目录

[1.1 实验环境准备 2](#_Toc444175672)

[1.1.1 软硬件准备 2](#_Toc444175673)

[1.1.2 安装nginx软件 2](#_Toc444175674)

[1.1.3 配置调试用于测试的web服务 2](#_Toc444175675)

[1.2 nginx负载均衡详细介绍 3](#_Toc444175676)

[1.2.1 实现一个简单的负载均衡 3](#_Toc444175677)

[1.3 upstream模块介绍 5](#_Toc444175678)

[1.3.1 upstream 模块语法 5](#_Toc444175679)

[1.3.2 upstream模块相关说明 6](#_Toc444175680)

[1.2 upstream模块调度算法 8](#_Toc444175681)

[1.3 proxy\_pass 指令 12](#_Toc444175682)

[1.3.1 http proxy模块参数 12](#_Toc444175683)

[1.3.2 根据URL的目录地址实现代理转发 14](#_Toc444175684)

[1.4 nginx负载均衡高可用 17](#_Toc444175685)

[1.4.1 配置keepalive实现nginx负载均衡的高可用 17](#_Toc444175686)

[1.5 nginx负载均衡检测节点状态 18](#_Toc444175687)

[1.6 nginx cache 18](#_Toc444175688)

[2. keepalive高可用集群 18](#_Toc444175689)

[2.1 keepalive服务介绍 18](#_Toc444175690)

[2.2 keepalive服务两大用途：healthcheck&failover 19](#_Toc444175691)

[2.2.1 LVSdirector是 failover功能 19](#_Toc444175692)

[2.2.2 LVS directors failover功能原理图 19](#_Toc444175693)

[2.2.3 LVS cluster nodes healthchecks功能 23](#_Toc444175694)

[2.3 keepalived故障切换转义原理介绍 24](#_Toc444175695)

[2.3.1 VRRP协议简单介绍 24](#_Toc444175696)

[2.4 配置keepalived实现服务高可用功能 25](#_Toc444175697)

[2.4.1 下载keepalived软件包 25](#_Toc444175698)

[2.4.2 安装keepalived 25](#_Toc444175699)

[2.4.3 keepalived检测裂脑脚本 29](#_Toc444175700)

[2.4.4 修改keepalived配置日志的操作： 29](#_Toc444175701)

[2.4.5 keepalived多实例： 30](#_Toc444175702)

## 1.1 实验环境准备

### 1.1.1 软硬件准备

（1） 测试硬件准备

4台虚拟机，两台做负载均衡两台做RS。

|  |  |  |
| --- | --- | --- |
| **HOSTNAME** | **IP** | **说明** |
| lb01 | 10.1.0.5 | nginx主负载均衡器 |
| lb02 | 10.1.0.6 | nginx辅负载均衡器 |
| web01 | 10.1.0.7 | apache web01 服务器 |
| web02 | 10.1.0.8 | apache web02 服务器 |

（2） 软件准备

系统：Centos6.6 x86\_64

软件：nginx-1.6.3.tar.gz (http://nginx.org/download/nginx-1.6.3.tar.gz)

### 1.1.2 安装nginx软件

mkdir -p /home/application/tools

cd /home/oldboy/tools

wget http://nginx.org/download/nginx-1.6.3.tar.gz

tar xf nginx-1.6.3.tar.gz

cd nginx-1.6.3

useradd -s /sbin/nologin nginx -M

make && make install

yum install pcre pcre-devlel -y

yum install openssl-devel –y

./configure --prefix=/application/nginx-1.6.3 --user=nginx --group=nginx --with-http\_ssl\_module --with-http\_stub\_status\_module

ln -s /application/nginx-1.6.3/ /application/nginx

/application/nginx/sbin/nginx

### 1.1.3 配置调试用于测试的web服务

注意：本小姐操作只在以下nginx web服务器节点操作：

|  |  |  |
| --- | --- | --- |
| RS01 | 10.1.0.7 | nginx web01 服务器 |
| RS02 | 10.1.0.8 | nginx web02 服务器 |

配置并查看web服务器配置结果。

[root@web01 ~]# echo 10.0.0.7 >/var/www/html/index.html

[root@web01 ~]# curl 10.1.0.7

10.0.0.7

[root@web02 ~]# echo 10.1.0.8 >/var/www/html/index.html

[root@web02 ~]# curl 10.1.0.8

10.1.0.8

提示：请记录下不同web的返回结果

## 1.2 nginx负载均衡详细介绍

### 1.2.1 实现一个简单的负载均衡

注意：本次操作只在以下nginx lb01服务器节点操作：

|  |  |  |
| --- | --- | --- |
| **HOSTNAME** | **IP** | **说明** |
| lb01 | 10.1.0.5 | nginx主负载均衡器 |

一个简单的nginx的负载均衡的配置：

worker\_processes 1;

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

upstream web\_pools {

server 10.1.0.7:80 weight=5;

server 10.1.0.8:80 weight=5;

server 10.1.0.6:80 backup;

}

server {

listen 80;

server\_name www.etiantian.org;

location / {

root html;

index index.html index.htm;

proxy\_pass http://web\_pools;

}

}

}

**实现nginx负载均衡功能的模块：**

upstream web\_pools {

server 10.1.0.7:80 weight=5;

server 10.1.0.8:80 weight=5;

server 10.1.0.6:80 backup;

}

server {

location / {

proxy\_pass http://**dynamic**;

health\_check;

}

}

1） Linux作为客户端测试结果

[root@lb01 conf]# curl 10.1.0.5

10.1.0.7

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

[root@lb01 conf]# curl 10.1.0.5

10.1.0.7

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

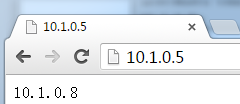
[root@lb01 conf]# curl 10.1.0.5

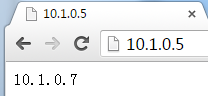
10.1.0.7

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

2） Windows作为客户端测试结果





3） 宕机掉任意RS测试结果

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

[root@lb01 conf]# curl 10.1.0.5

10.1.0.8

## 1.3 upstream模块介绍

nginx的负载均衡功能依赖于ngx\_http\_upstream\_module模块，所支持的代理方式有proxy\_pass,fastcgi\_pass,memcached\_pass。

官方地址：<http://nginx.org/en/docs/http/ngx_http_upstream_module.html>

### 1.3.1 upstream 模块语法

（1） 范例1：

upstream web\_pools {

server 10.1.0.7:80 weight=5;

server 10.1.0.8:80 weight=5;

}

（2） 范例2：

upstream web\_pools {

server 10.1.0.7:80 weight=5;

server 10.1.0.8:80 weight=5 **max\_fails=1 fail\_timeout=10s;**

}

（3）示例及解释：

upstream web\_pools {

server 10.1.0.7:80 weight=5;

## server是固定的，后面可以接域名或IP，如果不加端口，默认是80端口。weight代表权重，值越大被分配的几率越高；

server backend2.example.com:8080 weight=5;

#域名加端口。转发到后端的指定端口上。

server unix:/tmp/backend3;

##指定socket文件

提示：server如果街域名，需要内网有DNS服务器，或者在负载均衡的hosts文件做域名解析。server后面还可以直接接IP或IP加端口

server 10.1.0.6:80 backup;

##备份服务器，等上面指定的服务器都不可访问的时候会启用，backup的用法和haproxy中用法一样。

}

两台keepalive+http做高可用，那么这里使用nginx的backup参数就可以实现了。

### 1.3.2 upstream模块相关说明

1． upstream模块应放于nginx.conf配置的http{}标签内。

2． upstream模块默认算法是wrr（权重轮询weighted round-robin）。

3． upstream模块内部部分参数说明。

|  |  |
| --- | --- |
| server 10.0.0.10:80 | 负载均衡后面的RS配置，可以是IP或域名，端口不写，默认是80端口，高并发场景IP要换成域名，通过DNS做负载均衡。 |
| weight | 权重，默认是1，权重大接受的请求越多。 |
| max\_fails=2 | 最大尝试失败的次数，默认为1,0表示禁止失败尝试。企业场景：建议2-3次。京东1次，蓝汛10次，根据业务需求去配置。 |
| backup | 热备配置（RS节点的高可用），当前面激活的RS都失败后会自动启用热备RS。 |
| fail\_timeout=20s | 失败超时时间，默认是10s。京东1次，蓝汛10次，根据业务需求去配置。京东3秒，蓝汛3秒，根据业务需求去配置。常规业务2-3秒合理。 |
| down | 这标志着服务器永远不可用，这个参数一直配合ip\_hash使用。 |

提示：以上的参数和专业的haproxy参数几乎一样。

4. upstream参数官方说明：

1） weight=number

sets a weight of the server,by default 1.

设置该服务器的权重，默认值是1，这个数值越大，服务器会被转发更多的请求；

注意：当负载调度算法为ip\_hash时，后端服务器在负载均衡调度中的状态不能是weight和backup。

2） max\_fails=number

sets a number of unsuccessful attempts to communicate with the server during a time set by the fail\_timeout parameter after which it will be considered down for a period of time also set by the fail\_timeout parameter. By default, the number of unsuccessful attempts is set to 1. A value of zero disables accounting of attempts. What is considered to be an unsuccessful attempt is configured by the proxy\_next\_upstream,fastcgi\_next\_upstream, and memcached\_next\_upstream directives. The http\_404 stats is not considered an unsuccessful attempt.

nginx尝试连接后端主机失败的次数，这个数值是配合proxy\_next\_upstream，fastcgi\_next\_upstream,and memcached\_next\_upstream 这三个参数来使用的，当nginx接收后端服务器返回这三个参数定义的状态码的时候，会将这个请求转发给正常工作的后端服务器，例如404,502,503。max\_fails默认值是1；

3） fail\_timeout=time

sets a time during which the specified number of unsuccessful attempts to communicate with the server should happen for the server to be considered down;

and a period of time the server will be considered down.

By default,timeout is set to 10 seconds.

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；

如果max\_fails是5，他就检测5次，如果5次都是502，那么，就会根据fail\_timeout的值，等待10s再去检查

4） backup

marks the server as a backup server.It will be passed requests when the primary servers are down.

这标志着这个服务器作为备份服务器，当主服务器全部宕机的时候，才会向他转发请求；

注意：当负载调度算法为ip\_hash时，后端服务器在负载均衡调度中的状态不能是weight和backup。

5） down

marks the server as permanently down; used along with the ip\_hash directive.

这标志着服务器永远不可用，这个参数一致配合ip\_hash使用

示例：

upstream backend{

server backed1.example.com weight=5;

##如果就是单个server，没必要设置权重。

server 127.0.0.1:8080 max\_fails=5 fail\_timeout=10s;

##当检测次数等于3的时候，间隔30s再检查，这个参数和proxy/fasrcgi/memcached\_next\_upstream，相关；

server unix:/tmp/backend3;

server backup1.examle.com:8080 backup; ##热备机器设置。

}

6） max\_fails=5 fail\_timeout=10s

重新加载nginx配置，如果后端出现proxy\_next\_upstream中定义的错误（502），nginx会根据max\_fails的值去后端服务器检测，如果5次都是502，那么，就会根据fail\_timeout的值，等待10s再去检查，过10s后检查一次，如果还是502，那么继续等待10s，再去检查，还是只检查一次，如果持续502，在不重新加载nginx配置的情况下，每隔10s都只检测一次。

7） max\_conns=number

最大并发连接数

8） slow\_start=time

服务器宕机之后，重启之后认为多久才能恢复过来。

5. 特别说明

对于nginx代理cache服务时，可能需要使用hash算法，此时，如果宕机时，可通过设置down参数确保客户端用户按照当前的hash算法访问，这点很重要。

upstream web\_pools {

ip\_hash;

server 10.1.0.7:80 weight=5;

server 10.1.0.8:80 weight=5;

}

Haproxy负载均衡配置例子：

#开启对后端服务器的健康检查，通过GET /test/index.php来判断后端服务器的健康情况

server php\_server\_1 10.12.25.68:80 cookie 1 check inter 2000 rise 3 fall 3 weight 2

server php\_server\_2 10.12.25.72:80 cookie 2 check inter 2000 rise 3 fall 3 weight 1

server php\_server\_bak 10.12.25.79:80 cookie 3 check inter 1500 rise 3 fall 3 backup

## 1.2 upstream模块调度算法

1） rr轮询（默认）

按客户端请求顺序把客户端的请求逐一分配到不同的后端服务器，这相当于LVS中rr算法，如果后端服务器宕机（默认情况下只检测80端口，如果后端包502,404,403,503，还是会直接返回给用户），宕机服务器会被自动剔除，使用户访问不受影响，请求会分配给正常的服务器。

2） weight（权重）

在轮询算法的基础上加上权重（默认是rr+weight），权重轮询和访问成正比，权重越大，转发的请求也就越多。可以根据服务器的配置和性能指定权重值大小，可以有效解决新旧服务器性能不均进行请求分配问题。

示例：

后端服务器192.168.1.2配置：E5520\*2 CPU,8G内存。

后端服务器192.168.1.3配置：Xeon(TM)2.80GHz\*2,4G内存

我希望在有30个请求到达前端时，其中20个请求交给192.168.1.3处理，剩余10个请求交给192.168.1.2处理，就可以做如下配置；

upstream lili\_lb{

server 192.168.1.2 weight=1;

server 192.168.1.3 weight=2;

}

3） ip\_hash

每个请求按访问的ip的hash结果分配，当新的请求到达时，先将客户端ip通过哈希算法哈希出一个值，在随后请求客户端，ip的哈希值只要相同，就会被分配至同一台服务器（LVS负载均衡的-p参数，keepalive的配置里的persistence\_timeout 50），**该调度算法可以解决动态网页session共享问题**，但有时会导致请求分配不均，即无法保证1:1的负载均衡。在国内所有的公司都是NAT上网，多个PC对应一个外部IP。

提示：必须是最前端的服务器，后端也必须直接接应用服务器多数情况不能和权重参数一起使用。http://nginx.org/en/docs/http/ngx\_http\_upstream\_module.html

示例：

upstream lili\_lb{

ip\_hash;

server 192.168.1.2:80;

server 192.168.1.3:8080;

}

upstream backend{

ip\_hash

server backend1.example.com;

server backend2.example.com;

server backend3.example.com down;

server backend4.example.com;

}

注意：当负载调度算法为ip\_hash时，后端服务器在负载均衡调度中的状态不能是weight和backup

4） fair（第三方，NO）动态算法

按照后端服务器RS的响应时间分配请求，响应时间短的优先分配。

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的进行负载均衡，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。

示例：

upstream lili\_lb{

server 192.168.1.2;

server 192.168.1.3;

fair;

}

5） url\_hash（第三方，NO）

按访问URL的hash结果来分配请求，让每个URL定向到同一个后端服务器，后端服务器为缓存服务器时效果显著。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。

url\_hash。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命中率。nginx本身是不支持URL\_hash的，如果需要使用着用调度算法，必须安装nginx的hash软件包。

示例：

upstream lili\_lb{

server squid1:3128;

server squid2:3128;

hash $request\_uri;

hash\_method orc32;

}

6） least\_conn

最少连接数。哪个机器连接数少就分发。

7） 一致性HASH（高级J2EE程序架构师）

http{

upstream lili\_lb{

consistent\_hash $request\_uri;

server 127.0.0.1:9001 id=1001 weight=3;

server 127.0.0.1:9002 id=1002 weight=10;

server 127.0.0.1:9003 id=1003 weight=20;

}

}

## 1.3 proxy\_pass 指令

proxy\_pass指令属于ngx\_http\_proxy\_module模块，此模块可以将请求转发到另一台服务器。

见：<http://nginx.org/en/docs/http/ngx_http_proxy_module.html>

location/name{

proxy\_pass http://127.0.0.1/remote/;

}

location/some/path/{

proxy\_pass http://127.0.0.1;

}

location /name/ {

rewrite /name/([^/]+) /users?name=$1 break;

proxy\_pass http://127.0.0.1;

}

### 1.3.1 http proxy模块参数

nginx的代理功能是通过http proxy模块来实现的。默认在安装nginx时已经安装了http proxy模块因此可直接使用http proxy模块。

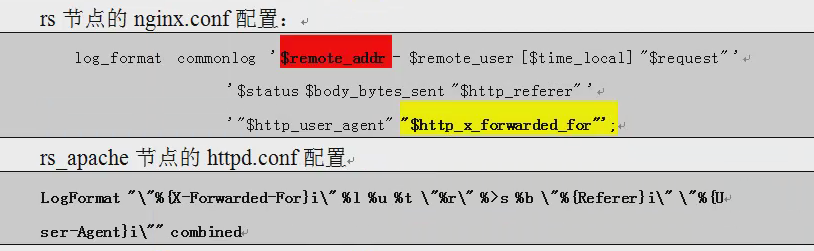
下面详细解释每个选项代表的含义。

|  |  |
| --- | --- |
| proxy\_set\_header | 设置由后端的服务器获取用户的主机名或者真是IP地址，以及代理者的真是IP地址。 |
| client\_body\_buffer\_size | 用于指定客户端请求主体缓冲区大小，可以理解为先保存到本地再传给用户。 |
| proxy\_connect\_timeout | 表示与后端服务器连接的超时时间，即发起握手等候响应的超时时间。 |
| proxy\_send\_timeout | 表示后端服务器的数据回传时间，即在规定时间之内后端服务器必须传完所有的数据，否则，nginx将断开这个连接。 |
| proxy\_read\_timeout | 设置nginx从代理的后端服务器获取信息的时间，表示连接建立成功后，nginx等待后端服务器的响应时间，其实是nginx已经进入后端的排队之中等候处理的时间。 |
| proxy\_buffer\_size | 设置缓冲区大小，默认，该缓冲区大小等于指令proxy\_buffers设置的大小。 |
| proxy\_bufers | 设置缓冲区的数量和大小。nginx从代理的后端服务器获取的响应信息，会放置到缓冲区。 |
| proxy\_busy\_buffers\_size | 用于设置系统很忙时可以使用的proxy\_buffers大小，官方推荐的大小为proxy\_buffers\*2 |
| proxy\_temp\_file\_write\_size | 指定proxy缓存临时文件的大小。 |

proxy\_pass参数说明：

|  |  |
| --- | --- |
| proxy\_pass <http://blog_pool>; | 用于指定反向代理的服务器池 |
| proxy\_set\_header Host $host; | 点后端web服务器上也配置有多个虚拟主机时，需要用该header来区分反向代理哪个主机名。 |
| proxy\_set\_header X-Forwarded-For $remote\_addr; | 如果后端web服务器上的程序需要获取用户IP，从该header头获取。 |

Apache记录真实IP配置：



nginx相关参数

[root@lb01 conf]# cat proxy.conf

proxy\_redirect off;

proxy\_set\_header Host $host;

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

proxy\_connect\_timeout 90;

proxy\_send\_timeout 90;

proxy\_read\_timeout 90;

proxy\_buffer\_size 4k;

proxy\_buffers 4 32k;

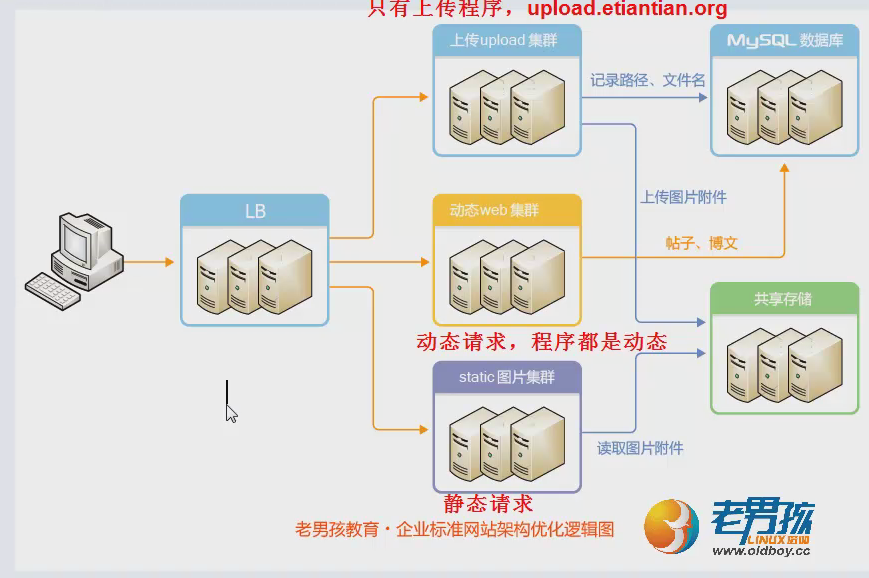
proxy\_busy\_buffers\_size 64k;

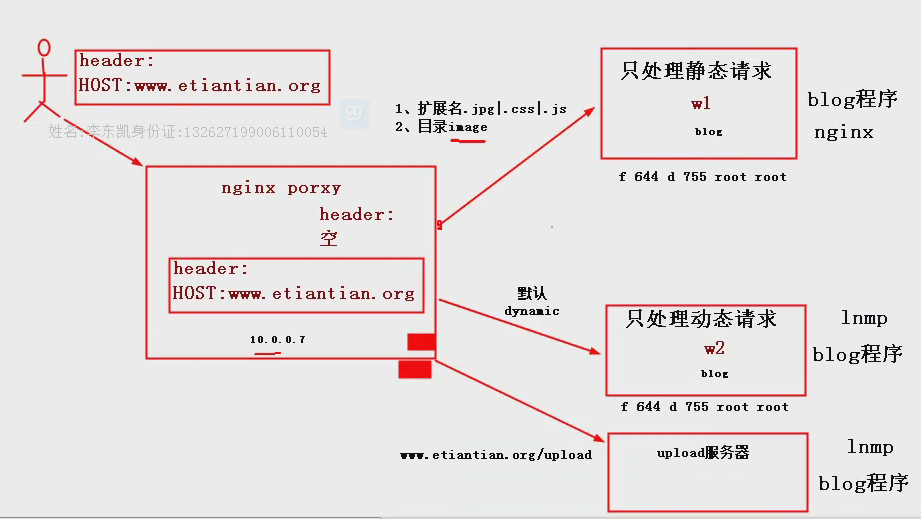
proxy\_temp\_file\_write\_size 64k;

### 1.3.2 根据URL的目录地址实现代理转发

企业案例：通过nginx实现动静分离（动态、静态服务分开访问）

#### 1.3.2.1 upstream模块的配置





动、静分离基于目录的配置文件

upstream static\_pools {

server 10.1.0.7:80 weight=5 max\_fails=1 fail\_timeout=10s;

}

upstream dynamic\_pools {

server 10.1.0.8:80 weight=5;

}

server {

listen 80;

server\_name www.etiantian.org;

location / {

root html;

index index.html index.htm;

proxy\_pass http://dynamic\_pools;

include proxy.conf;

}

location /image/ {

proxy\_pass http://static\_pools;

include proxy.conf;

}

}

}

动静分离基于扩展名（正则）配置文件：

upstream static\_pools {

server 10.1.0.7:80 weight=5 max\_fails=1 fail\_timeout=10s;

}

upstream dynamic\_pools {

server 10.1.0.8:80 weight=5;

}

server {

listen 80;

server\_name www.etiantian.org;

location / {

root html;

index index.html index.htm;

proxy\_pass http://dynamic\_pools;

include proxy.conf;

}

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

proxy\_pass http://static\_pools;

include proxy.conf;

}

}

}

在开发无法通过程序实现动静分离的时候，运维可以根据资源实体进行动静分离，而不依赖于开发。

根据user\_agent转发

location / {

if ($http\_user\_agent ~\* "MSIE")

{

proxy\_pass http://dynamic\_pools;

}

if ($http\_user\_agent ~\* "Firefox")

{

proxy\_pass http://static\_pools;

}

proxy\_pass http://dynamic\_pools;

include proxy.conf;

}

苹果、安卓端配置

upstream static\_pools {

server 10.1.0.7:80 weight=5 max\_fails=1 fail\_timeout=10s;

}

upstream dynamic\_pools {

server 10.1.0.8:80 weight=5;

}

server {

listen 80;

server\_name www.etiantian.org;

location / {

if ($http\_user\_agent ~\* "android")

{

proxy\_pass http://static\_pools;

}

if ($http\_user\_agent ~\* "iphone")

{

proxy\_pass http://dynamic\_pools;

}

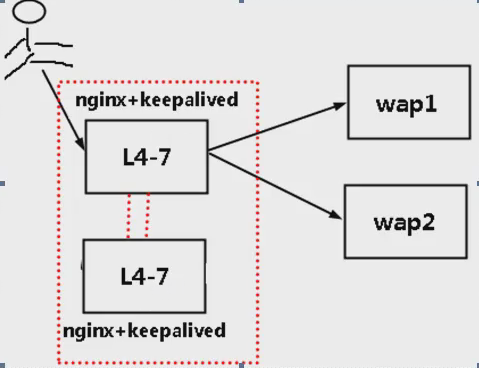
proxy\_pass http://dynamic\_pools;

include proxy.conf;

}

## 1.4 nginx负载均衡高可用

### 1.4.1 配置keepalive实现nginx负载均衡的高可用



keepalived单纯的ip漂移更简单，如果对资源服务有控制，heartbeat更适合。

配置nginx负载均衡器高可用的配置：

1） 两台机器都安装配置好nginx负载均衡器

2） 两台机器都安装配置好keepalive软件。

3） 选择提供服务的VIP，在keepalive中进行配置。

4） 确保两台负载均衡器的nginx都处于启动状态，监听本机的IP地址即可。

5） 这样主nginx负载均衡宕机时，备用负载均衡接管VIP，使得用户请求自身的nginx代理服务。

有个隐患：主服务器没有宕机，但是nginx负载均衡服务宕机了，这样的情况keepalive服务不会切换VIP相当于IP地址存在，但服务已经关闭了。所以，无法提供服务。

最简单的，在nginx负载均衡所在服务器本地，通过定时任务（守护进程）运行一个脚本，当nginx进程（或URL）不存在时，停掉keepalive服务，强制VIP切换转移。

**要严格确保VIP地址和nginx服务器始终在一台服务器上。**

proxy\_next\_upstream健康检查

server{

listen 80;

server\_name [www.etiantian.org](http://www.etiantian.org);

location / {

proxy\_pass <http://static_pools>;

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;

include proxy.conf;

}

}

nginx尝试连接后端主机失败的次数，这个数值是配合proxy\_next\_upstream,fastcgi\_next\_upstream,and memcached\_next\_upstream这三个参数来使用的，当nginx接收后端服务器返回这三个参数定义的状态码的时候，会将这个请求转发给正常工作的后端服务器，例如404,502,503。 Max\_fails默认值是1；

## 1.5 nginx负载均衡检测节点状态

脚本实现

## 1.6 nginx cache

<http://zyan.cc/nginx_cache>

# 2. keepalive高可用集群

## 2.1 keepalive服务介绍

keepalive起初是专为LVS设计的，**专门用来监控LVS集群系统红各个服务节点的状态**，后来又加入了VRRP的功能，因此不了配合LVS服务外，也可以作为其他服务（nginx，haproxy）的高可用软件，VRRP是virtual router redundancy protocol（虚拟路由器冗余协议）的缩写，VRRP出现的目的就是为了解决**静态路由**出现的单点故障问题，他能够保证网络的不间断、稳定的运行。所以，keepalive一方面具有LVS（cluster nodes healthchecks）功能，另一方面也具有LVS directors failover功能。

## 2.2 keepalive服务两大用途：healthcheck&failover

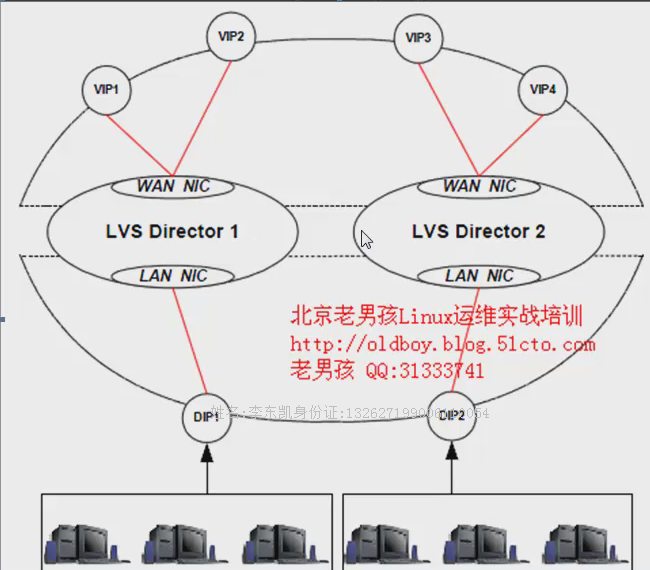
### 2.2.1 LVSdirector是 failover功能

ha failover功能：实现LB Master主机和backup主机之间故障转义和自动切换。

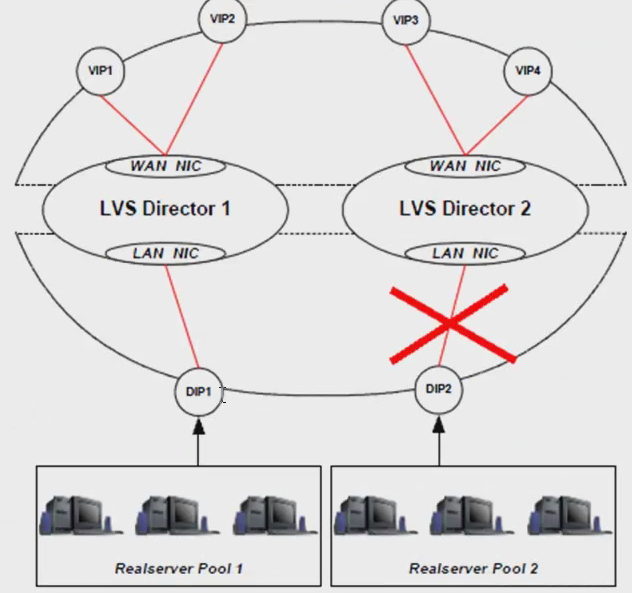
这是针对有**两个负载均衡器**director同时工作而采取的故障转移措施。当主负载均衡失效或出现故障时，备份负载均衡将自动接管主负载均衡的所有工作（VIP资源及相应服务）；一旦主LB故障修复，又会接管回它原来处理的工作，而备LB会释放master失效时它接管的工作，此时两者将恢复到最初各自的角色状态。

### 2.2.2 LVS directors failover功能原理图

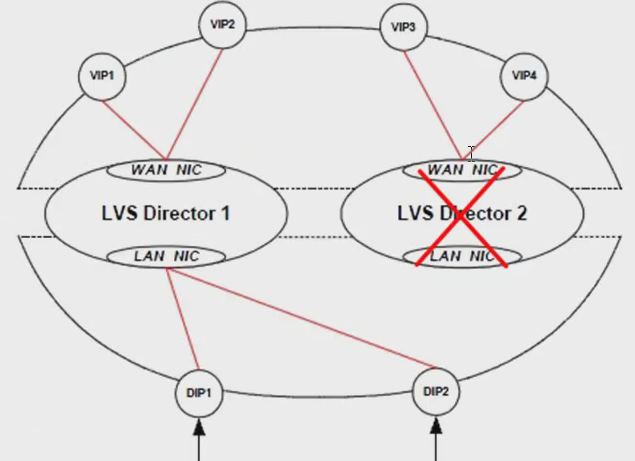
图一：keepalive集群正常工作双主架构简图

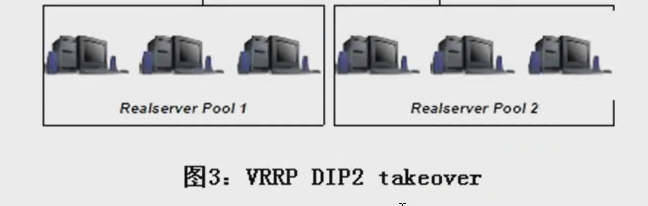


图二：keepalive集群LVS director2宕机状态

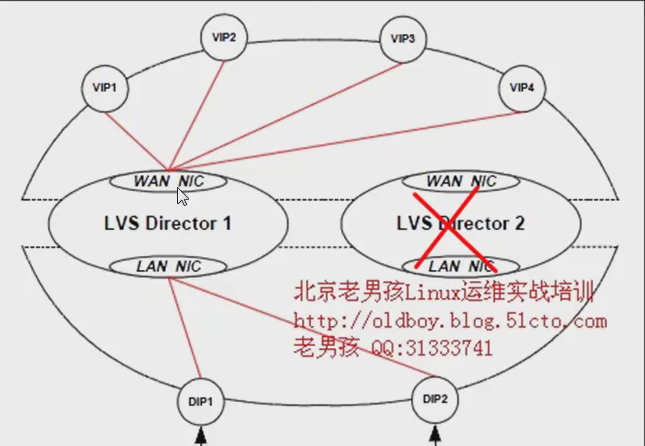


图三：keepalive集群DIP2 takeover图





图四：keepalive集群takeover正常工作架构简图





### 2.2.3 LVS cluster nodes healthchecks功能

1、keepalived.conf里配置就可以实现LVS功能。

2、keepalived可以对LVS下面的集群节点做健康检查。

re healthcheck功能：**负载均衡定期检查RS的可用性决定是否给其分发请求。**

当虚拟服务器中的某一个甚至是几个真是服务器同时发生故障无法提供服务时，负载均衡器会自动将失效的RS服务器从转发队列中清除出去，从而保证用户的访问不受影响；当故障的RS服务器被修复以后，系统又会自动把它们加入转发队列，分发请求提供正常服务。

## 2.3 keepalived故障切换转义原理介绍

keepalived directors高可用对之间的故障切换转移，是通过VRRP协议（virtual router redundancy Protocol虚拟路由器冗余协议）来实现的。

在keepalived Directors正常工作时，主director节点会不断的向备节点广播心跳消息，用以告诉备节点自己还活着，当主节点发生故障时，备节点就无法继续检测到主节点的心跳，进而调用自身的接管程序，接管主节点的IP资源及服务。而当主节点恢复故障时，备节点会释放主节点故障时自身接管的IP资源及服务，恢复到原来的自身的备用角色。

那么，什么是VRRP协议呢？

### 2.3.1 VRRP协议简单介绍

VRRP协议，全称Virtual Router Redundancy Protocol，中文名，虚拟路由冗余协议，VRRP的出现就是为了**解决静态路由的单点故障**，VRRP是通过一种**竞选协议机制**来将路有任务交给某台VRRP路由器。

MASTER和BACKUP

在一个VRRP虚拟路由器中，有多台物理的VRRP路由器，但是这多台物理的机器并不同时工作，而是由一台称为MASTER的负责路由工作，其他的都是BACKUP，MASTER并非一成不变，VRRP协议让每个VRRP路由器参与竞选，最终获胜的就是MASTER。MASTER有一些特权，比如拥有虚拟路由器的IP地址，我们的主机就是用这个IP地址作为静态路由的。拥有特权的MASTER要负责转发发送给网关地址的包和响应ARP请求。

VRRP通过竞选协议来实现虚拟路由器的功能，所有的协议报文都是通过IP多播（multicast）包（多播地址224.0.0.18）形式发送的。虚拟路由器由VRID（范围0-255）和一组IP地址组成，对外表现为一个周知的MAC地址：00-00-5E-00-01-{VRID}。所以，在一个虚拟路由器中，不管谁是MASTER，对外都是相同的MAC和IP（称之为VIP）。客户端主机并不需要因为MASTER的改变而修改自己的路由配置，对他们来说，这种主从的切换是透明的。

在一个虚拟路由器中，只有作为MASTER的VRRP路由器会一直发送VRRP广告包（VRRP Advertisement message），BACKUP不会抢占MASTER，除非它的优先级（priority）更高。当MASTER不可用时（BACKUP收不到广告包），多台BACKUP中优先级最高的这台会被抢占为MASTER。这种抢占是非常快速的（<ls），以保证服务的连续性。

处于安全性考虑，VRRP包使用了加密协议进行加密。

**小结：keepalive工作原理**

VRRP协议介绍：

1） VRRP协议，全称：虚拟路由器冗余协议，VRRP的出现就是为了解决静态路由的单点故障。

2） VRRP是通过一种竞选协议机制来将路有任务交给某台VRRP路由器。

3） VRRP通信是通过IP多播的方式实现通信。

4） 主发包，备接包，当备接不到包的时候，就启动接管程序，接管主的资源。备可以有多个，通过优先级竞选。

5） VRRP使用了加密协议。

**keepalived工作原理**

keepalived高可用对之间是通过VRRP协议通信的，VRRP协议是通过竞选机制来确定主备的，主的优先级高于备，因此，工作时主会获得所有的资源，备节点处于等待状态，当主挂了的时候，备节点，接管主节点的资源，然后顶替主节点对外提供服务。

VRRP协议是通过IP多播包的方式（224.0.0.18）发送的。

在keepalived之间，只有作为主的服务器会一致发送VRRP广播包，告诉备他还活着，此时备不会抢占主，当主不可用时，即备监听不到主发送的广播包时，就会启动相关服务接管资源，保证业务的连续性。接管速度可以小于1秒。VRRP使用加密协议加密发送广播包。

## 2.4 配置keepalived实现服务高可用功能

keepalived的官方站点是<http://www.keepalived.org>，

官方文档是：<http://www.keepalived.org/documentation.html>

### 2.4.1 下载keepalived软件包

mkdir -p /home/application/tools

cd /home/application/tools

wget http://www.keepalived.org/software/keepalived-1. 2.16.tar.gz

### 2.4.2 安装keepalived

提示：安装强请确认路径：

--with-kernel-dir=/usr/src/kernels/2.6.32-573.el6.x86\_64，如果没有此路径，请看下面的补充安装

ln -s /usr/src/kernels/2.6.32-573.el6.x86\_64/ /usr/src/linux

提示：安装时，偶可能会没有/usr/src/kernels/2.6.32-573.el6.x86\_64，这是因为缺少kernel-devel-2.6.32-573.el6.x86\_64软件包，此时需要提前通过yum install kernel-devel –y 命令来安装。

tar zxf keepalived-1.2.16.tar.gz

cd keepalived-1.2.16

./configure

make

make install

cp /usr/local/etc/rc.d/init.d/keepalived /etc/init.d/ #生成启动脚本命令

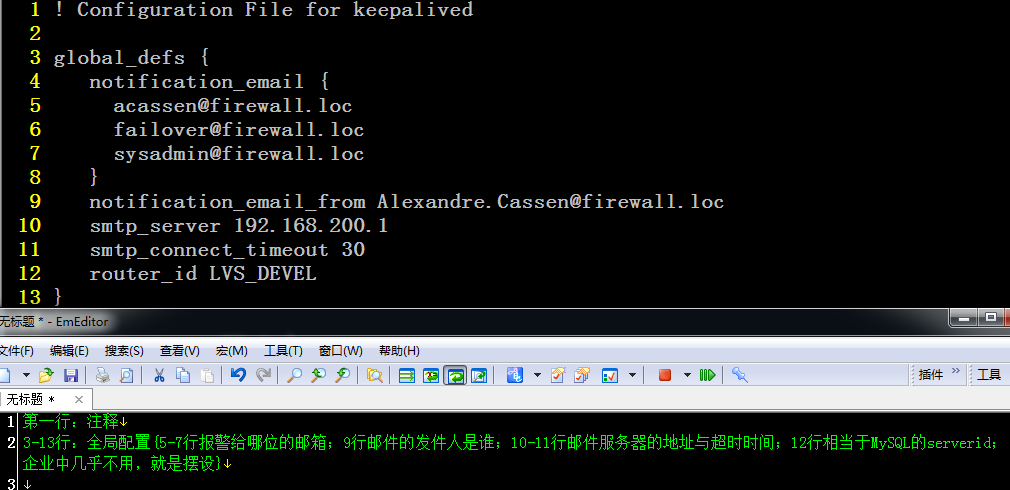
cp /usr/local/etc/sysconfig/keepalived /etc/sysconfig/ #配置启动脚本的参数

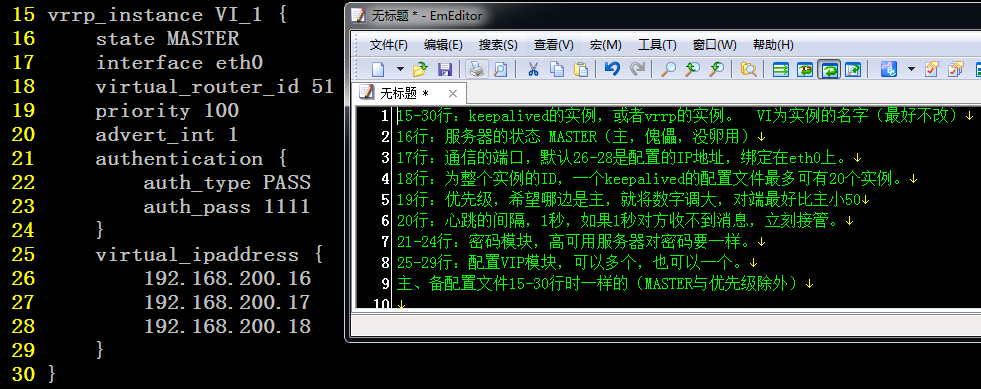
mkdir /etc/keepalived #常见默认的keepalived配置文件路径。

cp /usr/local/etc/keepalived/keepalived.conf /etc/keepalived/ #将keepalived.conf模板拷贝到/etc/keepalived下

cp /usr/local/sbin/keepalived /usr/sbin

/etc/init.d/keepalived start 或 stop





第一行：注释

3-13行：全局配置{5-7行报警给哪位的邮箱；9行邮件的发件人是谁；10-11行邮件服务器的地址与超时时间；12行相当于MySQL的serverid；企业中几乎不用，就是摆设}

15-30行：keepalived的实例，或者vrrp的实例。 VI为实例的名字（最好不改）

16行：服务器的状态 MASTER（主，傀儡，没卵用）

17行：通信的端口，默认26-28是配置的IP地址，绑定在eth0上。

18行：为整个实例的ID，一个keepalived的配置文件最多可有20个实例。

19行：优先级，希望哪边是主，就将数字调大，对端最好比主小50

20行：心跳的间隔，1秒，如果1秒对方收不到消息，立刻接管。

21-24行：密码模块，高可用服务器对密码要一样。

25-29行：配置VIP模块，可以多个，也可以一个。

主、备配置文件15-30行时一样的（MASTER与优先级除外）

keepalived主机MASTER配置文件：

[root@lb01 keepalived]# cat keepalived.conf

! Configuration File for keepalived

global\_defs {

notification\_email {

343012818@qq.com

}

notification\_email\_from Alexandre.Cassen@firewall.loc

smtp\_server 192.168.200.1

smtp\_connect\_timeout 30

router\_id LVS\_01

}

vrrp\_instance VI\_1 {

state MASTER

interface eth0

virtual\_router\_id 51

priority 150

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

10.1.0.17/24

}

}

keepalived备机BACKUP配置文件：

[root@lb02 keepalived]# cat keepalived.conf

! Configuration File for keepalived

global\_defs {

notification\_email {

343012818@qq.com

}

notification\_email\_from Alexandre.Cassen@firewall.loc

smtp\_server 192.168.200.1

smtp\_connect\_timeout 30

router\_id LVS\_02

}

vrrp\_instance VI\_1 {

state BACKUP

interface eth0

virtual\_router\_id 51

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

10.1.0.17/24

}

}

### 2.4.3 keepalived检测裂脑脚本

可以ping通主，备节点有VIP就认为裂脑

[root@lb02 scripts]# cat check\_split\_brain.sh

#!/bin/sh

while true

do

ping -c 2 -W 3 10.1.0.5 &>/dev/null

if [ $? -eq 0 -a `ip add|grep 10.1.0.17|wc -l` -eq 1 ];then

echo "ha is split brain.warning."

else

echo "ha is ok"

fi

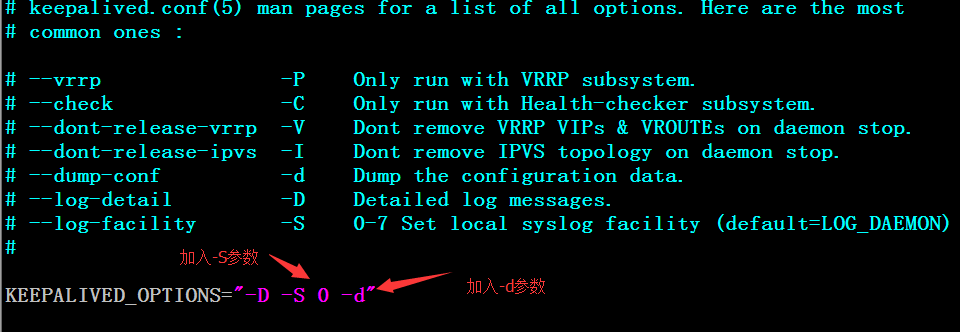
sleep 5

done

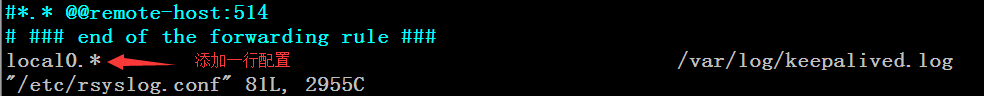
### 2.4.4 修改keepalived配置日志的操作：

默认keepalived日志是在message中，通过修改配置将日志放入到keepalived.log文件

vim /etc/sysconfig/keepalived



vim /etc/rsyslog.conf



/etc/init.d/rsyslog restart

/etc/init.d/keepalived restart

重启服务后，在看一下/var/log/keepalived.log就有日志文件了。

### 2.4.5 keepalived多实例：

vrrp\_instance VI\_2 {

state MASTER

interface eth0

virtual\_router\_id 52

priority 150

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

10.1.0.18/24

}

}

就是在主配置文件中添加15行配置即可。

# 3. lvs+keepalived集群架构服务应用

## 3.1 LVS负载均衡集群介绍

### 3.1.1 搭建负载均衡服务的需求

负载均衡（load Balance）集群提供了一种**廉价、有效、透明**的方法，来扩展网络设备和服务器的**负载、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。

**搭建负载均衡服务的需求**

1）把单台计算机无法承受的大规模的并发访问或数据流量分担到多台节点设备上分别处理，减少用户等待响应的时间，提升用户体验；

２）　单个重负载的运算分担到多台节点设备上做并行处理，每个节点设备处理结束后，将结果汇总，返回给用户，系统处理能力得到大幅度提高。

３） 7\*24的服务保证，任意一个或多个有限后面节点设备宕机，要求不能影响业务。

在负载均衡集群中，所有计算机都应该提供相同的服务。集群负载均衡器截获所有对该服务的入站请求。然后将这些请求尽可能的平均低分配所有集群节点上。

### 3.1.2 LVS（Linux virtual server）介绍

LVS是Linux Virtual Server的简写，意即Linux虚拟服务器，是一个虚拟的服务器集群系统，可以在NUIX/LINUX平台下实现负载均衡集群功能。该项目在1998年5月由章文嵩博士组织成立，是中国宫内最早出现的自由软件项目之一。

LVS 官方网站

|  |  |
| --- | --- |
| **标题** | **地址** |
| LVS项目介绍 | <http://www.linuxvirtualserver.org/zh/lvs1.html> |
| LVS集群的体系结构 | <http://www.linuxvirtualserver.org/zh/lvs2.html> |
| LVS集群中的IP负载均衡技术 | <http://www.linuxvirtualserver.org/zh/lvs3.html> |
| LVS集群的负载调度 | <http://www.linuxvirtualserver.org/zh/lvs4.html> |

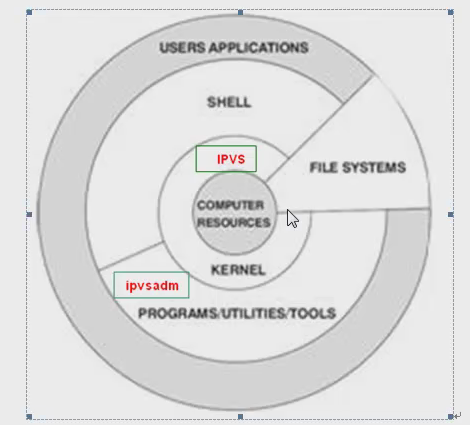
### 3.1.3 IPVS（LVS）发展史

早在2.2内核时，IPVS就已经以内核补丁的形式出现。

从2.4.23 版本开始，IPVS软件就是合并到Linux内核的常用版本的内核补丁的集合。

从2.4.24 以后IPVS已经成为Linux官方标准内核的一部分。

### 3.1.4 IPVS软件工作层次图



从上图看出，LVS负载均衡调度技术是在Linux内核中实现的，因此，被称为Linux虚拟服务器（Linux virtual server），**使用该软件配置LVS时候，不能直接配置内核中的ipvs，而需要使用ipvs的管理工具ipvsadm进行管理，可以通过keepalived软件直接管理ipvs，并不是通过ipvsadm来管理**ipvs。

ipvs的管理工具ipvsadm管理ipvs。

LVS 技术点小结：

1、 真正实现负载调度的工具是IPVS，工作在Linux内核层面。

2、 LVS自带的IPVS管理工具是ipvsadm。

3、 keepalived实现管理IPVS及对负载均衡器的高可用。

4、 redhat工具piranha web管理实现调度的工具IPVS。

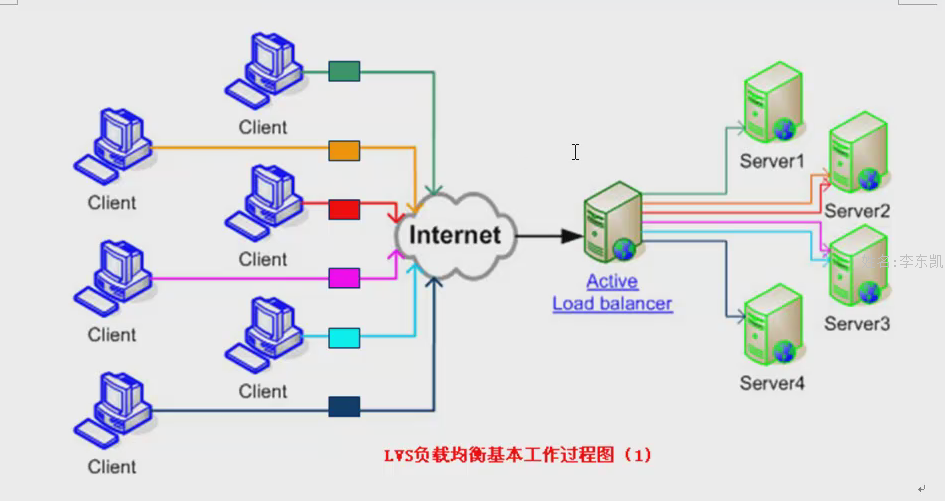
### 3.1.5 LVS系统结构与工作原理简单描述

LVS集群负载均衡器接收服务的所有入站客户端计算机请求，并根据调度算法决定哪个集群节点应该处理回复请求。负载均衡器（检查LB）有时也被称为LVS Director（简称Director）。

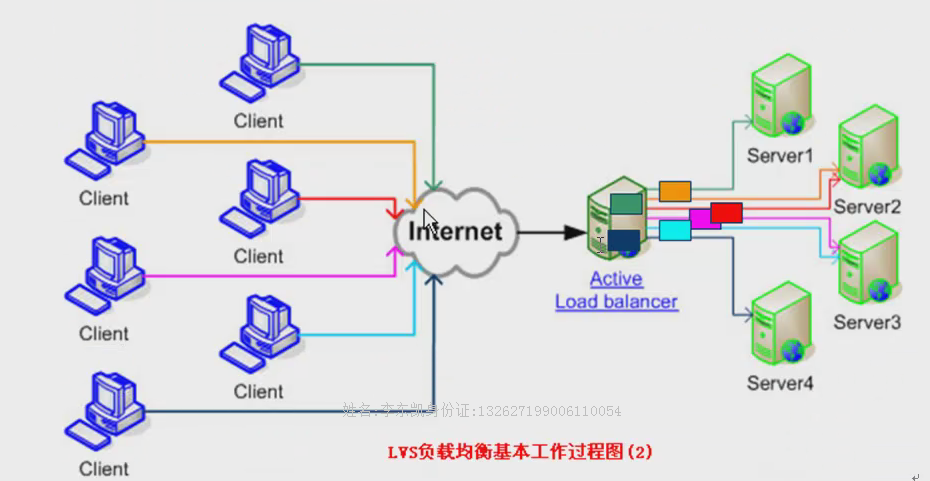
LVS虚拟服务器的系统结构如下图所示，一组服务器通过高速的局域网或地理分布的广域网相互连接，在他们的前端有一个负载调度器（Load Balancer）。负载调度器能无缝的将网络请求调度到真实服务器上，从而使得服务器集群的结构对客户是透明的，客户访问集群系统提供的网络服务就想访问一台高性能、高可用的服务器一样。客户程序不受服务器集群的影响不需做任何修改。系统的伸缩性通过在服务集群中透明的加入和删除一个节点来达到，通过检测你节点或服务进程故障和正确的系统达到高可用性。由于我们的负载调度技术是在Linux内核中实现的，我们称之为Linux虚拟服务器。

### 3.1.6 LVS基本工作过程图

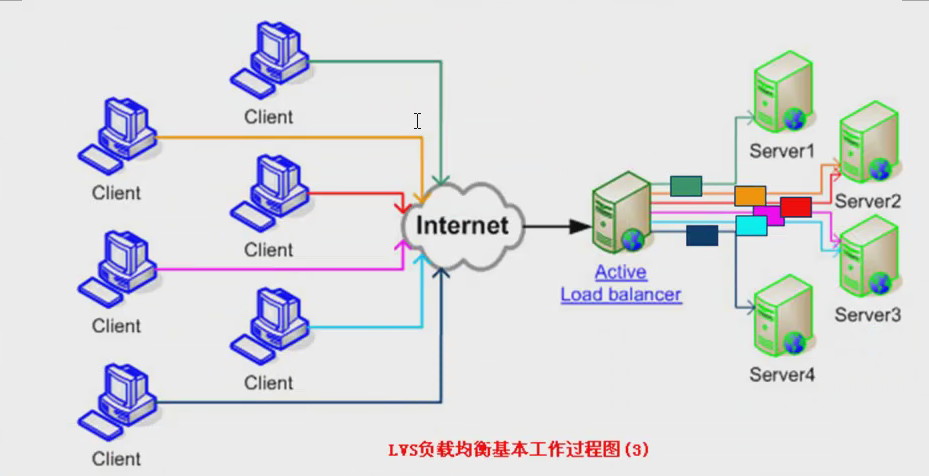
LVS基本工作过程图1：带颜色的小方块代表不同的客户端请求：

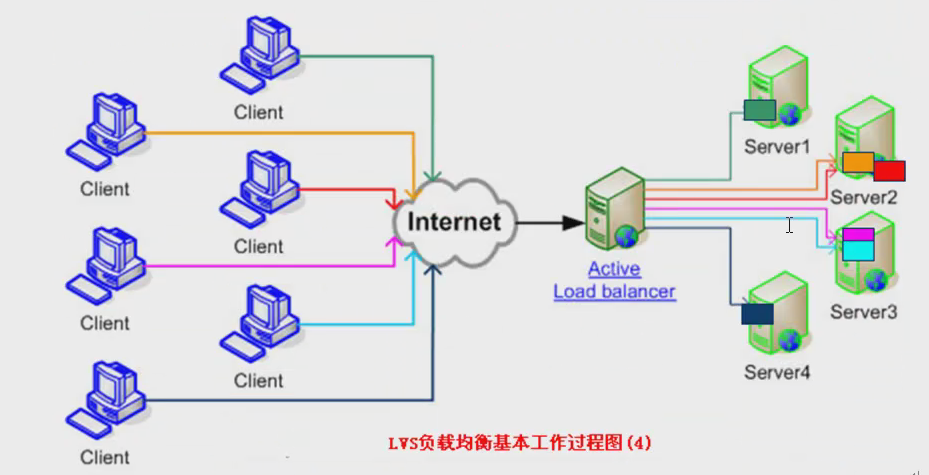


LVS基本工作过程图2：不同的客户端请求小方块经过负载均衡器，通过制定的分配策略被分发到后面的机器上。



LVS基本工作过程图3

LVS基本工作过程图4：



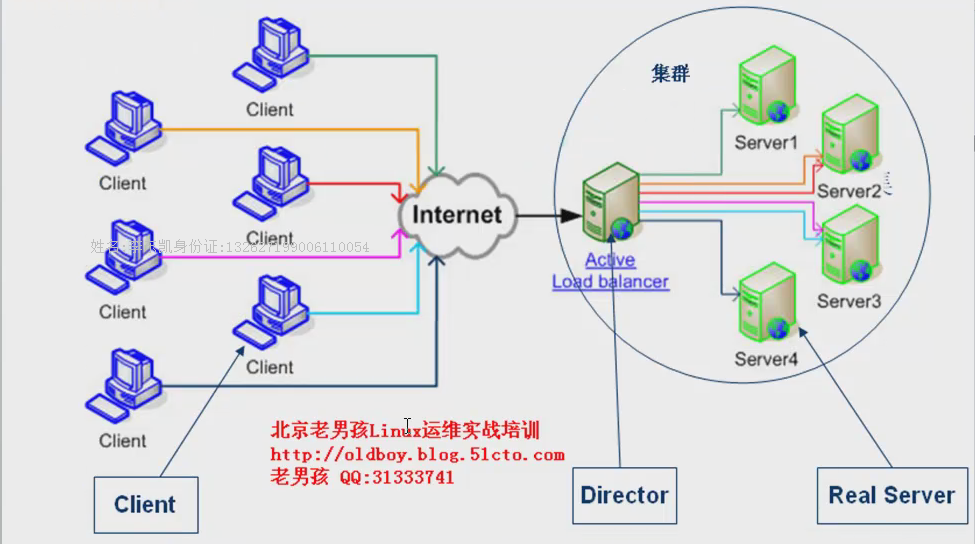
### 3.1.7 LVS相关术语命名约定

为了方便探讨LVS技术，LVS社区提供了一个命名的约定，内容如下表：

|  |  |  |
| --- | --- | --- |
| **名称** | **缩写** | **说明** |
| 虚拟IP地址（Virtual ip address） | **VIP** | VIP为director用于向客户端计算机提供服务的IP地址。 |
| 真是IP地址（real server ip address） | **RIP** | 在集群下面节点上使用的IP地址，物理IP地址。 |
| director的IP地址（director IP address） | **DIP** | director用于连接内外网络的IP地址，物理网卡上的IP地址。是负载均衡器上的IP。 |
| 客户端主机IP地址（client IP address） | CIP | 客户端用户计算机请求集群服务器的IP地址，该地址用做发送给集群的请求的源IP地址。 |

LVS集群内部的节点称为真实服务器（real server）也叫作集群节点。请求集群服务器的计算机称为客户端计算机。

与计算机通常在网上交换数据包的方式相同，客户端计算机、director和真是服务器使用IP地址彼此进行通信。



### 3.1.8 LVS集群的3中工作模式介绍与原理讲解

IP虚拟服务器软件IPVS

在调度器的实现技术中，IP负载均衡技术是效率醉倒的。在已有的IP负载均衡技术中可用的虚拟服务器，我们称之为VS/NAT技术（virtual server via network address translation），大多数商业化的IP负载均衡调度器产品都是使用NAT的方法，如Cisco的LocalDirector、F5、Netscaler的big/ip和alteon的ACEDirector。

在分析VS/NAT的缺点和网络服务的非对称的基础上，我们提出通过IP隧道实现虚拟服务器的方法VS/TUN（virtual server via IP tunneling）和通过直接路由实现虚拟服务器的方法VS/DR（virtual server via direct Routing），他们可以极大地提高系统的伸缩性。所以，IPVS软件实现了这三种IP负载均衡技术，他们的大致原理如下。淘宝开源的模式FULLNAT。FULLNAT（）

### 3.1.9 什么是ARP协议

ARP协议，全称“Address Resolution Protocol”，中文名是**地址解析协议**，使用ARP协议可**实现通过IP地址获得对应主机的物理地址（MAC地址）。**

ARP小结：

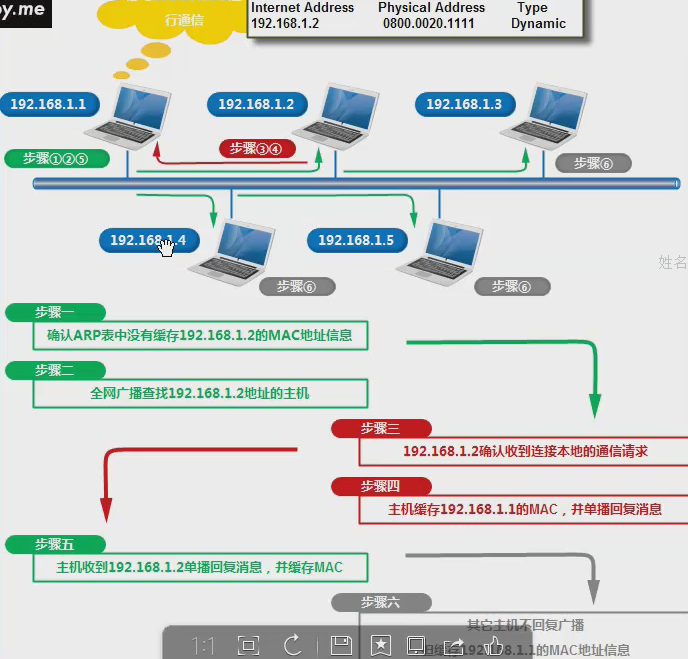
1、 ARP全称Address Resolution Protocol。

2、 作用，实现局域网内通过IP地址获取主机的MAC地址。

3、 MAC地址是48位主机的物理地址，局域网内唯一。

4、 ARP协议类似DNS服务，但不需要配置服务。

5、 ARP协议是三层协议。



MAC（media Access Control或者Medium Access Control）地址，意译为媒体访问控制，或称为物理地址、硬件地址，用来定义网络设备的位置。在OSI模型中，第三层网络层负责IP地址，第二层数据链路层负责MAC地址。因此一个主机会有一个MAC地址，而每个网络位置会有一个专属于它的IP地址。

#### 3.1.9.1 ARP缓存表

在每台安装藕TCP/IP协议的电脑里都会有一个ARP缓存表（Windows命令提示符里输入ARP -a即可），表里的IP地址与MAC地址是一一对应的，例如：

C:\Users\admin>arp -a

接口: 192.168.0.103 --- 0xe

Internet 地址 物理地址 类型

192.168.0.1 bc-46-99-73-44-32 动态

192.168.0.255 ff-ff-ff-ff-ff-ff 静态

arp常用命令

arp -a 查所有记录

arp -d 清除

arp -s 绑定IP和MAC

企业案例：

路由器不稳定，公司增加了新的路由器，要求无缝切换，如何实现？

解答：

1、 旧的路由器配置导入到新的配置里。

2、 arping，使用这个命令可以让客户端失效，指定IP地址的ARP失效。

#### 3.1.9.2 ARP缓存表是把双刃剑

1） 主机有了ARP缓存表，可以加快ARP的解析速度，减少局域网内广播风暴。

2） 正是有了ARP缓存表，给恶意黑客带来了攻击服务器主机的风险，这个就是ARP欺骗攻击。

3） 切换路由器，负载均衡器等设备时，可能会导致短时网络中断。

#### 3.1.9.3 为什么要使用ARP协议

OSI模型把网络工作分为七层，彼此不直接打交道，只通过接口（layer interface）。IP地址工作在第三层，Mac地址工作在第二层。当协议在发送数据包时，需要先封装第三层IP地址，第二层MAC地址的报头，但协议之知道目的节点的IP地址，不知道目的节点的MAC地址，又不能跨第二、三层，所以得用ARP协议服务，来帮助获取到目的节点的MAC地址。

**请问：ARP协议是二层协议，还是三层协议？**

答：**ARP协议属于三层**，**工作在二层。**

#### 3.1.9.4 ARP在生产环境产生的问题及解决办法

1） ARP病毒，ARP欺骗。

2） 高可用服务器对之间切换时要考虑ARP缓存的问题。

3） 路由器等设备无缝时要考虑ARP缓存的问题，例如：更换办公室的路由器。

#### 3.1.9.5 ARP欺骗原理

**ARP攻击就是通过伪造IP地址和MAC地址对实现ARP欺骗的**，如果一台主机中了ARP病毒，那么他就能够在网络中产生大量的ARP通信量（它会以很快的频率进行广播），以至于使网络阻塞，攻击者只要持续不断的发出伪造的ARP响应包就能**更改局域网中目标主机ARP缓存中的IP-MAC条目**，造成网络中断或中间人攻击。

ARP攻击主要是存在于局域网网络中，局域网中若有一个人感染ARP木马，则感染该ARP木马的系统将会试图通过“ARP欺骗”手段获取所在网络内其他计算机的通信信息，并因此造成网内其他计算机的通信故障。

解决ARP病毒的方法：

1、 员工入职时，会记录MAC记录并绑定，当中病毒后使用ARP -a 命令查看是哪台设备中毒

#### 3.1.9.6 服务器切换ARP问题

当网络中一台提供服务的机器宕机后，当在其他运行正常的机器添加宕机的机器的IP时，会因为客户端的ARP table cache的地址解析还是宕机的机器的MAC地址。从而导致，即使在其他运行正常的机器添加宕机的机器的IP，也会发生客户依然无法访问的情况。

**解决办法是**：当机器宕机，IP地址迁移到其他机器上时，需要通过arping命令来通知所有网络内机器清除其本地的ARP table cache，从而使得客户机访问时重新广播获取MAC地址。

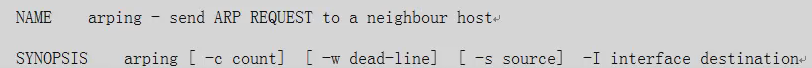
这个在自己开发脚本实现服务器的高可用时是要必须考虑的问题之一，几乎所有的高可用软件都会考虑这个问题。

ARP广播而进行新的地址解析。

Linux下的具体命令：

/sbin/arping -I ech0 -c 3 -s 10.0.0.162 10.0.0.253

/sbin/arping -U -I eth0 10.0.0.162



#### 3.1.9.7 ARP技术点

1、 什么是ARP协议

2、 ARP协议工作原理。

3、 工作中ARP带来的实际问题和解决方案

a.局域网ARP欺骗原理及解决方案。

b.切换网关路由器，ARP表带来的问题。

c.集群架构中高可用服务器对之间的切换，ARP表带来的问题。

4、 局域网客户端ARP问题防御。

#### 3.1.9.8 ARP技术资料

1） ARP相关细节介绍

<http://book.chinaunix.net/special/ebook/oreilly/Understanding_Linux_Network_lnternals/>

2） VIP ARP相应过程

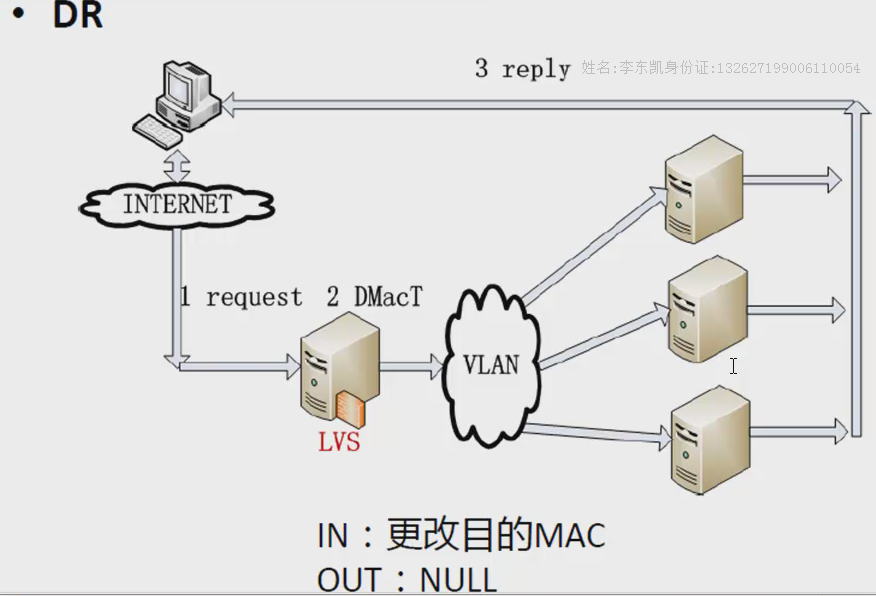
<http://book.chinaunix.net/special/ebook/oreilly/Understanding_Linux_Network_lnternals/0596002556/understandlni-CHP-28-SECT-5.html>

## 3.2 LVS DR模式

### 3.2.1 LVS DR模式原理：

1、 通过更改数据包的目标MAC地址实现数据包的转发的。

2、 所有节点和LVS要处于一个局域网。



LVS只接收入站请求

DR模式的特点：

1、 通过在**调度器LB上修改数据包的目的MAC地址实现转发**。注意，源IP地址仍然是CIP，目的IP地址仍然是VIP。

2、 请求的报文经过调度器，而RS响应处理后的报文无需经过调度器LB，因此，并发访问量大时使用效率很高（和NAT模式比）

3、 因DR模式是通过MAC地址的改写机制实现的转发，因此，所有RS节点和调度器LB只能在一个局域网LAN中（小缺点）。

4、 乣注意RS节点的VIP的绑定（lo:vim/32,lo1:vip/32）和ARP抑制问题。

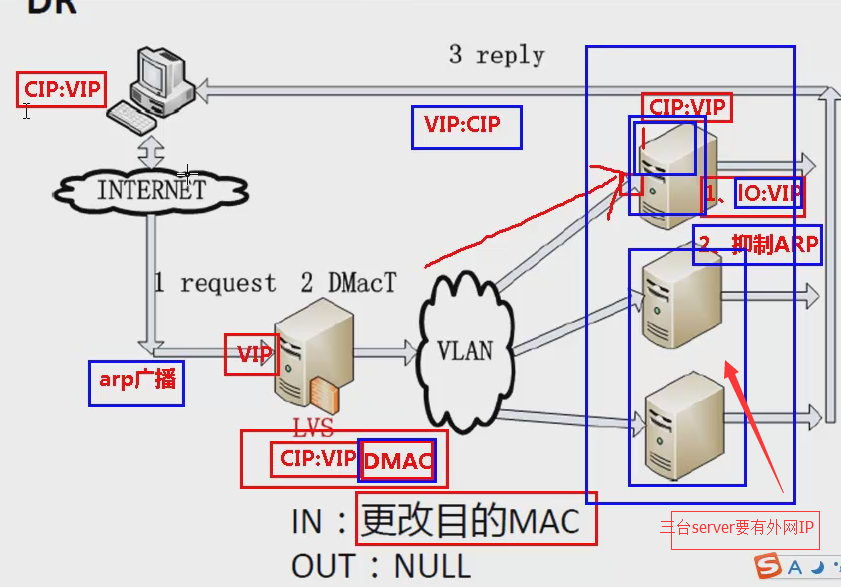
5、 强调下：RS节点的默认网关不需要是调度器LB的DIP，而直接是IDC机房分配的上级路由器的IP（这是RS带有外网IP地址的情况），理论讲：只要RS可以出网即可，不是必须要配置外网IP。

6、 由于DR模式的调度器仅进行了目的MAC地址的改写，因此，调度器LB无法改变请求的报文的目的端口（和NAT要区别）。

7、 当前，调度器LB支持几乎所有的Unix、Linux系统，但目前不支持Windows系统。真实服务器RS节点可以是Windows系统。

8、 总的来说DR模式效率很高，但是配置也较麻烦，因此，访问量不是特别大的公司可以用haproxy/nginx取代之。这符合运维的原则：简单、易用、高效。日1000-2000W PV或并发请求1万以下都可以考虑用haproxy/nginx（LVS NAT模式）

9、 直接对外的访问业务，例如：web服务做RS节点，RS最好用公网IP地址。如果不直接对外的业务，例如：MySQL，存储系统RS节点，最好只用内部IP地址。



## 3.3 LVS NAT模式

**NAT模式特点：**

1、 NAT技术奖请求的报文（通过DNAT方式改写）和响应的报文（通过SNAT方式改写），通过调度器地址重写然后在转发给内部的服务器，报文返回时在改写成原来的用户请求的地址。

2、 只需要在调度器LB上配置WAN公网IP即可，调度器也要有私有LAN IP和内部RS节点通信。

3、 每台内部RS节点的网关地址，必须要配成调度器LB的私有LAN内物理网卡地址（LDIP），这样才能确保数据报文返回时仍然经过调度器LB。

4、 由于请求与响应的数据报文都经过调度器LB，因此，网站访问量大时调度器LB有较大瓶颈，一般要求最多10-20台节点。

5、 NAT模式支持对IP及端口的转换，即用户请求10.0.0.1:80，可以通过调度器转换到RS节点的10.0.0.2:8080（DR和TUN模式不具备的）。

6、 所有NAT内部RS节点只需要私有LAN IP即可。

7、 由于数据包来回都需要经过调度器，因此，要开启内核转发net.ipv4.ip\_forward=1，让然也包括iptables防火墙的forward功能（DR和TUN模式不需要）。

