三台实验机器:
web1:192.168.56.101　　Proxy服务器
web2:192.168.56.102　　web服务器
web3:192.168.56.103　　web服务器
web4:192.168.56.104　　web服务器
我主机的ip：192.168.56.1

环境设置：
关掉iptable,selinux
安装lnmp，web主目录均在/home/www
web主目录均有一个index.html 页面，存入了各自机器的ip地址
设置完成后，保证三台服务器均可以正常访问

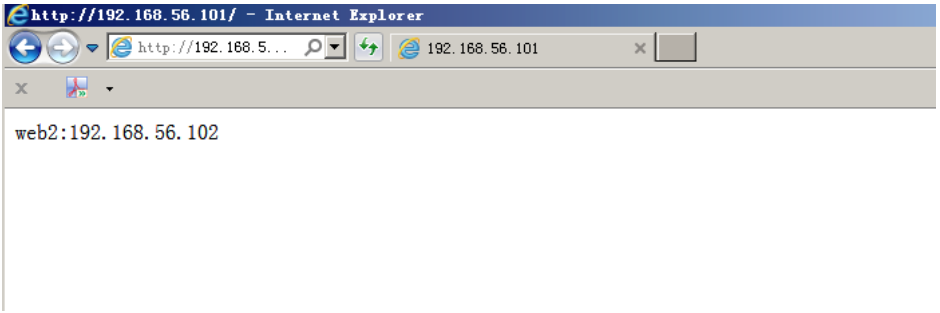
nginx代理模块主要是proxy_pass
官方中文文档:
http://www.howtocn.org/nginx:nginx%E6%A8%A1%E5%9D%97%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C%E4%B8%AD%E6%96%87%E7%89%88:s
指令说明：proxy_pass
语法：proxy_pass URL
默认值：no
使用字段：location, location中的if字段
这个指令设置被代理服务器的地址和被映射的URI，地址可以使用主机名或IP加端口号的形式，例如：proxy_pass http://localhost:8000/uri/;

1.单个ip的反向代理
1.1设置web1的nginx配置：

```
server
    {
        listen      80;
        server_name www.web1.com;
        index index.html index.htm index.php;
        root  /home/www/;

        location / {
            proxy_pass http://192.168.56.102;     #被代理的服务器的端口
        }
```

现在访问web1



说明代理成功了，
1.2日志中获取客户真实ip

现在还有一个问题，是代理服务器怎么获取到客户真实ip的问题，
这里主要使用X-Forwarded-For 字段
解释（维基百科）：
X-Forwarded-For(XFF)是用来识别通过HTTP代理或负载均衡方式连接到Web服务器的客户端最原始的IP地址的HTTP请求头字段。
地址：http://zh.wikipedia.org/wiki/X-Forwarded-For
1.2.1 Apeche(WEB)
1.2.1.1 Nginx配置

```
location / {
  proxy_pass http://192.168.56.102;
   proxy_set_header Host $host;
  proxy_set_header X-Forwarded-For $remote_addr;
}
```

1.2.1.2 Apache日志格式配置（这里和HAProxy一样）

```
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

提示：

(1) 注意大括号后面还有一个“i”；
(2)（自己测试）大括号内“X-Forwarded-For”的值可以随意定义只要和Nginx中X-Forwarded-For 一致，不区分大小写
1.2.2 Nginx(WEB)
1.2.2.1 Nginx(Proxy)
示例：

```
location  / {
  proxy_set_header Host $host;
  proxy_set_header X-Forwarded-For $remote_addr;
  proxy_pass http://192.168.56.102;
}
```

1.2.2.2 Nginx（WEB）日志格式配置

```
log_format  main  '$http_x_forwarded_for - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" ';
```

1.3 程序中获取客户端的ip
写了一个php程序放在web2上，代码如下:

```
<?php
echo  'HTTP_X_FORWARDED_FOR:'.$_SERVER["HTTP_X_FORWARDED_FOR"];
echo '<br>';
echo  'REMOTE_ADDR:'.$_SERVER["REMOTE_ADDR"];
?>
```

执行结果:

```
HTTP_X_FORWARDED_FOR:192.168.56.1
REMOTE_ADDR:192.168.56.101
```

可以看出使用
$_SERVER["HTTP_X_FORWARDED_FOR"]获取到的是客户端真实的ip
$_SERVER["REMOTE_ADDR"] 获取到的是代理服务器的ip

还有一种方法是通过nginx的Real IP模块
官方中文文档:
http://www.howtocn.org/nginx:nginx%E6%A8%A1%E5%9D%97%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C%E4%B8%AD%E6%96%87%E7%89%88:c
这个模块默认是没有开启的，需要编译的时候加上--with-http_realip_module 来开启这个模块，
nginx的配置
Nginx(Proxy)
示例：

```
        location / {
                proxy_pass http://192.168.56.102;
                proxy_set_header  X-Real-IP  $remote_addr;
                proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;
                proxy_set_header  Host            $host;
            }
```

Nginx（WEB)
nginx的配置:

```
server
      {
            listen       80;
            server_name www.lnmp.org;
            index index.html index.htm index.php;
            root  /home/www/;
            set_real_ip_from   192.168.56.0/24;    #指定接收来自哪个前端发送的 IP head 可以是单个IP或者IP段
            set_real_ip_from    192.168.56.101
            real_ip_header     X-Real-IP;
```

nginx的日志格式

```
            log_format  access  '$remote_addr - $remote_user [$time_local] "$request" '
          '$status $body_bytes_sent "$http_referer" '
          '"$http_user_agent" $http_x_forwarded_for';
```

现在日志里就可以获取到真实客户端的ip了，
程序中获取客户端的ip
写了一个php程序放在web2上，代码如下:

```
<?php
echo  'HTTP_X_FORWARDED_FOR:'.$_SERVER["HTTP_X_FORWARDED_FOR"];
echo '<br>';
```

```
echo  'REMOTE_ADDR:'.$_SERVER["REMOTE_ADDR"];
?>
```

执行结果:
HTTP_X_FORWARDED_FOR:192.168.56.1
REMOTE_ADDR:192.168.56.1
可以看出使用
$_SERVER["HTTP_X_FORWARDED_FOR"]获取到的是客户端真实的ip
$_SERVER["REMOTE_ADDR"] 获取到的是客户端真实的ip

注:如果后端是apache，要使用apache的mod_rpaf模块，

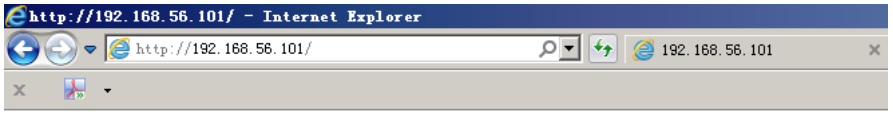
2.多ip的负载均衡
nginx的负载均衡，主要使用的模块是Upstream
官方中文文档:
http://www.howtocn.org/nginx:nginx%E6%A8%A1%E5%9D%97%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C%E4%B8%AD%E6%96%87%E7%89%88:s
web1的nginx配置:

```
http
    {
        include       mime.types;
        default_type  application/octet-stream;

upstream test.com_server{       #通过upstream指令指定了一个负载均衡器的名称test.com_server。这个名称可以任意指定，在后面需要用到的地方直接调用即可
    server 192.168.56.102:80 weight=1 ;
    server 192.168.56.103:80 weight=1 ;  #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
}
server
    {
        listen      80;
        server_name www.web1.com;
        index index.html index.htm index.php;
        root  /home/www/;
        location / {
            proxy_pass http://test.com_server;
            proxy_set_header  X-Real-IP  $remote_addr;
            proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;
            proxy_set_header  Host            $host;
        }
    }
```

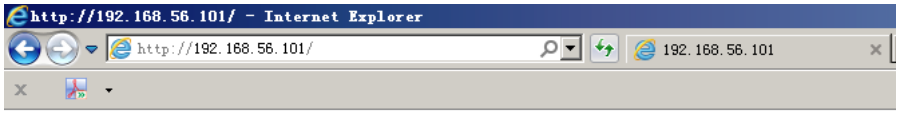
注，upstream是定义在server{ }之外的，不能定义在server{ }内部。定义好upstream之后，用proxy_pass引用一下即可。

现在访问192.168.56.101的时候，第一次访问会是



web2:192.168.56.102

再刷新一次，就变成



web3:192.168.56.103

下面写一下nginx的负载均衡算法
1、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，如果后端服务器down掉，能自动剔除。

2、weight
指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

例如：

```
upstream bakend {
server 192.168.1.10 weight=10;
server 192.168.1.11 weight=10;
}
```

3、ip_hash
每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。
例如：

```
upstream resinserver{
ip_hash;
server 192.168.1.10:8080;
server 192.168.1.11:8080;
server 192.168.1.12:8080 down;
}
```

注：无法将权重（weight）与ip_hash联合使用来分发连接(他们是不同且彼此冲突的策略)。backup 也不能和 ip_hash 关键字一起使用
4、fair（第三方）
按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。

```
upstream resinserver{
server server1;
server server2;
fair;
}
```

5、url_hash（第三方）
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，后端服务器为缓存时比较有效。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。
例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的hash算法

```
upstream resinserver{
server squid1:3128;
server squid2:3128;
hash $request_uri;
hash_method crc32;
}
```

负载均衡调度中的状态。常用的状态有：
down，表示当前的server暂时不参与负载均衡。这标志着服务器永远不可用，这个参数只配合ip_hash使用
backup，预留的备份机器。当其他所有的非backup机器出现故障或者忙的时候，才会请求backup机器，因此这台机器的压力最轻。
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，在max_fails定义的失败次数后，距离下次检查的间隔时间，默认是10s；max_fails可以和fail_timeout一起使用。

比如：

```
upstream webservers {
      server 192.168.56.102 weight=1 max_fails=2 fail_timeout=2;
      server 192.168.56.103 weight=1 max_fails=2 fail_timeout=2;
   }
```

有一种情况需要注意，就是 upstream 中只有一个 server 时，max_fails 和 fail_timeout 参数可能不会起作用。导致的问题就是 nginx 只会尝试一次 upstream 请求，如果失败这个请求就被抛弃了：(……解决的方法，比较取巧，就是在 upstream 中将你这个可怜的唯一 server 多写几次，如下：

```
upstream webservers {
   server 192.168.56.102 max_fails=2 fail_timeout=30s;
   server 192.168.56.102 max_fails=2 fail_timeout=30s;
   server 192.168.56.102 max_fails=2 fail_timeout=30s;
}
```

现在付一个可以在线上使用的nginx负载均衡配置:

```
user  www www;
worker_processes 8;
error_log  logs/error.log  crit;
```

```
pid        logs/nginx.pid;

worker_rlimit_nofile 65535;
events
{
  use epoll;
  worker_connections 65535;
}
http
{
  include       mime.types;
  default_type  application/octet-stream;
  #charset  gbk;

  server_names_hash_bucket_size 128;
  client_header_buffer_size 32k;
  large_client_header_buffers 4 32k;

  proxy_redirct off;                    #其作用是对发送给客户端的URL进行修改,
  client_max_body_size 10m;      #允许客户端请求的最大的单个文件字节数
  client_body_buffer_size 128k;  #缓冲区代理缓冲用户端请求的最大字节数 可以理解为先保存到本地再传给用户
  proxy_connect_timeout 600;       #跟后端服务器连接的超时时间_发起握手等候响应超时时间
  proxy_read_timeout 600;          #连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理
  proxy_send_timeout 600;        #后端服务器数据回传时间_就是在规定时间之内后端服务器必须传完所有的数据
  proxy_buffer_size 16k;          #代理请求缓存区_这个缓存区间会保存用户的头信息以供Nginx进行规则处理_一般只要能保存下头信息即可
  proxy_buffers 4 32k;              #同上 告诉Nginx保存单个用的几个Buffer 最大用多大空间
  proxy_busy_buffers_size 64k;   #如果系统很忙的时候可以申请更大的proxy_buffers 官方推荐*2
  proxy_temp_file_write_size 64k;    #proxy 缓存临时文件的大小
  sendfile on;
  tcp_nopush     on;
  keepalive_timeout 60;
  tcp_nodelay on;

  upstream web.abc.com {
     ip_hash;
     server 192.168.1.10:80 max_fails=2 fail_timeout=30s;
     server 192.168.1.11:80 max_fails=2 fail_timeout=30s;
  }

server
   {
      listen 80;
      server_name 192.168.1.102;
      index index.htm index.html;
      root  /home/www;

   location / {
      proxy_pass http://web.abc.com;
      proxy_next_upstream http_500 http_502 http_503 error timeout invalid_header;  #设置从其中一个upstream   返回时如果错误，超时，或500等则转到下一个upstream
    }
   }
```

参考资料:

Nginx 反向代理、负载均衡、页面缓存、URL重写及读写分离详解

http://freeloda.blog.51cto.com/2033581/1288553

轻松实现Nginx HTTP 反向代理+负载均衡

http://yapeng.blog.51cto.com/4455269/1172502

使用Nginx轻松实现开源负载均衡

http://www.slideshare.net/Cary/nginx-presentation