcast.cn #指定发件人

#smtp\_server XXX.smtp.com #指定smtp服务器地址

#smtp\_connect\_timeout 30 #指定smtp连接超时时间

router\_id LVS\_DEVEL #运行keepalived机器的一个标识

}

vrrp\_script check\_nginx {

script "/etc/keepalived/check\_nginx.sh" ##监控脚本

interval 2 ##时间间隔，2秒

weight 2 ##权重

}

vrrp\_instance VI\_1 {

state MASTER #标示状态为MASTER 备份机为BACKUP

interface eth0 #设置实例绑定的网卡

virtual\_router\_id 51 #同一实例下virtual\_router\_id必须相同

priority 100 #MASTER权重要高于BACKUP 比如BACKUP为99

advert\_int 1 #MASTER与BACKUP负载均衡器之间同步检查的时间间隔，单位是秒

authentication { #设置认证

auth\_type PASS #主从服务器验证方式

auth\_pass 8888

}

track\_script {

check\_nginx #监控脚本

}

virtual\_ipaddress { #设置vip

192.168.101.100 #可以多个虚拟IP，换行即可

}

}

修改后重启keepalived

#### 测试

回到负载均衡高可用的初始状态，保证主、备上的keepalived、nginx全部启动。

停止主nginx服务

观察keepalived日志：

tail -f /var/log/keepalived.log



查看keepalived进程已经不存在。

查看eth0已经没有绑定vip。

# nginx实现web缓存

## 什么是nginx的web缓存

http的request和response过程即客户端发起一个url请求到web服务器，web服务器向客户端响应，如果让客户端先请求给一个缓存服务器由缓存服务器中间转发请求至web服务器，web服务器向客户端响应也是通过缓存服务器给客户端响应这个就是nginx的代理功能，nginx不仅有代理功能还有缓存功能，它可以将web服务器响应的信息缓存至内存或磁盘中，当用户再次发起相同的url时nginx就不用请求缓存服务器而直接从缓存中取出响应给客户端，从而大大提高请求响应的效率。如下图：

http客户端

web服务器

nginx缓存服务器

proxy\_cache

缓存区域

## proxy\_cache缓存过程

proxy\_cache是nginx内置的一个缓存模块，它用于反向代理时对后端web服务的内容进行缓存。**这里需要注意proxy\_cache只在使用反向代理proxy\_pass时进行缓存**。

proxy\_cache缓存过程如下：

1、客户端第一次请求url，比如请求ccc.test.com/index.html

2、nginx计算url的哈希值

md5(url)

即：MD5('ccc.test.com/index.html')，得到b5ac8d82f6ef789fe7081ef4a43f9230

3、创建缓存目录，写入缓存数据

如果缓存目录的规则设置为 levels=1:2，创建目录如下：

取出b5ac8d82f6ef789fe7081ef4a43f9230的最后一位0作为目录名创建目录，再取出0前边的两位23作为目录名在0目录下边创建目录，即一级目录为0，二级目录名为23，最后将代理获得的http响应数据存储在缓存数据文件中，缓存文件命名为b5ac8d82f6ef789fe7081ef4a43f9230写入0/23目录下。

4、客户端再次请求相同的url，对url进行哈希得到缓存数据文件的地址，如果找到缓存数据则直接返回给客户端不再请求web服务器。

## 需求

1、对静态文件进行缓存：html,js,css,png,gif,jpg,jpeg,bmp,swf。

2、对符合url规则的请求进行缓存。

3、针对某个url进行清除缓存。

## 安装ngx\_cache\_purge

ngx\_cache\_purge是第三方开发的一个nginx模块，不包含在 Nginx 的源码发布版，它的作用是清理nginx缓存，对FastCGI、proxy、SCGI 和uWSGI缓存进行清除。

### 解压ngx\_cache\_purge

将ngx\_cache\_purge-2.3.tar.gz拷贝至/usr/local/下，

tar -zxvf ngx\_cache\_purge-2.3.tar.gz

### nginx添加ngx\_cache\_purge模块

解压nginx-1.8.0至/usr/local/

cd /usr/local/

tar -zxvf nginx-1.8.0.tar.gz

进入nginx-1.8.0目录

cd nginx-1.8.0

下边的配置参数注意最后一行添加add-module=/usr/local/ngx\_cache\_purge-2.3

./configure \

--prefix=/usr/local/nginx \

--pid-path=/var/run/nginx/nginx.pid \

--lock-path=/var/lock/nginx.lock \

--error-log-path=/var/log/nginx/error.log \

--http-log-path=/var/log/nginx/access.log \

--with-http\_gzip\_static\_module \

--http-client-body-temp-path=/var/temp/nginx/client \

--http-proxy-temp-path=/var/temp/nginx/proxy \

--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \

--http-uwsgi-temp-path=/var/temp/nginx/uwsgi \

--http-scgi-temp-path=/var/temp/nginx/scgi \

--add-module=/usr/local/ngx\_cache\_purge-2.3

编译、安装

make

make install

## 缓存配置

http {

include mime.types;

default\_type application/octet-stream;

#log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

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

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

#access\_log logs/access.log main;

sendfile on;

#tcp\_nopush on;

#keepalive\_timeout 0;

keepalive\_timeout 65;

#gzip on;

upstream tomcat\_server\_pool{

server 192.168.101.5:8080 weight=10;

server 192.168.101.6:8080 weight=10;

}

#proxy\_temp\_path代理临时目录

proxy\_temp\_path /var/temp/nginx/proxy;

#proxy\_cache\_path代理缓存目录，和proxy\_temp\_path必须在同一个分区

#/var/temp/nginx/proxy\_cache\_dir缓存目录

#levels指定该缓存空间有两层hash目录，第一层目录名是1个字母或数字长度，第二层目录名为2个字母或数字长度

#keys\_zone=cache\_one:50m缓存区名称为cache\_one，在内存中的空间是50M，inactive=1d表示1天清空一次缓存 ，max\_size指定磁盘空间大小为500M

proxy\_cache\_path /var/temp/nginx/proxy\_cache\_dir levels=1:2 keys\_zone=cache\_one:50m inactive=1d max\_size=500m;

server {

listen 80;

server\_name ccc.test.com;

#清空缓存配置，注意必须放在最上边，当在请求的url前加/purge/时将此url的缓存清空

location ~ /purge(/.\*)

{

#安全设置，指定请求客户端的IP或IP段才可以清除URL缓存，这里为了方便测试设置为all

#allow 127.0.0.1;

allow all;

#指定清空缓存的区域名称cache\_one(要和上边proxy\_cache\_path缓存配置中指定的缓存区域名称一致)

#指定缓存的key规则$host$1$is\_args$args，要和下边设置缓存的key一致$host$uri$is\_args$args

#注意$host$1$is\_args$args中的$1表示当前请求的uri，$host$1$is\_args$args=$host$uri$is\_args$args

proxy\_cache\_purge cache\_one $host$1$is\_args$args;

}

#所有的location规则都不满足走/，使用proxy\_pass将请求转发给tomcat

location /{

proxy\_pass http://tomcat\_server\_pool;

index index.jsp index.html index.htm;

}

#请求url以/item/开头的进行缓存，使用此location

location ~ /item(/.\*){

#指定缓存区域名称

proxy\_cache cache\_one;

#以域名、URI、参数组合成Web缓存的Key值，Nginx根据Key值哈希

proxy\_cache\_key $host$uri$is\_args$args;

#请求头中添加请求的主机名

proxy\_set\_header Host $host;

#请求头中添加真实的客户端ip，通过X-Forwarded-For可获取

proxy\_set\_header X-Forwarded-For $remote\_addr;

#代理访问后端tomcat

#请求item从缓存中找，如果缓存中没有则向tomcat请求

#$1表示取出正则表达式(/.\*)所匹配的内容，使用$1的效果例如请求http://ccc.test.com/item/100010.html则请求tomcat服务器 http://ip:port/100010.html

#如果不使用$1则会将/item/...加在tomcat服务地址之后访问，即http://ip:port/item/100010.html

proxy\_pass http://tomcat\_server\_pool$1;

}

#请求url以.html、js、css、png、gif结尾和此location匹配成功

location ~ \.( html|js|css|png|gif|jpg|jpeg|bmp|swf)$ {

#如果后端的服务器返回502、504、执行超时等错误，自动将请求转发到 upstream负载均衡池中的另一台服务器，实现故障转移。

proxy\_next\_upstream http\_502 http\_504 error timeout invalid\_header;

proxy\_cache cache\_one;

#对不同的HTTP状态码设置不同的缓存时间

proxy\_cache\_valid 200 10m;

proxy\_cache\_valid 304 1m;

proxy\_cache\_valid 301 302 1h;

proxy\_cache\_valid any 1m;

#以域名、URI、参数组合成Web缓存的Key值，Nginx根据Key值哈希

proxy\_cache\_key $host$uri$is\_args$args;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $remote\_addr;

#如果没有缓存则通过proxy\_pass转向tomcat请求

proxy\_pass http://tomcat\_server\_pool;

}

}

## 测试

### 第一次请求

http://ccc.test.com/item/docs/

观察后台tomcat的访问日志



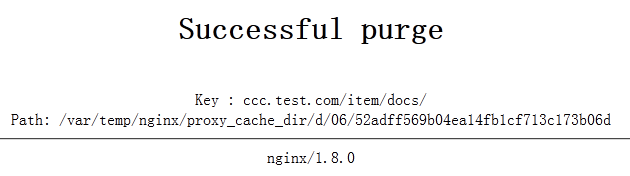
### 第二次请求

http://ccc.test.com/item/docs/

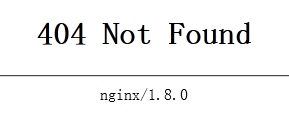
观察后台tomcat的访问日志，没有发现请求日志

### 清空缓存

在浏览器请求：http://ccc.test.com/purge/item/docs/



说明清除缓存成功，当再次刷新此地址则报404说明没有找到要清除的缓存文件：



### 清除缓存后再次请求

请求：http://ccc.test.com/item/docs/

观察后台tomcat的访问日志



根据预期基本正确又请求了/docs/，但是为什么没有第一次请求日志中的：



因为在nginx.conf中配置了对静态资源文件的缓存，所以/docs/images/asf-logo.gif也被缓存，刚才清除缓存只是清除了http://ccc.test.com/item/docs/的缓存没有清除/docs/images/asf-logo.gif的缓存。

### 图片缓存清除

下边对/docs/images/asf-logo.gif的缓存清除：

请求http://ccc.test.com/purge/item/docs/images/asf-logo.gif



### 再次请求

请求：http://ccc.test.com/item/docs/

观察后台tomcat的访问日志



思考下这次怎么没有/docs/的请求日志？？

# 附录

## nginx编译参数

--prefix= 指向安装目录

--sbin-path 指向（执行）程序文件（nginx）

--conf-path= 指向配置文件（nginx.conf）

--error-log-path= 指向错误日志目录

--pid-path= 指向pid文件（nginx.pid）

--lock-path= 指向lock文件（nginx.lock）（安装文件锁定，防止安装文件被别人利用，或自己误操作。）

--user= 指定程序运行时的非特权用户

--group= 指定程序运行时的非特权用户组

--builddir= 指向编译目录

--with-rtsig\_module 启用rtsig模块支持（实时信号）

--with-select\_module 启用select模块支持（一种轮询模式,不推荐在高载环境下使用）禁用：--without-select\_module

--with-poll\_module 启用poll模块支持（功能与select相同，与select特性相同，为一种轮询模式,不推荐在高载环境下使用）

--with-file-aio 启用file aio支持（一种APL文件传输格式）

--with-ipv6 启用ipv6支持

--with-http\_ssl\_module 启用ngx\_http\_ssl\_module支持（使支持https请求，需已安装openssl）

--with-http\_realip\_module 启用ngx\_http\_realip\_module支持（这个模块允许从请求标头更改客户端的IP地址值，默认为关）

--with-http\_addition\_module 启用ngx\_http\_addition\_module支持（作为一个输出过滤器，支持不完全缓冲，分部分响应请求）

--with-http\_xslt\_module 启用ngx\_http\_xslt\_module支持（过滤转换XML请求）

--with-http\_image\_filter\_module 启用ngx\_http\_image\_filter\_module支持（传输JPEG/GIF/PNG 图片的一个过滤器）（默认为不启用。gd库要用到）

--with-http\_geoip\_module 启用ngx\_http\_geoip\_module支持（该模块创建基于与MaxMind GeoIP二进制文件相配的客户端IP地址的ngx\_http\_geoip\_module变量）

--with-http\_sub\_module 启用ngx\_http\_sub\_module支持（允许用一些其他文本替换nginx响应中的一些文本）

--with-http\_dav\_module 启用ngx\_http\_dav\_module支持（增加PUT,DELETE,MKCOL：创建集合,COPY和MOVE方法）默认情况下为关闭，需编译开启

--with-http\_flv\_module 启用ngx\_http\_flv\_module支持（提供寻求内存使用基于时间的偏移量文件）

--with-http\_gzip\_static\_module 启用ngx\_http\_gzip\_static\_module支持（在线实时压缩输出数据流）

--with-http\_random\_index\_module 启用ngx\_http\_random\_index\_module支持（从目录中随机挑选一个目录索引）

--with-http\_secure\_link\_module 启用ngx\_http\_secure\_link\_module支持（计算和检查要求所需的安全链接网址）

--with-http\_degradation\_module 启用ngx\_http\_degradation\_module支持（允许在内存不足的情况下返回204或444码）

--with-http\_stub\_status\_module 启用ngx\_http\_stub\_status\_module支持（获取nginx自上次启动以来的工作状态）

--without-http\_charset\_module 禁用ngx\_http\_charset\_module支持（重新编码web页面，但只能是一个方向--服务器端到客户端，并且只有一个字节的编码可以被重新编码）

--without-http\_gzip\_module 禁用ngx\_http\_gzip\_module支持（该模块同-with-http\_gzip\_static\_module功能一样）

--without-http\_ssi\_module 禁用ngx\_http\_ssi\_module支持（该模块提供了一个在输入端处理处理服务器包含文件（SSI）的过滤器，目前支持SSI命令的列表是不完整的）

--without-http\_userid\_module 禁用ngx\_http\_userid\_module支持（该模块用来处理用来确定客户端后续请求的cookies）

--without-http\_access\_module 禁用ngx\_http\_access\_module支持（该模块提供了一个简单的基于主机的访问控制。允许/拒绝基于ip地址）

--without-http\_auth\_basic\_module禁用ngx\_http\_auth\_basic\_module（该模块是可以使用用户名和密码基于http基本认证方法来保护你的站点或其部分内容）

--without-http\_autoindex\_module 禁用disable ngx\_http\_autoindex\_module支持（该模块用于自动生成目录列表，只在ngx\_http\_index\_module模块未找到索引文件时发出请求。）

--without-http\_geo\_module 禁用ngx\_http\_geo\_module支持（创建一些变量，其值依赖于客户端的IP地址）

--without-http\_map\_module 禁用ngx\_http\_map\_module支持（使用任意的键/值对设置配置变量）

--without-http\_split\_clients\_module 禁用ngx\_http\_split\_clients\_module支持（该模块用来基于某些条件划分用户。条件如：ip地址、报头、cookies等等）

--without-http\_referer\_module 禁用disable ngx\_http\_referer\_module支持（该模块用来过滤请求，拒绝报头中Referer值不正确的请求）

--without-http\_rewrite\_module 禁用ngx\_http\_rewrite\_module支持（该模块允许使用正则表达式改变URI，并且根据变量来转向以及选择配置。如果在server级别设置该选项，那么他们将在 location之前生效。如果在location还有更进一步的重写规则，location部分的规则依然会被执行。如果这个URI重写是因为location部分的规则造成的，那么 location部分会再次被执行作为新的URI。 这个循环会执行10次，然后Nginx会返回一个500错误。）

--without-http\_proxy\_module 禁用ngx\_http\_proxy\_module支持（有关代理服务器）

--without-http\_fastcgi\_module 禁用ngx\_http\_fastcgi\_module支持（该模块允许Nginx 与FastCGI 进程交互，并通过传递参数来控制FastCGI 进程工作。 ）FastCGI一个常驻型的公共网关接口。

--without-http\_uwsgi\_module 禁用ngx\_http\_uwsgi\_module支持（该模块用来医用uwsgi协议，uWSGI服务器相关）

--without-http\_scgi\_module 禁用ngx\_http\_scgi\_module支持（该模块用来启用SCGI协议支持，SCGI协议是CGI协议的替代。它是一种应用程序与HTTP服务接口标准。它有些像FastCGI但他的设计更容易实现。）

--without-http\_memcached\_module 禁用ngx\_http\_memcached\_module支持（该模块用来提供简单的缓存，以提高系统效率）

-without-http\_limit\_zone\_module 禁用ngx\_http\_limit\_zone\_module支持（该模块可以针对条件，进行会话的并发连接数控制）

--without-http\_limit\_req\_module 禁用ngx\_http\_limit\_req\_module支持（该模块允许你对于一个地址进行请求数量的限制用一个给定的session或一个特定的事件）

--without-http\_empty\_gif\_module 禁用ngx\_http\_empty\_gif\_module支持（该模块在内存中常驻了一个1\*1的透明GIF图像，可以被非常快速的调用）

--without-http\_browser\_module 禁用ngx\_http\_browser\_module支持（该模块用来创建依赖于请求报头的值。如果浏览器为modern ，则$modern\_browser等于modern\_browser\_value指令分配的值；如果浏览器为old，则$ancient\_browser等于 ancient\_browser\_value指令分配的值；如果浏览器为 MSIE中的任意版本，则 $msie等于1）

--without-http\_upstream\_ip\_hash\_module 禁用ngx\_http\_upstream\_ip\_hash\_module支持（该模块用于简单的负载均衡）

--with-http\_perl\_module 启用ngx\_http\_perl\_module支持（该模块使nginx可以直接使用perl或通过ssi调用perl）

--with-perl\_modules\_path= 设定perl模块路径

--with-perl= 设定perl库文件路径

--http-log-path= 设定access log路径

--http-client-body-temp-path= 设定http客户端请求临时文件路径

--http-proxy-temp-path= 设定http代理临时文件路径

--http-fastcgi-temp-path= 设定http fastcgi临时文件路径

--http-uwsgi-temp-path= 设定http uwsgi临时文件路径

--http-scgi-temp-path= 设定http scgi临时文件路径

-without-http 禁用http server功能

--without-http-cache 禁用http cache功能

--with-mail 启用POP3/IMAP4/SMTP代理模块支持

--with-mail\_ssl\_module 启用ngx\_mail\_ssl\_module支持

--without-mail\_pop3\_module 禁用pop3协议（POP3即邮局协议的第3个版本,它是规定个人计算机如何连接到互联网上的邮件服务器进行收发邮件的协议。是因特网电子邮件的第一个离线协议标准,POP3协议允许用户从服务器上把邮件存储到本地主机上,同时根据客户端的操作删除或保存在邮件服务器上的邮件。POP3协议是TCP/IP协议族中的一员，主要用于支持使用客户端远程管理在服务器上的电子邮件）

--without-mail\_imap\_module 禁用imap协议（一种邮件获取协议。它的主要作用是邮件客户端可以通过这种协议从邮件服务器上获取邮件的信息，下载邮件等。IMAP协议运行在TCP/IP协议之上，使用的端口是143。它与POP3协议的主要区别是用户可以不用把所有的邮件全部下载，可以通过客户端直接对服务器上的邮件进行操作。）

--without-mail\_smtp\_module 禁用smtp协议（SMTP即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式。SMTP协议属于TCP/IP协议族，它帮助每台计算机在发送或中转信件时找到下一个目的地。）

--with-google\_perftools\_module 启用ngx\_google\_perftools\_module支持（调试用，剖析程序性能瓶颈）

--with-cpp\_test\_module 启用ngx\_cpp\_test\_module支持

--add-module= 启用外部模块支持

--with-cc= 指向C编译器路径

--with-cpp= 指向C预处理路径

--with-cc-opt= 设置C编译器参数（PCRE库，需要指定–with-cc-opt=”-I /usr/local/include”，如果使用select()函数则需要同时增加文件描述符数量，可以通过–with-cc- opt=”-D FD\_SETSIZE=2048”指定。）

--with-ld-opt= 设置连接文件参数。（PCRE库，需要指定–with-ld-opt=”-L /usr/local/lib”。）

--with-cpu-opt= 指定编译的CPU，可用的值为: pentium, pentiumpro, pentium3, pentium4, athlon, opteron, amd64, sparc32, sparc64, ppc64

--without-pcre 禁用pcre库

--with-pcre 启用pcre库

--with-pcre= 指向pcre库文件目录

--with-pcre-opt= 在编译时为pcre库设置附加参数

--with-md5= 指向md5库文件目录（消息摘要算法第五版，用以提供消息的完整性保护）

--with-md5-opt= 在编译时为md5库设置附加参数

--with-md5-asm 使用md5汇编源

--with-sha1= 指向sha1库目录（数字签名算法，主要用于数字签名）

--with-sha1-opt= 在编译时为sha1库设置附加参数

--with-sha1-asm 使用sha1汇编源

--with-zlib= 指向zlib库目录

--with-zlib-opt= 在编译时为zlib设置附加参数

--with-zlib-asm= 为指定的CPU使用zlib汇编源进行优化，CPU类型为pentium, pentiumpro

--with-libatomic 为原子内存的更新操作的实现提供一个架构

--with-libatomic= 指向libatomic\_ops安装目录

--with-openssl= 指向openssl安装目录

--with-openssl-opt 在编译时为openssl设置附加参数

--with-debug 启用debug日志