Nginx构建反向代理缓存服务器

代理服务可简单的分为**正向代理**和**反向代理:**

正向代理: 用于代理内部网络对 Internet 的连接请求(如 VPN/NAT),客户端指定代理服务器,并将本来要直接发送给目标 Web 服务器的 HTTP 请求先发送到代理服务器上, 然后由代理服务器去访问 Web 服务器, 并将 Web 服务器的 Response 回传给客户端:

反向代理: 与正向代理相反,如果局域网向 Internet 提供资源,并让 Internet 上的其他用户可以访问局域网内资源, 也可以设置一个代理服务器, 它提供的服务就是反向代理. 反向代理服务器接受来自 Internet 的连接,然后将请求转发给内部网络上的服务器,并将 Response 回传给

Internet 上请求连接的客户端:

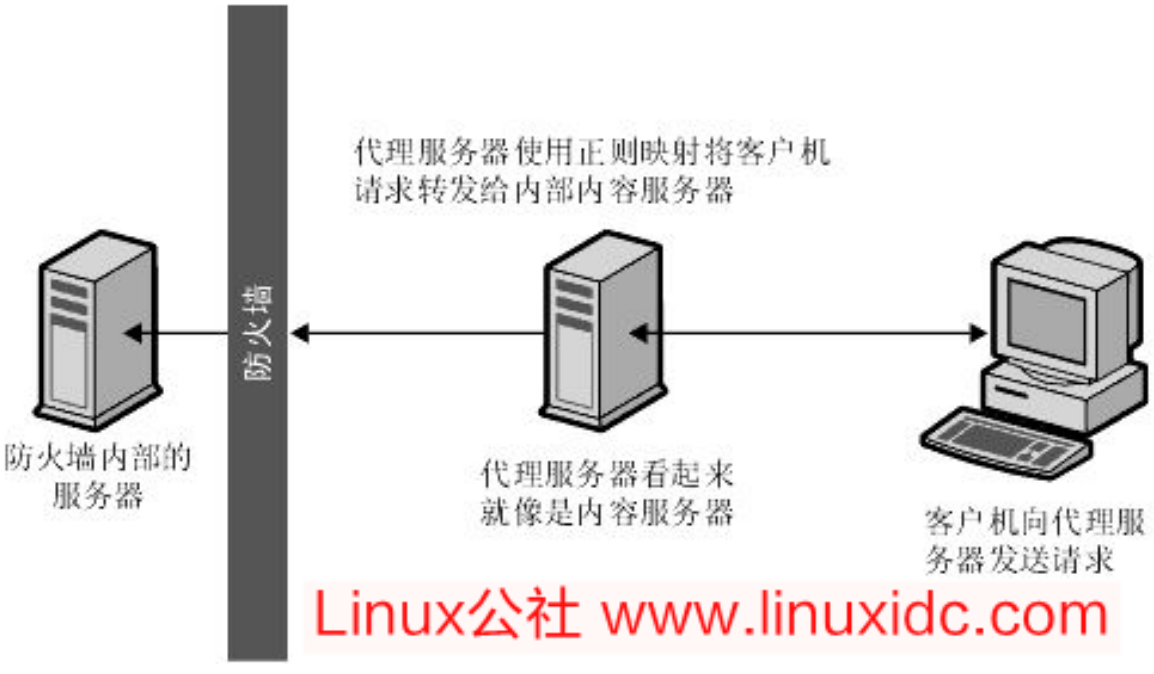
一、Nginx 反向代理：Web 服务器的调度器

1、反向代理（Reverse Proxy）方式是指以代理服务器来接受客户端的连接请求，然后将请求转发给网络上的 web 服务器（可能是 apache、nginx、tomcat、iis 等），并将从 web 服务器上得到的结果返回给请求连接的客户端，此时代理服务器对外就表现为一个服务器。

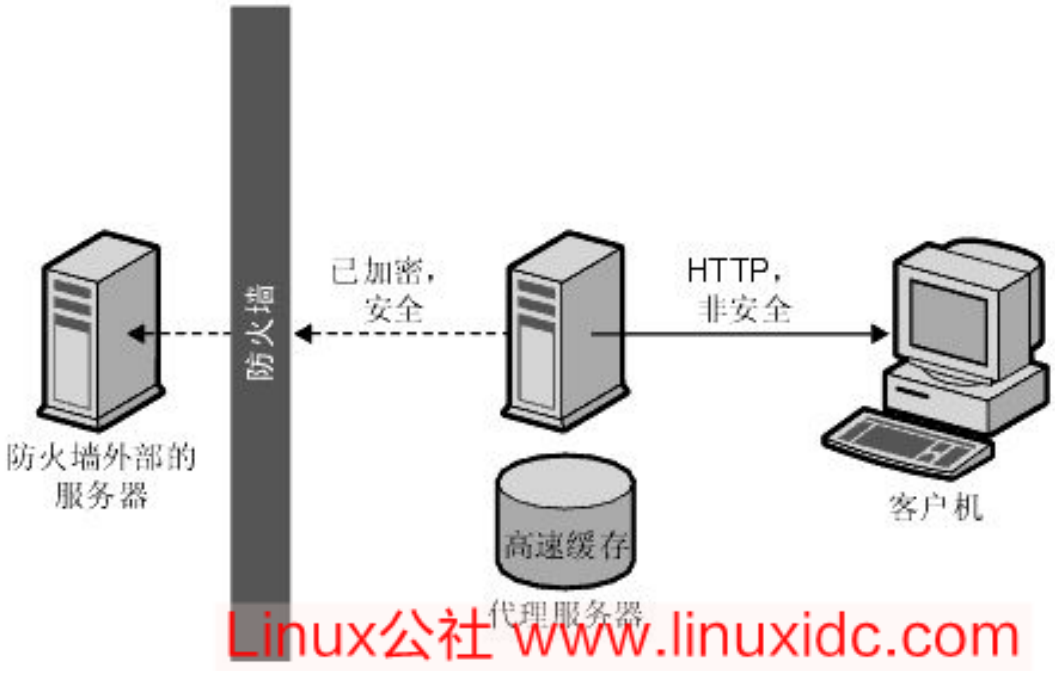
从上图可以看出：反向代理服务器代理网站 Web 服务器接收 Http 请求，对请求进行转发。而且 nginx 作为反向代理服务器可以根据用户请求的内容把请求转发给后端不同的 web 服务器，例如**静动分离**，再例如在 nginx 上创建多个虚拟主机，这样就成功的做到了在浏览器中输入不同域名（url）的时候访问后端的不同 web 服务器或 web 群集。

2、反向代理的作用？

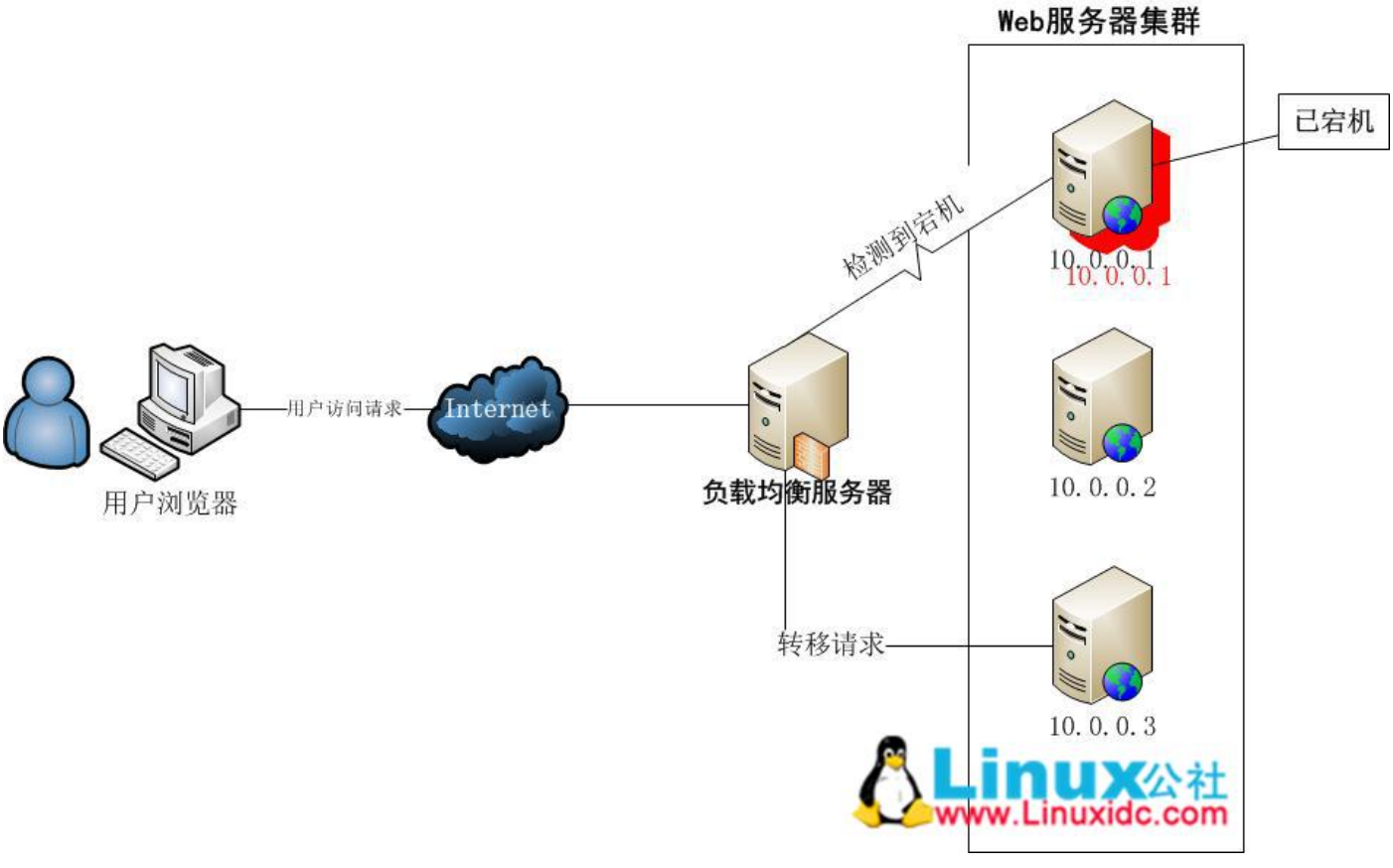
①保护网站安全：任何来自 Internet 的请求都必须先经过代理服务器；



②通过配置缓存功能加速 Web 请求：可以缓存真实 Web 服务器上的某些静态资源，减轻真实 Web 服务器的负载压力；



③实现负载均衡：充当负载均衡服务器均衡地分发请求，平衡集群中各个服务器的负载压力；



----------分割线 ----------

CentOS 6.2 实战部署 Nginx+MySQL+PHP http://www.linuxidc.com/Linux/2013-09/90020.htm

使用 Nginx 搭建 WEB 服务器 http://www.linuxidc.com/Linux/2013-09/89768.htm

搭建基于 Linux6.3+Nginx1.2+PHP5+MySQL5.5 的 Web 服务器全过程 http://www.linuxidc.com/Linux/2013-09/89692.htm

CentOS 6.3 下 Nginx 性能调优 http://www.linuxidc.com/Linux/2013-09/89656.htm

CentOS 6.3 下配置 Nginx 加载 ngx_pagespeed 模块 http://www.linuxidc.com/Linux/2013-09/89657.htm

CentOS 6.4 安装配置 Nginx+Pcre+php-fpm http://www.linuxidc.com/Linux/2013-08/88984.htm

Nginx 安装配置使用详细笔记 http://www.linuxidc.com/Linux/2014-07/104499.htm

Nginx 日志过滤 使用 ngx_log_if 不记录特定日志 http://www.linuxidc.com/Linux/2014-07/104686.htm

CentOS 上安装 Nginx 服务器实现虚拟主机和域名重定向 http://www.linuxidc.com/Linux/2017-04/142642.htm

Nginx 的详细介绍：请点这里
Nginx 的下载地址：请点这里

----------分割线 ----------

二、什么是 nginx

1、nginx 简介

Nginx 是一款轻量级的网页服务器、反向代理器以及电子邮件代理服务器。因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。Nginx（发音同 engine x），它是由俄罗斯程序员 Igor Sysoev 所开发的。起初是供俄国大型的门户网站及搜索引擎 Rambler（俄语：Рамблер）使用。此软件 BSD-like 协议下发行，可以在 UNIX、GNU/Linux、BSD、Mac OS X、Solaris，以及 Microsoft Windows 等操作系统中运行。

Nginx 的应用现状

Nginx 已经在俄罗斯最大的门户网站——Rambler Media（www.rambler.ru）上运行，同时俄罗斯超过 20%的虚拟主机平台采用 Nginx 作为反向代理服务器。

在国内，已经有 淘宝、新浪博客、新浪播客、网易新闻、六间房、56.com、Discuz!、水木社区、豆瓣、YUPOO、海内、迅雷在线 等多家网站使用 Nginx 作为 Web 服务器或反向代理服务器。

2、Nginx 的核心特点

（1）跨平台：Nginx 可以在大多数 OS 编译运行，而且也有 Windows 的版本；

（2）配置异常简单：非常容易上手。

（3）非阻塞、高并发连接：官方测试能够支撑 5 万并发连接，在实际生产环境中跑到 2~3 万并发连接数。（这得益于 Nginx 使用了最新的 epoll 模型）；

注：

对于一个 Web 服务器来说，首先看一个请求的基本过程：建立连接—接收数据—发送数据，在系统底层看来 ：上述过程（建立连接—接收数据—发送数据）在系统底层就是读写事件。

如果采用阻塞调用的方式，当读写事件没有准备好时，那么就只能等待，当前线程被挂起，等事件准备好了，才能进行读写事件。

如果采用非阻塞调用的方式：事件马上返回，告诉你事件还没准备好呢，过会再来吧。过一会，再来检查一下事件，直到事件准备好了为止，在这期间，你就可以先去做其它事情，然后再来看看事件好了没。虽然不阻塞了，但你得不时地过来检查一下事件的状态，你可以做更多的事情了，但带来的开销也是不小的。非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程

（4）事件驱动：通信机制采用 epoll 模型，支持更大的并发连接。

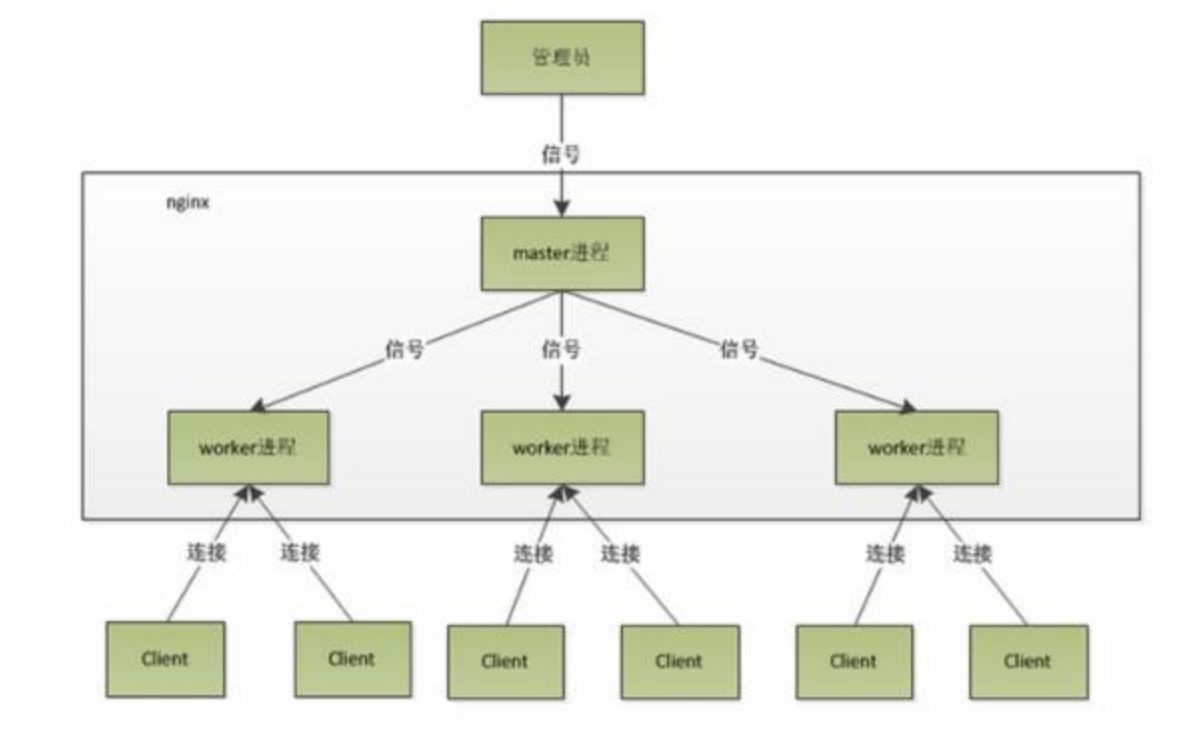
非阻塞通过不断检查事件的状态来判断是否进行读写操作，这样带来的开销很大，因此就有了异步非阻塞的事件处理机制。这种机制让你可以同时监控多个事件，调用他们是非阻塞的，但可以设置超时时间，在超时时间之内，如果有事件准备好了，就返回。这种机制解决了上面阻塞调用与非阻塞调用的两个问题。

以 epoll 模型为例：当事件没有准备好时，就放入 epoll(队列)里面。如果有事件准备好了，那么就去处理；当事件没有准备好时，才在 epoll 里面等着。这样，我们就可以并发处理大量的并发了，当然，这里的并发请求，是指未处理完的请求。线程只有一个，所以同时能处理的请求当然只有一个了，只是在请求之间进行不断地切换而已，切换也是因为异步事件未准备好，而主动让出的。这里的切换是没有任何代价，你可以理解为循环处理多个准备好的事件。

多线程方式相比，这种事件处理方式是有很大的优势的，不需要创建线程，每个请求占用的内存也很少，没有上下文切换， 事件处理非常的轻量级，并发数再多也不会导致无谓的资源浪费（上下文切换）。对于 apache 服务器，每个请求会独占一个工作线程，当并发数上到几千时，就同时有几千的线程在处理请求了。这对操作系统来说，是个不小的挑战：因为线程带来的内存占用非常大，线程的上下文切换带来的 cpu 开销很大，自然性能就上不 去，从而导致在高并发场景下性能下降严重。

总结：通过异步非阻塞的事件处理机制，Nginx 实现由进程循环处理多个准备好的事件，从而实现高并发和轻量级。

（5）Master/Worker 结构：一个 master 进程，生成一个或多个 worker 进程。



注：Master-Worker 设计模式主要包含两个主要组件 Master 和 Worker，Master 维护着 Worker 队列，将请求下发到多个 Worker 并行执行，Worker 主要进行实际逻辑计算，并将结果返回给 Master。

nginx 采用这种进程模型有什么好处？采用独立的进程，可以让互相之间不会影响，一个进程退出后，其它进程还在工作，服务不会中断，Master 进程则很快重新启动新的 Worker 进程。当然，Worker 进程的异常退出，肯定是程序有 bug 了，异常退出，会导致当前 Worker 上的所有请求失败，不过不会影响到所有请求，所以降低了风险。

（6）**内存消耗小**：处理大并发的请求内存消耗非常小。在 3 万并发连接下，开启的 10 个 Nginx 进程才消耗 150M 内存（15M*10=150M）。

（7）**内置的健康检查功能**：如果 Nginx 代理的后端的某台 Web 服务器宕机了，不会影响前端访问。

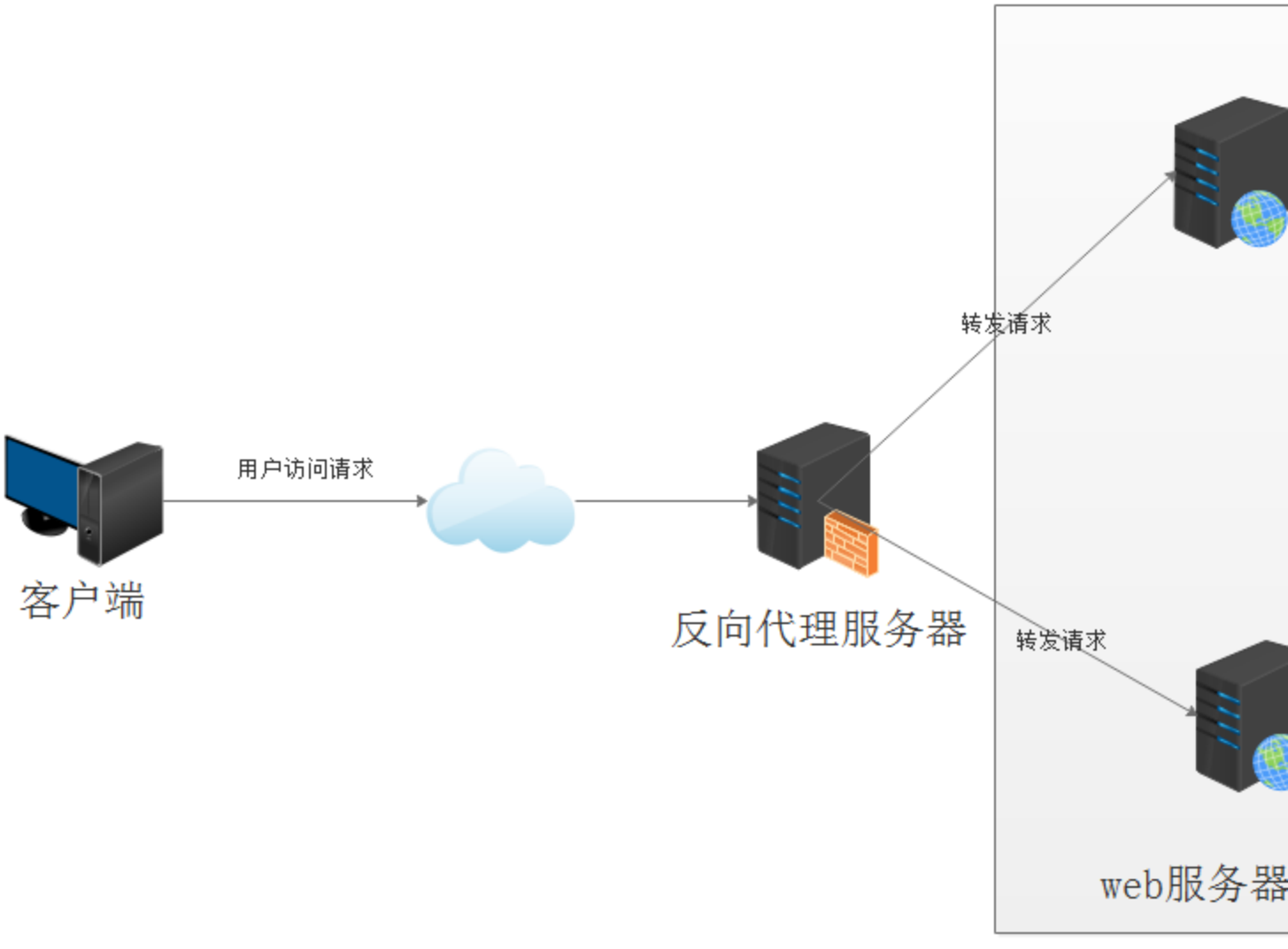
（8）**节省带宽**：支持 GZIP 压缩，可以添加浏览器本地缓存的 Header 头。

（9）**稳定性高**：用于反向代理，宕机的概率微乎其微。

三、Nginx+apache 构筑 Web 服务器集群的负载均衡

nginx 配置反向代理

配置 nginx 作为反向代理和负载均衡，同时利用其缓存功能，将静态页面在 nginx 缓存，以达到降低后端服务器连接数的目的并检查后端 web 服务器的健康状况。



1、安装 nginx

环境：

OS:centos7.2

nginx：192.168.31.83

apache1:192.168.31.141

apache2:192.168.31.250

安装 zlib-devel、pcre-devel 等依赖包

[root@www ~]# yum -y install gcc gcc-c++ make libtool zlib zlib-devel pcre pcre-devel openssl openssl-devel

注：

结合 proxy 和 upstream 模块实现后端 web 负载均衡

使用 proxy 模块实现静态文件缓存

结合 nginx 默认自带的 ngx_http_proxy_module 模块 和 ngx_http_upstream_module 模块

实现后端服务器的健康检查，也可以使用第三方模块 nginx_upstream_check_module

使用 nginx-sticky-module 扩展模块实现 Cookie 会话黏贴（保持会话）

使用 ngx_cache_purge 实现更强大的缓存清除功能

上面提到的 2 个模块都属于第三方扩展模块，需要提前下好源码，然后编译时通过--add-moudle=src_path 一起安装。

安装 nginx

[root@www ~]# groupadd www　　#添加 www 组

[root@www ~]# useradd -g www www -s /sbin/nologin　　#创建 nginx 运行账户 www 并加入到 www 组，不允许 www 用户直接登录系统

#tar zxf nginx-1.10.2.tar.gz

#tar zxf ngx_cache_purge-2.3.tar.gz

#tar zxf master.tar.gz

cd nginx-1.10.2/

[root@www nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.10 --user=www --group=www --with-http_stub_status_module --with-http_realip_module --with-http_ssl_module --with-http_gzip_static_module --http-client-body-temp-path=/var/tmp/nginx/client --http-proxy-temp-path=/var/tmp/nginx/proxy --http-fastcgi-temp-path=/var/tmp/nginx/fcgi --with-pcre --add-module=../ngx_cache_purge-2.3 --with-http_flv_module --add-module=../nginx-goodies-nginx-sticky-module-ng-08a395c66e42

[root@www nginx-1.10.2]# make && make install

注：nginx 的所有模块必须在编译的时候添加，不能再运行的时候动态加载。

优化 nginx 程序的执行路径

[root@www nginx-1.10.2]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/

[root@www nginx-1.10.2]# nginx -t

[root@www nginx-1.10.2]# mkdir -p /var/tmp/nginx/client

[root@www nginx-1.10.2]# chown -R www:www /var/tmp/nginx/

[root@www nginx-1.10.2]# nginx -t

nginx: the configuration file /usr/local/nginx1.10/conf/nginx.conf syntax is ok

nginx: configuration file /usr/local/nginx1.10/conf/nginx.conf test is successful

2、编写 nginx 服务脚本：

[root@www ~]# vi /etc/init.d/nginx　　内容如下：

#!/bin/bash

chkconfig: 2345 99 20

```
# description: Nginx Service Control Script

PROG="/usr/local/nginx1.10/sbin/nginx"

PIDF="/usr/local/nginx1.10/logs/nginx.pid"

case "$1" in

 start)

  netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /dev/null

  if [ $? -eq 0 ]

  then

   echo "Nginx service already running."

  else

   $PROG -t &> /dev/null

   if [ $? -eq 0 ] ; then

    $PROG

    echo "Nginx service start success."

   else

   $PROG -t

   fi

  fi

  ;;

 stop)

  netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /dev/null

  if [ $? -eq 0 ]

  then

   kill -s QUIT $(cat $PIDF)

   echo "Nginx service stop success."

  else

   echo "Nginx service already stop"

  fi

  ;;
```

```
restart)
 $0 stop
 $0 start
 ;;
status)
 netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /dev/null
 if [ $? -eq 0 ]
 then
  echo "Nginx service is running."
 else
  echo "Nginx is stop."
 fi
;;
reload)
 netstat -anplt |grep ":80" &> /dev/null && pgrep "nginx" &> /dev/null
 if [ $? -eq 0 ]
 then
  $PROG -t &> /dev/null
  if [ $? -eq 0 ] ; then
   kill -s HUP $(cat $PIDF)
   echo "reload Nginx config success."
  else
   $PROG -t
  fi
 else
  echo "Nginx service is not run."
 fi
 ;;
*)
```

```
  echo "Usage: $0 {start|stop|restart|reload}"
  exit 1
esac
[root@www ~]# chmod +x /etc/init.d/nginx
[root@www ~]# chkconfig --add nginx
[root@www ~]# chkconfig nginx on
[root@www ~]# service  nginx start
Nginx service start success.
[root@www ~]# service  nginx status
Nginx service is running.
[root@www ~]# netstat -anpt | grep nginx
tcp       0      0 0.0.0.0:80            0.0.0.0:*               LISTEN      3977/nginx: master
[root@www ~]# firewall-cmd --permanent --add-port=80/tcp
success
[root@www ~]# firewall-cmd --reload
Success
```

注：如果你想在已安装好的 nginx 上添加第三方模块，依然需要重新编译，但为了不覆盖你原有的配置，请不要 make install，而是直接拷贝可执行文件：

```
# nginx –V
[root@www nginx-1.10.2]#./configure       --add-module=……          #你的第三方模块
[root@www nginx-1.10.2]   #make 后不要 make install,改为手动拷贝，先备份
[root@www nginx-1.10.2]   #cp /usr/local/nginx1.10/sbin/nginx /usr/local/nginx1.10/sbin/nginx.bak
[root@www nginx-1.10.2]   #cp objs/nginx /usr/local/nginx1.10/sbin/nginx
```

配置 nginx 反向代理：反向代理+负载均衡+健康探测

查看 nginx 加载的模块

```
[root@www ~]## nginx -V
nginx version: nginx/1.10.2
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-4) (GCC)
built with OpenSSL 1.0.1e-fips 11 Feb 2013
```

TLS SNI support enabled

configure arguments: --prefix=/usr/local/nginx1.10 --user=www --group=www --with-http_stub_status_module --with-http_realip_module --with-http_ssl_module --with-http_gzip_static_module --http-client-body-temp-path=/var/tmp/nginx/client --http-proxy-temp-path=/var/tmp/nginx/proxy --http-fastcgi-temp-path=/var/tmp/nginx/fcgi --with-pcre --add-module=../ngx_cache_purge-2.3 --with-http_flv_module --add-module=../nginx-goodies-nginx-sticky-module-ng-08a395c66e42

nginx 的所有模块必须在编译的时候添加，不能再运行的时候动态加载。

3、nginx-sticky-module 模块：

这个模块的作用是通过 cookie 黏贴的方式将来自同一个客户端（浏览器）的请求发送到同一个后端服务器上处理，这样一定程度上可以解决多个 backend servers 的 session 同步的问题 —— 因为不再需要同步，而 RR 轮询模式必须要运维人员自己考虑 session 同步的实现。

另外内置的 ip_hash 也可以实现根据客户端 IP 来分发请求，但它很容易造成负载不均衡的情况，而如果 nginx 前面有 CDN 网络或者来自同一局域网的访问，它接收的客户端 IP 是一样的，容易造成负载不均衡现象。nginx-sticky-module 的 cookie 过期时间，默认浏览器关闭就过期。

这个模块并不合适不支持 Cookie 或手动禁用了 cookie 的浏览器，此时默认 sticky 就会切换成 RR。它不能与 ip_hash 同时使用。

```
upstream backend {

    server 192.168.31.141:80 weight=1;

    server 192.168.31.250:80 weight=1;

                sticky;

}
```

配置起来超级简单，一般来说一个 sticky 指令就够了。

相关信息可以查看官方文档 https://bitbucket.org/nginx-goodies/nginx-sticky-module-ng

4、load-balance 其它调度方案：

这里顺带介绍一下 nginx 的负载均衡模块支持的其它调度算法：

轮询（默认）： 每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某台服务器宕机，故障系统被自动剔除，使用户访问不受影响。Weight 指定轮询权值，Weight 值越大，分配到的访问机率越高，主要用于后端每个服务器性能不均的情况下。

ip_hash： 每个请求按访问 IP 的 hash 结果分配，这样来自同一个 IP 的访客固定访问一个后端服务器，有效解决了动态网页存在的 session 共享问题。当然如果这个节点不可用了，会发到下个节点，而此时没有 session 同步的话就注销掉了。

least_conn： 请求被发送到当前活跃连接最少的 realserver 上。会考虑 weight 的值。

url_hash： 此方法按访问 url 的 hash 结果来分配请求，使每个 url 定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率。Nginx 本身是不支持 url_hash 的，如果需要使用这种调度算法，必须安装 Nginx 的 hash 软件包 nginx_upstream_hash 。

fair：这是比上面两个更加智能的负载均衡算法。此种算法可以依据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间来分配请求，响应时间短的优先分配。Nginx 本身是不支持 fair 的，如果需要使用这种调度算法，必须下载 Nginx 的 upstream_fair 模块。

5、负载均衡与健康检查：

严格来说，nginx 自带是没有针对负载均衡后端节点的健康检查的，但是可以通过默认自带的 ngx_http_proxy_module 模块和 ngx_http_upstream_module 模块中的相关指令来完成当后端节点出现故障时，自动切换到下一个节点来提供访问。

```
upstream backend {
                    sticky;
    server 192.168.31.141:80 weight=1 max_fails=2 fail_timeout=10s;
    server 192.168.31.250:80 weight=1 max_fails=2 fail_timeout=10s;
}
server {
        ……
    location / {
        proxy_pass http://backend;
    }
    ……
}
```

weight： 轮询权值也是可以用在 ip_hash 的，默认值为 1

max_fails： 允许请求失败的次数，默认为 1。当超过最大次数时，返回 proxy_next_upstream 模块定义的错误。

fail_timeout： 有两层含义，一是在 10s 时间内最多容许 2 次失败；二是在经历了 2 次失败以后，10s 时间内不分配请求到这台服务器。

6、nginx 的 proxy 缓存使用：

缓存也就是将 js、css、image 等静态文件从后端服务器缓存到 nginx 指定的缓存目录下，既可以减轻后端服务器负担，也可以加快访问速度，但这样缓存及时清理成为了一个问题，所以需要 ngx_cache_purge 这个模块来在过期时间未到之前，手动清理缓存。

proxy 模块中常用的指令时 proxy_pass 和 proxy_cache.

nginx 的 web 缓存功能的主要是由 proxy_cache、fastcgi_cache 指令集和相关指令集完成，proxy_cache 指令负责反向代理缓存后端服务器的静态内容，fastcgi_cache 主要用来处理 FastCGI 动态进程缓存。

```
http {
        #$upstream_cache_status 记录缓存命中率
```

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"'
        '"$upstream_cache_status"';
access_log  logs/access.log  main;
proxy_buffering on;        #代理的时候，开启或关闭缓冲后端服务器的响应
proxy_temp_path /usr/local/nginx1.10/proxy_temp;
    proxy_cache_path /usr/local/nginx1.10/proxy_cache levels=1:2 keys_zone=my-
  cache:100m    inactive=600m    max_size=2g;
     server {
listen      80;
server_name  localhost;
root   html;
index  index.php index.html index.htm;
                #ngx_cache_purge 实现缓存清除
                location  ~/purge(/.*) {
  allow 127.0.0.1;
  allow 192.168.31.0/24;
  deny all;
  proxy_cache_purge my-cache $host$1$is_args$args;
}
            location ~ .*\.(gif|jpg|png|html|htm|css|js|ico|swf|pdf)(.*) {
  proxy_pass  http://backend;
  proxy_redirect off;
  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                    proxy_ignore_headers Set-Cookie;
        proxy_hide_header Set-Cookie;
```

```
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
        proxy_cache my-cache;
        add_header Nginx-Cache $upstream_cache_status;
        proxy_cache_valid 200 304 301 302 8h;
        proxy_cache_valid 404 1m;
        proxy_cache_valid any 1d;
        proxy_cache_key $host$uri$is_args$args;
        expires 30d;
    }
```

相关选项说明：

proxy_buffering on; 代理的时候，开启或关闭缓冲后端服务器的响应。

当开启缓冲时，nginx 尽可能快地从被代理的服务器接收响应，再将它存入缓冲区中。

proxy_temp_path： 缓存临时目录。后端的响应并不直接返回客户端，而是先写到一个临时文件中，然后被 rename 一下当做缓存放在 proxy_cache_path 。0.8.9 版本以后允许 temp 和 cache 两个目录在不同文件系统上（分区），然而为了减少性能损失还是建议把它们设成一个文件系统上。

proxy_cache_path： 设置缓存目录，目录里的文件名是 cache_key 的 MD5 值。

levels=1:2 keys_zone=my-cache:100m 表示采用 2 级目录结构，第一层目录只有一个字符，是由 levels=1:2 设置，总共二层目录，子目录名字由二个字符组成。Web 缓存区名称为 my-cache，内存缓存空间大小为 100MB，这个缓冲 zone 可以被多次使用。文件系统上看到的缓存文件名类似于 /usr/local/nginx1.10/proxy_cache/c/29/b7f54b2df7773722d382f4809d65029c 。

inactive=600 max_size=2g 表示 600 分钟没有被访问的内容自动清除，硬盘最大缓存空间为 2GB，超过这个大学将清除最近最少使用的数据。

需要在默认情况，nginx 不缓存从后端响应的 http 头中带有 Set-Cookie 的对象。如果客户端发送的请求带有 Cookie header，varnish 将忽略缓存，直接将请求传递到后端。nginx 中通过 proxy_ignore_headers 设置忽略它们，设置方法如下：

解决办法：

```
proxy_ignore_headers Set-Cookie;
proxy_hide_header Set-Cookie;
```

proxy_cache： 引用前面定义的缓存区 my-cache

proxy_cache_key：定义如何生成缓存的键，设置 web 缓存的 key 值，nginx 根据 key 值 md5 哈希存储缓存

proxy_cache_valid： 为不同的响应状态码设置不同的缓存时间，比如 200、302 等正常结果可以缓存的时间长点，而 404、500 等缓存时间设置短一些，这个时间到了文件就会过期，而不论是否刚被访问过。

add_header 指令来设置 response header， 语法：add_header name value;

$upstream_cache_status 这个变量来显示缓存的状态，我们可以在配置中添加一个 http 头来显示这一状态，

$upstream_cache_status 包含以下几种状态：
·MISS 未命中，请求被传送到后端
·HIT 缓存命中
·EXPIRED 缓存已经过期请求被传送到后端
·UPDATING 正在更新缓存，将使用旧的应答
·STALE 后端将得到过期的应答

expires： 在响应头里设置 Expires:或 Cache-Control:max-age，返回给客户端的浏览器缓存失效时间。

下面的 nginx.conf 实现 nginx 在前端做反向代理服务器的完整配置文件的例子，处理 js、png 等静态文件，jsp/php 等动态请求转发到其它服务器 tomcat/apache

```
user  www www;

worker_processes  4;

worker_cpu_affinity 0001 0010 0100 1000;

error_log  logs/error.log;

#error_log  logs/error.log  notice;

#error_log  logs/error.log  info;

worker_rlimit_nofile 10240;

pid        logs/nginx.pid;

events {

   use epoll;

   worker_connections  4096;

}

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"'

                '"$upstream_cache_status"';
```

```
    access_log  logs/access.log  main;

    server_tokens off;

sendfile        on;

#tcp_nopush     on;

#keepalive_timeout  0;

keepalive_timeout  65;

#Compression Settings

gzip on;

gzip_comp_level 6;

gzip_http_version 1.1;

gzip_proxied any;

gzip_min_length 1k;

gzip_buffers 16 8k;

gzip_types text/plain text/css text/javascript application/json application/javascript application/x-javascript
application/xml;

gzip_vary on;

#end gzip

# http_proxy Settings

client_max_body_size   10m;

client_body_buffer_size   128k;

proxy_connect_timeout   75;

proxy_send_timeout   75;

proxy_read_timeout   75;

proxy_buffer_size   4k;

proxy_buffers   4 32k;

proxy_busy_buffers_size   64k;

    proxy_temp_file_write_size  64k;

    proxy_buffering on;

proxy_temp_path /usr/local/nginx1.10/proxy_temp;
```

```
    proxy_cache_path /usr/local/nginx1.10/proxy_cache levels=1:2 keys_zone=my-cache:100m max_size=1000m
inactive=600m max_size=2g;

    #load balance Settings

    upstream backend {

        sticky;

        server 192.168.31.141:80 weight=1 max_fails=2 fail_timeout=10s;

        server 192.168.31.250:80 weight=1 max_fails=2 fail_timeout=10s;

    }

    #virtual host Settings

    server {

        listen       80;

        server_name  localhost;

        charset utf-8;

        location  ~/purge(/.*) {

          allow 127.0.0.1;

          allow 192.168.31.0/24;

          deny all;

          proxy_cache_purge my-cache $host$1$is_args$args;

        }

        location / {

          index  index.php index.html index.htm;

          proxy_pass       http://backend;

          proxy_redirect off;

          proxy_set_header  Host  $host;

          proxy_set_header  X-Real-IP  $remote_addr;

          proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;

                              proxy_ignore_headers Set-Cookie;

                    proxy_hide_header Set-Cookie;

          proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;

        }
```

```
    location ~ .*\.(gif|jpg|png|html|htm|css|js|ico|swf|pdf)(.*) {
      proxy_pass  http://backend;
      proxy_redirect off;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
      proxy_cache my-cache;
      add_header Nginx-Cache $upstream_cache_status;
      proxy_cache_valid 200 304 301 302 8h;
      proxy_cache_valid 404 1m;
      proxy_cache_valid any 1d;
      proxy_cache_key $host$uri$is_args$args;
      expires 30d;
    }
    location /nginx_status {
      stub_status on;
      access_log off;
      allow 192.168.31.0/24;
      deny all;
    }
  }
}
```

常用指令说明：

main 全局配置：

woker_processes 4

在配置文件的顶级 main 部分，worker 角色的工作进程的个数，master 进程是接收并分配请求给 worker 处理。这个数值简单一点可以设置为 cpu 的核数 grep ^processor /proc/cpuinfo | wc -l，也是 auto 值，如果开启了 ssl 和 gzip 更应该设置成与逻辑 CPU 数量一样甚至为 2 倍，可以减少 I/O 操作。如果 nginx 服务器还有其它服务，可以考虑适当减少。

worker_cpu_affinity
也是写在 main 部分。在高并发情况下，通过设置 cpu 粘性来降低由于多 CPU 核切换造成的寄存器等现场重建带来的性能损耗。如 worker_cpu_affinity 0001 0010 0100 1000;（四核）。

附：

CPU 工作状况：（输入 top 后，按 1 查看）

```
[root@www ~]# top
top - 22:56:36 up  2:31,  2 users,  load average: 0.00, 0.01, 0.05
Tasks: 443 total,   2 running, 441 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3866948 total,  2754648 free,   555544 used,   556756 buff/cache
KiB Swap:  4063228 total,  4063228 free,        0 used.  3031288 avail Mem
```

上面的配置表示：4 核 CPU，开启 4 个进程。0001 表示开启第一个 cpu 内核， 0010 表示开启第二个 cpu 内核，依次类推；有多少个核，就有几位数，1 表示该内核开启，0 表示该内核关闭。

例如：

1、2 核 CPU，开启 2 个进程

worker_processes 2;

worker_cpu_affinity 01 10;

2、2 核 CPU，开启 4 进程

worker_processes 4;

worker_cpu_affinity 01 10 01 10;

3、2 核 CPU，开启 8 进程

worker_processes 8;

worker_cpu_affinity 01 10 01 10 01 10 01 10;

4、8 核 CPU，开启 2 进程

worker_processes 2;

worker_cpu_affinity 10101010 01010101;

说明：10101010 表示开启了第 2,4,6,8 内核，01010101 表示开始了 1,3,5,7 内核

通过 apache 的 ab 测试查看 nginx 对 CPU 的使用状况：

```
top - 23:04:04 up  2:39,  3 users,  load average: 0.94, 0.22, 0.11
Tasks: 450 total,   8 running, 442 sleeping,   0 stopped,   0 zombie
%Cpu0  :  1.2 us, 22.4 sy,  0.0 ni, 70.1 id,  0.0 wa,  0.0 hi,  6.2 si,  0.0 st
%Cpu1  :  1.6 us, 11.6 sy,  0.0 ni, 81.7 id,  0.0 wa,  0.0 hi,  5.2 si,  0.0 st
%Cpu2  :  1.6 us, 15.7 sy,  0.0 ni, 79.5 id,  0.0 wa,  0.0 hi,  3.2 si,  0.0 st
%Cpu3  :  1.3 us, 11.3 sy,  0.0 ni, 66.2 id,  0.0 wa,  0.0 hi, 21.2 si,  0.0 st
KiB Mem :  3866948 total,  2659984 free,   606244 used,   600720 buff/cache
KiB Swap:  4063228 total,  4063228 free,        0 used.  2942672 avail Mem
```

如果多个 CPU 内核的利用率都相差不多，证明 nginx 已经成功的利用了多核 CPU。

测试结束后，CPU 内核的负载应该都同时降低。

worker_connections 4096
写在 events 部分。每一个 worker 进程能并发处理（发起）的最大连接数（包含与客户端或后端被代理服务器间等所有连接数）。

worker_rlimit_nofile 10240
写在 main 部分。worker 进程的最大打开文件数限制。默认是没有设置，如果没设置的话，这个值为操作系统的限制(ulimit -n)。可以限制为操作系统最大的限制 65535。把这个值设高，这样 nginx 就不会有“too many open files”问题了。

use epoll
写在 events 部分。在 Linux 操作系统下，nginx 默认使用 epoll 事件模型，得益于此，nginx 在 Linux 操作系统下效率相当高。同时 Nginx 在 OpenBSD 或 FreeBSD 操作系统上采用类似于 epoll 的高效事件模型 kqueue。

http 服务器:

与提供 http 服务相关的一些配置参数。例如：是否使用 keepalive 啊，是否使用 gzip 进行压缩等。

sendfile on
开启高效文件传输模式。

keepalive_timeout 65 : 长连接超时时间，单位是秒，长连接请求大量小文件的时候，可以减少重建连接的开销，如果设置时间过长，用户又多，长时间保持连接会占用大量资源。

client_max_body_size 10m
允许客户端请求的最大单文件字节数。如果有上传较大文件，请设置它的限制值

client_body_buffer_size 128k
缓冲区代理缓冲用户端请求的最大字节数

server_tokens off;

隐藏 nginx 的版本号

模块 http_proxy：
这个模块实现的是 nginx 作为反向代理服务器的功能，包括缓存功能

proxy_connect_timeout 60
nginx 跟后端服务器连接超时时间(代理连接超时)

proxy_read_timeout 60
定义从后端服务器读取响应的超时。此超时是指相邻两次读操作之间的最长时间间隔，而不是整个响应传输完成的最长时间。如果后端服务器在超时时间段内没有传输任何数据，连接将被关闭。

定义向后端服务器传输请求的超时。此超时是指相邻两次写操作之间的最长时间间隔，而不是整个请求传输完成的最长时间。如果后端服务器在超时时间段内没有接收到任何数据，连接将被关闭。

proxy_buffer_size 4k
设置缓冲区的大小为 size。nginx 从被代理的服务器读取响应时，使用该缓冲区保存响应的开始部分。这部分通常

包含着一个小小的响应头。该缓冲区大小默认等于 proxy_buffers 指令设置的一块缓冲区的大小，但它也可以被设置得更小。

proxy_buffers 8 4k

语法: proxy_buffers the_number is_size;

为每个连接设置缓冲区的数量为 number，每块缓冲区的大小为 size。这些缓冲区用于保存从被代理的服务器读取的响应。每块缓冲区默认等于一个内存页的大小。这个值是 4K 还是 8K，取决于平台。

附：查看 Linux 内存页大小

[root@www ~]# getconf PAGESIZE

4096

或

[root@www ~]# getconf PAGE_SIZE

4096

proxy_busy_buffers_size 64k
高负荷下缓冲大小（默认大小是 proxy_buffers 指令设置单块缓冲大小的 2 倍）

proxy_max_temp_file_size
当 proxy_buffers 放不下后端服务器的响应内容时，会将一部分保存到硬盘的临时文件中，这个值用来设置最大临时文件大小，默认 1024M。

proxy_temp_file_write_size 64k
当缓存被代理的服务器响应到临时文件时，这个选项限制每次写临时文件的大小。

模块 http_gzip：

gzip on : 开启 gzip 压缩输出，减少网络传输。

gzip_min_length 1k： 设置允许压缩的页面最小字节数，页面字节数从 header 头得 content-length 中进行获取。建议设置成大于 1k 的字节数，小于 1k 可能会越压越大。

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，按照原始数据大小以 16k 为单位的 4 倍申请内存。如果没有设置，默认值是申请跟原始数据相同大小的内存空间去存储 gzip 压缩结果

gzip_http_version 1.1： 用于识别 http 协议的版本，早期的浏览器不支持 Gzip 压缩，用户就会看到乱码，所以为了支持前期版本加上了这个选项，如果你用了 Nginx 的反向代理并期望也启用 Gzip 压缩的话，由于末端通信是 http/1.1，故请设置为 1.1。

gzip_comp_level 6： gzip 压缩比，1 压缩比最小处理速度最快，9 压缩比最大但处理速度最慢(传输快但比较消耗 cpu)

gzip_types：匹配 mime 类型进行压缩，无论是否指定"text/html"类型总是会被压缩的。

默认值: gzip_types text/html (默认不对 js/css 文件进行压缩)
压缩类型，匹配 MIME 类型进行压缩
不能用通配符 text/*
(无论是否指定)text/html 默认已经压缩
设置哪压缩种文本文件可参考 conf/mime.types

gzip_proxied any： Nginx 作为反向代理的时候启用，根据某些请求和应答来决定是否在对代理请求的应答启用 gzip 压缩，是否压缩取决于请求头中的“Via”字段，指令中可以同时指定多个不同的参数，意义如下：

off – 关闭所有的代理结果数据的压缩
expired – 启用压缩，如果 header 头中包含 “Expires” 头信息
no-cache – 启用压缩，如果 header 头中包含 “Cache-Control:no-cache” 头信息
no-store – 启用压缩，如果 header 头中包含 “Cache-Control:no-store” 头信息
private – 启用压缩，如果 header 头中包含 “Cache-Control:private” 头信息
no_last_modified – 启用压缩,如果 header 头中不包含 “Last-Modified” 头信息
no_etag – 启用压缩 ,如果 header 头中不包含 “ETag” 头信息
auth – 启用压缩 , 如果 header 头中包含 “Authorization” 头信息
any – 无条件启用压缩

gzip_vary on：和 http 头有关系，加个 vary 头，给代理服务器用的，有的浏览器支持压缩，有的不支持，所以避免浪费不支持的也压缩，所以根据客户端的 HTTP 头来判断，是否需要压缩

模块 http_stream：
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 后接负载均衡器的名字，后端 realserver 以 host:port options; 方式组织在 {} 中。如果后端被代理的只有一台，也可以直接写在 proxy_pass 。

Location：

root /var/www/html

定义服务器的默认网站根目录位置。如果 locationURL 匹配的是子目录或文件，root 没什么作用，一般放在 server 指令里面或/下。

index index.jsp index.html index.htm

定义路径下默认访问的文件名，一般跟着 root 放

proxy_pass http:/backend

请求转向 backend 定义的服务器列表，即反向代理，对应 upstream 负载均衡器。也可以 proxy_pass http://ip:port。

proxy_redirect off;

指定是否修改被代理服务器返回的响应头中的 location 头域跟 refresh 头域数值

例如：

设置后端服务器“Location”响应头和“Refresh”响应头的替换文本。 假设后端服务器返回的响应头是 “Location: http://localhost:8000/two/some/uri/”，那么指令

proxy_redirect http://localhost:8000/two/ http://frontend/one/;

将把字符串改写为 “Location: http://frontend/one/some/uri/”。

proxy_set_header Host $host;

允许重新定义或者添加发往后端服务器的请求头。

Host 的含义是表明请求的主机名，nginx 反向代理服务器会向后端真实服务器发送请求，并且请求头中的 host 字段重写为 proxy_pass 指令设置的服务器。因为 nginx 作为反向代理使用，而如果后端真实的服务器设置有类似防盗

链或者根据 http 请求头中的 host 字段来进行路由或判断功能的话，如果反向代理层的 nginx 不重写请求头中的 host 字段，将会导致请求失败。

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

后端的 Web 服务器可以通过 X-Forwarded-For 获取用户真实 IP

X_Forward_For 字段表示该条 http 请求是有谁发起的？如果反向代理服务器不重写该请求头的话，那么后端真实服务器在处理时会认为所有的请求都来自反向代理服务器，如果后端有防攻击策略的话，那么机器就被封掉了。因此，在配置用作反向代理的 nginx 中一般会增加两条配置，修改 http 的请求头：

```
proxy_set_header Host $host;
proxy_set_header X-Forward-For $remote_addr;
```

proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;

增加故障转移，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_set_header X-Real-IP $remote_addr;

web 服务器端获得用户的真实 ip 但是，实际上要获得用户的真实 ip，也可以通过 X-Forward-For

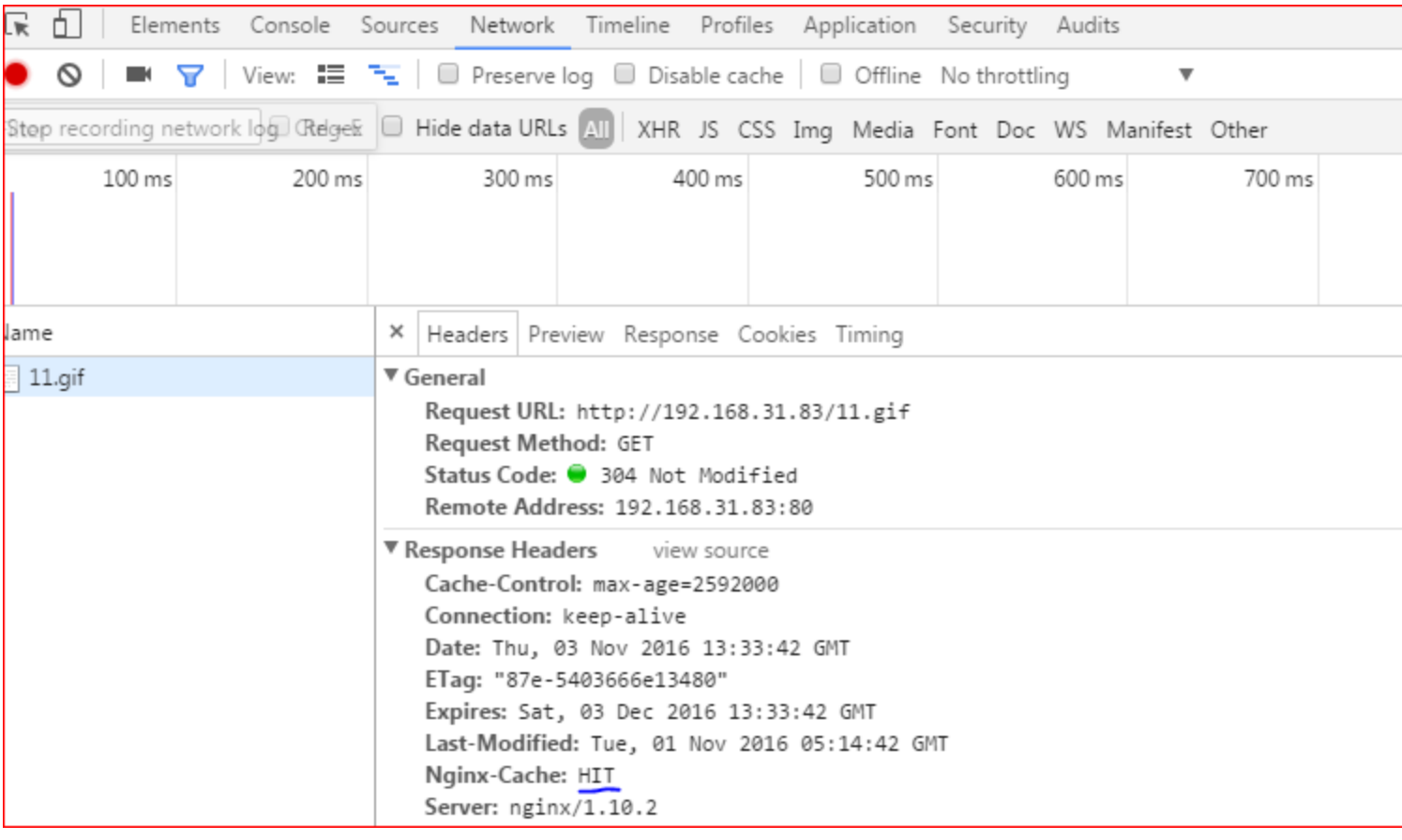
7、验证：nginx 反向代理的缓存功能、负载均衡及健康检查

1）下面我们来测试一下缓存功能

如果在缓存时间之内需要更新被缓存的静态文件怎么办呢，这时候就需要手动来清除缓存了。

ngx_cache_pure 清除缓存模块使用说明

用谷歌浏览器测试的时候，可以按 F12 调用开发工具，选择 Network 选项，我们可以看到，Response Headers，在这里我们可以看到，我们请求的是否是缓存



从图中我们可以看到，我们访问的服务器是 192.168.31.83，缓存命中。

也可以查看缓存目录或 nginx 的访问日志

清除缓存:

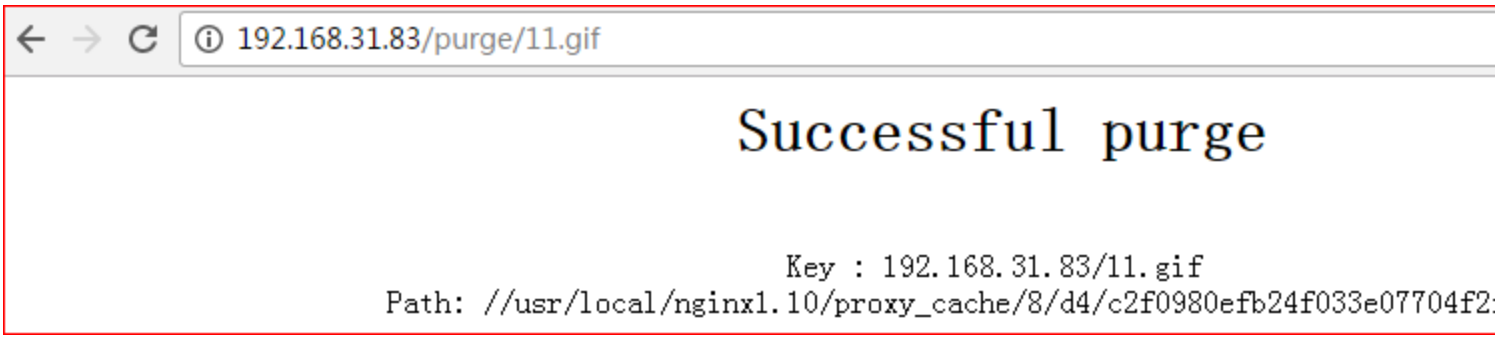
上述配置的 proxy_cache_purge 指令用于方便的清除缓存，但必须按照第三方的 ngx_cache_purge 模块才能使用

使用 ngx_cache_purge 模块清除缓存（直接删除缓存目录下的文件也算一种办法）：

GET 方式请求 URL

即使用配置文件中的 location ~ /purge(/.*)

浏览器访问 http://192.168.31.83/purge/your/may/path 来清除缓存



缓存清除成功。

备注：

（1）purge 是 ngx_cache_pure 模块指令
（2）your/may/path　是要清除的缓存文件 URL 路径

2）若只有一台客户端要验证负载均衡和健康检查可以先关掉缓存功能和保持 session 会话

#proxy_buffering off;

#sticky

测试过程略

扩展知识 1：
nginx 修改版本等信息
1、vi /usr/local/src/nginx-1.0.12/src/core/nginx.h　#编译前编辑
#define nginx_version
#define NGINX_VERSION
#define NGINX_VER
#define NGINX_VAR
修改上面的信息，即可更改 nginx 显示版本。
2、vi /usr/local/src/nginx-1.0.12/src/http/ngx_http_special_response.c　#编译前编辑
static u_char ngx_http_error_full_tail[] =
static u_char ngx_http_error_tail[] =
修改上面的信息为你自己的。
3、vi /usr/local/src/nginx-1.0.12/src/http/ngx_http_header_filter_module.c　#编译前编辑
static char ngx_http_server_string[]=
修改上面的信息为你自己的。
4、编译完成之后，修改/usr/local/nginx/conf 目录下面
fastcgi.conf、fastcgi.conf.default、fastcgi_params、fastcgi_params.default
这四个文件里面的版本名称
/usr/local/nginx/sbin/nginx -V　#查看 nginx 版本号

欢迎点击这里的链接进入精彩的 Linux 公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

Linux 公社微信公众号：linuxidc_com