01 LVS负载基础知识

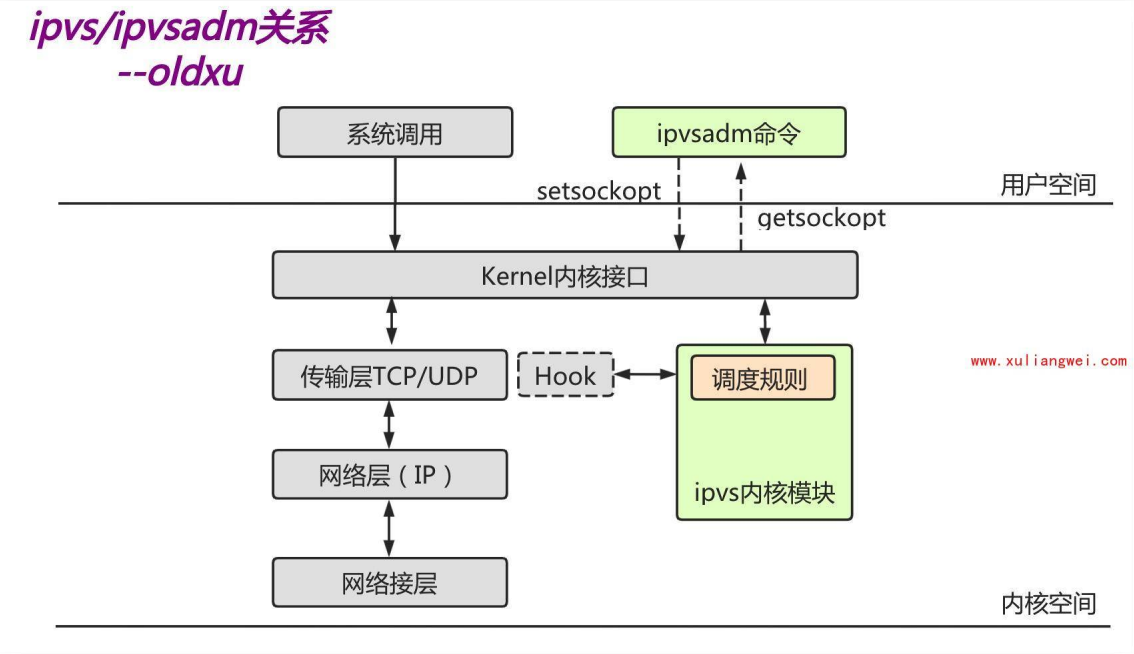
1.LVS基本概述

1.1 什么是LVS

LVS的英文全称是 `Linux Virtual Server`，即 `Linux`虚拟服务器。其实它是一种 `Cluster` 集群技术，主要用于负载均衡，将用户请求均匀的调度到不同的服务器上执行。
注意：`LVS`是基于四层 `IP：PORT` 的负载均衡。

1.2 LVS组成部分

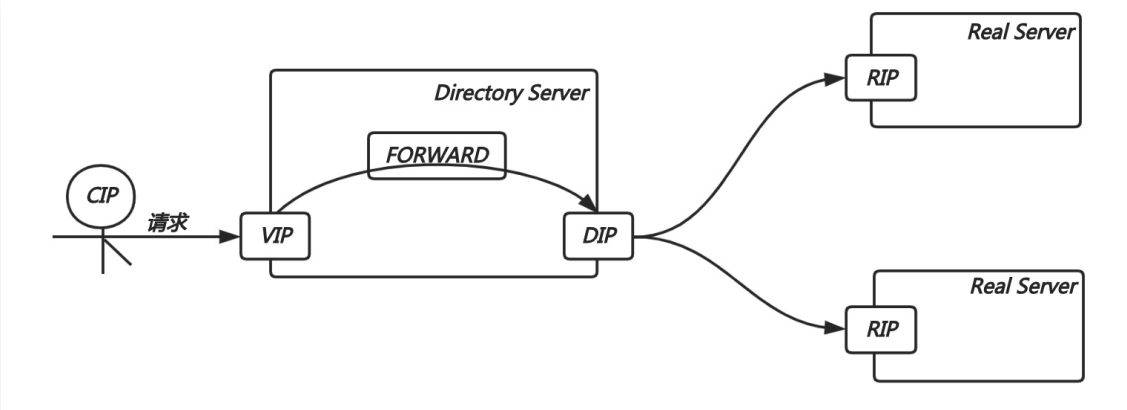
- `ipvs`：工作在内核空间，实现集群服务的"调度"，借鉴了 `iptables` 的实现方式
- `ipvsadm`：工作在用户空间，负责为 `ipvs` 内核框架编写规则。定义谁是集群服务，谁是后端服务器，数据包如何调度，调度到哪个节点。



1.3 LVS相关名词

接下来我们需要了解LVS中的名词，比如：`DS、RS、CIP、VIP、DIP、RIP`，通过下面的图来做了解其含义。

https://www.mockplus.cn/



名称	详细名称	描述
DS	Director Server	目标服务器，即负载均衡器LVS
RS	Real Server	真实应用服务，即后端服务器
CIP	Client IP	客户端请求IP
VIP	Virtual IP	直接面向用户的IP地址，通常为公网IP
DIP	Director Server IP	用于与后端RIP通信的IP地址
RIP	Real Server IP	后端真实服务器的IP地址

2.LVS应用场景

- 腾讯云负载均衡场景示例
- 阿里云负载均衡场景示例
- Ucloud负载均衡场景示例
- 青云负载均衡场景示例

3.LVS常见模型

`LVS` 负载均衡模型有 `NAT、DR、TUN、FULL-NAT`，较为常见的模型有 `NAT、DR`，使用最为广泛的模型是 `DR`

3.1 NAT模型

`NAT`：通过修改请求报文的目标 `IP` 地址，然后根据算法挑选出某台 `RS` 进行转发。（请求进入负载均衡器 `LVS` 时做 `DNAT` ，后端返回数据报文出负载均衡时做 `SNAT`

3.2 DR模型

`DR`：通过修改请求报文的目标 `MAC` 地址，然后根据算法挑选出某台RS进行转发。（请求进入负载均衡器 `LVS` 时做 `MAC` 地址转换，后端返回数据报文不经过负载均衡，所以无需做转换

4.LVS集群命令介绍

- `ipvsadm` 的用法大概分类如下两类、
  - 管理集群服务（定义负载均衡配置）
  - 管理后端RS（定义负载均衡后端节点的增删改查）

```
ipvsadm - Linux Virtual Server administration
ipvsadm -A|E -t|u|f service-address [-s scheduler] [-p [timeout]]
[-M netmask] [--pe persistence_engine] [-b sched-flags]
ipvsadm -D -t|u|f service-address
ipvsadm -C
ipvsadm -R
ipvsadm -S [-n]
ipvsadm -a|e -t|u|f service-address -r server-address [options]
ipvsadm -d -t|u|f service-address -r server-address
ipvsadm -L|l [options]
ipvsadm -Z [-t|u|f service-address]
ipvsadm --set tcp tcpfin udp
ipvsadm --start-daemon state [--mcast-interface interface] [--
syncid sid]
```

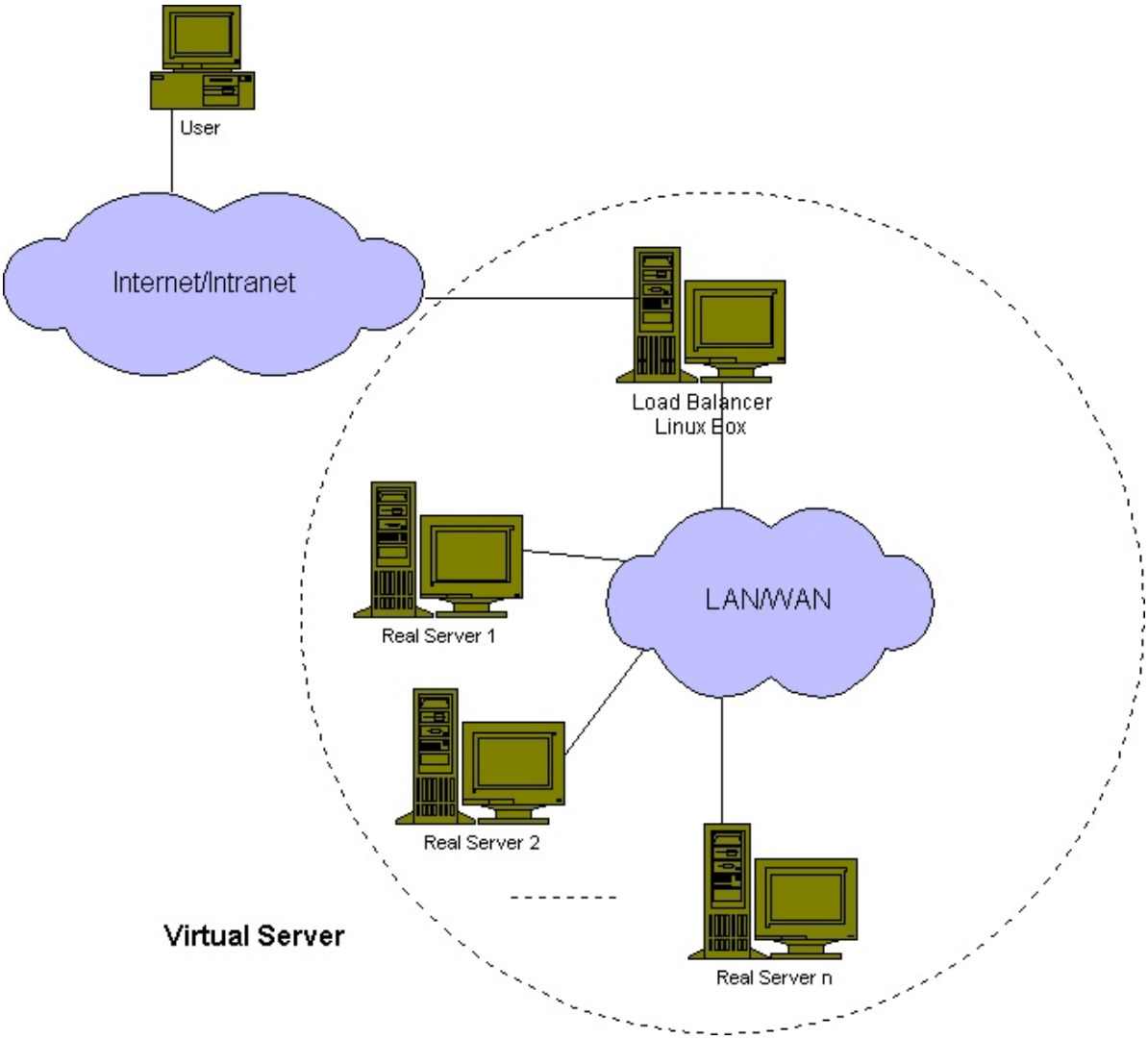
```
ipvsadm --stop-daemon state

# COMMANDS-Cluster
        -A    , --add-service       # 添加一个集群服务
        -E    , --edit-service      # 修改已添加的集群服务
        -D    , --delete-service    # 删除虚拟服务
        -C    , --clear             # 清空集群所有规则
        -R    , --restore           # 从文件中恢复集群
        -S    , --save              # 将集群信息报文至文件中
        -L|-l, --list               # 列出当前集群信息
        -Z    , --zero              # 清空集群计数器
        -n                          # 数字格式显示 ip 和 port，注意-n只
能写在-L之后

# COMMANDS-RS
        -a, --add-server        # 表示要添加 RS 节点
        -e, --edit-server       # 表示要修改 RS 节点
        -d, --delete-server     # 表示要删除 RS 节点
        -t, service-address     # 指定操作哪个节点地址与端口，
host[:port]，tcp协议
        -u, service-address     # 指定操作哪个节点地址与端口，
host[:port]，udp协议
        -r, --real-server       # 指定 RS 节点地址与端口
        -w, --weight            # 指定 RS 节点的权重
        -m, --masquerading      # 指定 LVS 工作模型 （ NAT模型 ）
        -g, --gatewaying        # 指定 LVS 工作模型 （ DR模型 ）
        -i, --ipip              # 指定 LVS 工作模型 （ tun模型 ）
        -s, --scheduler         # 指定 LVS 调度策略，默认为wlc
        -p, --persistent        # 持久连接超时时间
        -f, --fwmark-service    # 防火墙标记
        -c, --connection        # 显示 ipvs 连接信息
```

02 LVS NAT模型实践

1.LVS NAT 模型概念

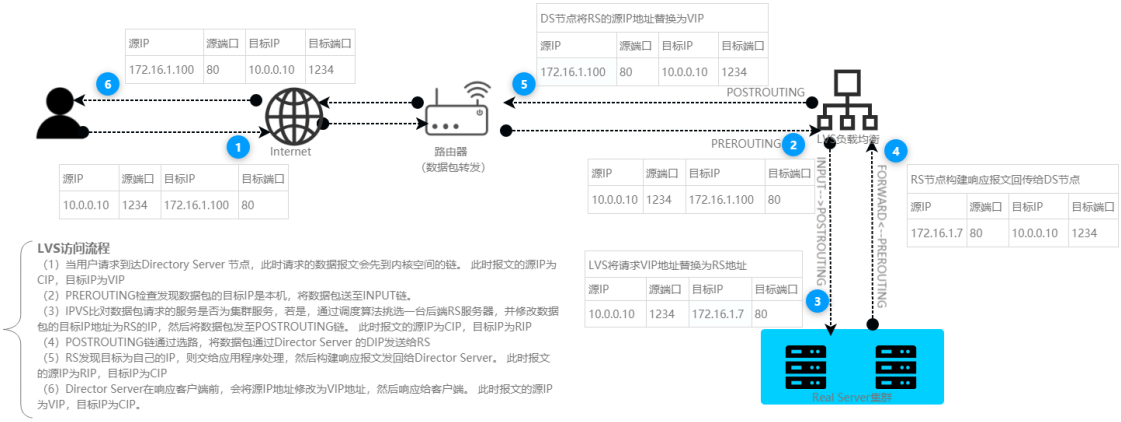
通过修改请求报文的`目标IP`地址，而后根据调度算法挑选出一台`RS`节点进行转发。（请求进入负载均衡器`LVS`时做`DNAT`，后端返回数据出负载均衡时做`SNAT`）

1.1.NAT 基础图解



1.2.NAT 底层实现(New IP)

- 客户端：`10.0.0.1（外网）`
- DS：`172.16.1.100（VIP）`、`172.16.1.3（DIP）`
- RS：`172.16.1.7`、`172.16.1.8`、`Gateway：172.16.1.3`



1.3.NAT访问原理

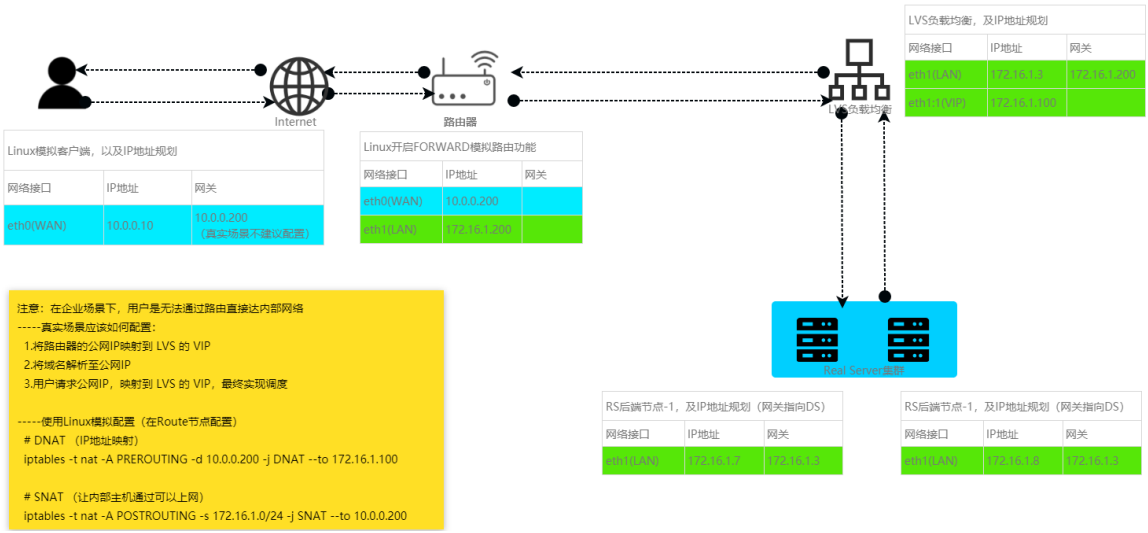
- 1、当用户请求到达 `Director Server`，此时请求的数据报文会先到内核空间的`PREROUTING`链。 此时报文的 `源IP为CIP`， `目标IP为VIP`
- 2、`PREROUTING`检查发现数据包的目标 `IP` 是本机，将数据包送至 `INPUT` 链。
- 3、`IPVS` 比对数据包请求的服务是否为集群服务，若是，通过调度算法挑选一台后端 `RS` 服务器，并修改数据包的`目标IP` 为 `RS的IP`，然后将数据包发至 `POSTROUTING` 链。 此时报文的 `源IP为CIP`， `目标IP为RIP`
- 4、`POSTROUTING` 链通过选路，将数据包通过 `Director Server`的 `DIP` 发送给 `RS`
- 5、`RS`发现目标为自己的 `IP`，则交给应用程序处理，然后构建响应报文发回给 `Director Server`。 此时报文的 `源IP为RIP`， `目标IP为CIP`
- 6、`Director Server` 在响应客户端前，会将源 `IP` 地址修改为 `VIP` 地址，然后响应给客户端。此时报文的 `源IP为VIP`， `目标IP为CIP`

1.4.NAT 特性

- 1、`RS` 必须使用私有地址，并需要将网关指向 `DS`
- 2、`RIP` 和 `DIP` 必须为同一网段内。
- 3、`NAT` 模型支持端口映射。
- 4、`RS` 可以使用任意操作系统。例如 `Linux、Windows` 等。
- 5、请求和响应报文都要经过 `DS`，高负载场景中， `DS` 易称为瓶颈。

2.LVS NAT模型实战

2.1.NAT 架构规划



2.2.NAT Route配置

1.将 `Linux` 服务器配置为路由器，先配置其 `IP` 地址

- `eth0` 配置信息如下

```
[root@route ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.0.200
PREFIX=24
GATEWAY=10.0.0.2         # 指向能出公网的IP
DNS1=223.5.5.5
```

- `eth1` 配置信息如下

```
[root@route ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=172.16.1.200
PREFIX=24
```

2.在 `Route` 节点启用 `FORWARD` 转发功能，实现路由功能。

```
[root@route ~]# echo "net.ipv4.ip_forward = 1" >>
/etc/sysctl.conf
[root@route ~]# sysctl -p
```

2.3.NAT RS配置

1.配置 `RS` 节点 `eth1` 网卡为 `LAN` 模式，然后将网关统一指向 `DS` 服务器。（所有 `RS` 节点都需要操作）

```
[root@rs01 ~]# ifdown eth0     # 关闭 eth0网关，真实生产环境也仅有一块网卡
[root@rs01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=172.16.1.5          # 不同的 RS 节点地址不一样
GATEWAY=172.16.1.3         # 所有的 RS 节点网关都指向 DS 节点的 DIP
PREFIX=24
DNS=223.5.5.5
```

2.重启 `eth1` 网卡，使其生效。

```
[root@rs01 ~]# ifdown eth1 && ifup eth1
```

3.检查 RS 节点路由信息

```
[root@rs01 ~]# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.1.3      0.0.0.0         UG    100    0        0 eth1
172.16.1.0      0.0.0.0         255.255.255.0   U     100    0        0 eth1
```

4.配置后端所有 `RS` 的 `web` 服务，注意 `RS1` 和 `RS2` 页面不一样，方便验证效果。

- `rs1` 节点配置

```
[root@rs01 ~]# yum install nginx -y
[root@rs01 ~]# cat /etc/nginx/conf.d/lvs.oldxu.com.conf
server {
    listen  80;
    server_name lvs.oldxu.com;
    root /opt;

    location / {
        index index.html;
    }
}
[root@rs01 ~]# echo "Web Page RS-Node1" > /opt/index.html
[root@rs01 ~]# systemctl start nginx
```

```
# 本机测试访问
[root@rs01 ~]# curl -HHost:lvs.oldxu.com http://172.16.1.5
Web Page RS-Node1
```

- `rs2` 节点配置

```
[root@rs02 ~]# yum install nginx -y
[root@rs02 ~]# cat /etc/nginx/conf.d/lvs.oldxu.com.conf
server {
    listen  80;
    server_name lvs.oldxu.com;
    root /opt;

    location / {
        index index.html;
    }
}
[root@rs02 ~]# echo "Web Page RS-Node2" > /opt/index.html
[root@rs02 ~]# systemctl start nginx

# 本机测试访问
[root@rs02 ~]# curl -HHost:lvs.oldxu.com http://172.16.1.6
Web Page RS-Node2
```

2.4.NAT DS配置

1.编辑网卡配置，将 `DS` 节点的 `eth1` 网关指向路由节点。

```
[root@lvs-01 ~]# ifdown eth0
[root@lvs-01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=172.16.1.3
PREFIX=24
GATEWAY=172.16.1.200
DNS1=223.5.5.5
```

2.新增 `VIP` 地址的网卡配置，将 `VIP` 绑定到 `eth1:1` 网卡上。

```
[root@lb01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1:1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1:1
DEVICE=eth1:1
ONBOOT=yes
IPADDR=172.16.1.100
PREFIX=24
```

3.重启 `DS` 节点 `eth1、eth1:1` 的网卡

```
[root@lvs-01 ~]# ifdown eth1 && ifup eth1
[root@lvs-01 ~]# ifdown eth1:1 && ifup eth1:1
```

4.开启 `DS` 节点的内核转发功能，不然 `RS` 节点发送的数据包会被丢弃。

```
[root@lvs-01 ~]# echo "net.ipv4.ip_forward = 1" >>
/etc/sysctl.conf
[root@lvs-01 ~]# sysctl -p
```

5.`DS` 节点负载均衡配置

```
# 定义 LVS 集群
[root@lb01 ~]# ipvsadm -A -t 172.16.1.100:80 -s rr

# 添加 RS1、RS2集群节点
[root@lb01 ~]# ipvsadm -a -t 172.16.1.100:80 -r 172.16.1.7:80 -m
[root@lb01 ~]# ipvsadm -a -t 172.16.1.100:80 -r 172.16.1.8:80 -m

# 查看集群状态信息
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 rr
  -> 172.16.1.7:80                Masq    1      0          0
  -> 172.16.1.8:80                Masq    1      0          0
```

2.5.NAT Client测试

1.配置 `Client` 节点 `eth0` 网络。

```
[root@client ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.0.100
PREFIX=24
GATEWAY=10.0.0.200        # 真实场景不可能将客户端网关指向企业的路由器上
```

2.重启 `Client` 节点网络

```
[root@client ~]# systemctl restart network
```

3.使用 `Client` 测试访问效果

```
[root@client ~]# curl -HHost:lvs.oldxu.com http://172.16.1.100
Web Page RS-Node1
[root@client ~]# curl -HHost:lvs.oldxu.com http://172.16.1.100
Web Page RS-Node2
```

4.模拟真实场景，首先删除 `Client` 节点指向 `Route` 网关信息，然后配置 `Route` 打开 `IP映射DNAT`、以及 `SNAT（共享上网功能）`

- 删除 `Client` 节点的网关配置

```
[root@client ~]# sed -i '/GATEWAY/d' /etc/sysconfig/network-scripts/ifcfg-eth0
[root@client ~]# systemctl restart network
```

- 配置 `Route` 路由节点的 `DNAT` 以及 `SNAT`

```
# DNAT （地址映射，企业环境使用）
[root@route ~]# iptables -t nat -A PREROUTING -d 10.0.0.200 -j DNAT --to 172.16.1.100

#SNAT （让内部主机通过路由可以上网）
[root@route ~]# iptables -t nat -A POSTROUTING -s 172.16.1.0/24 -j SNAT --to 10.0.0.200
```

5.最后用 `Client` 再次测试

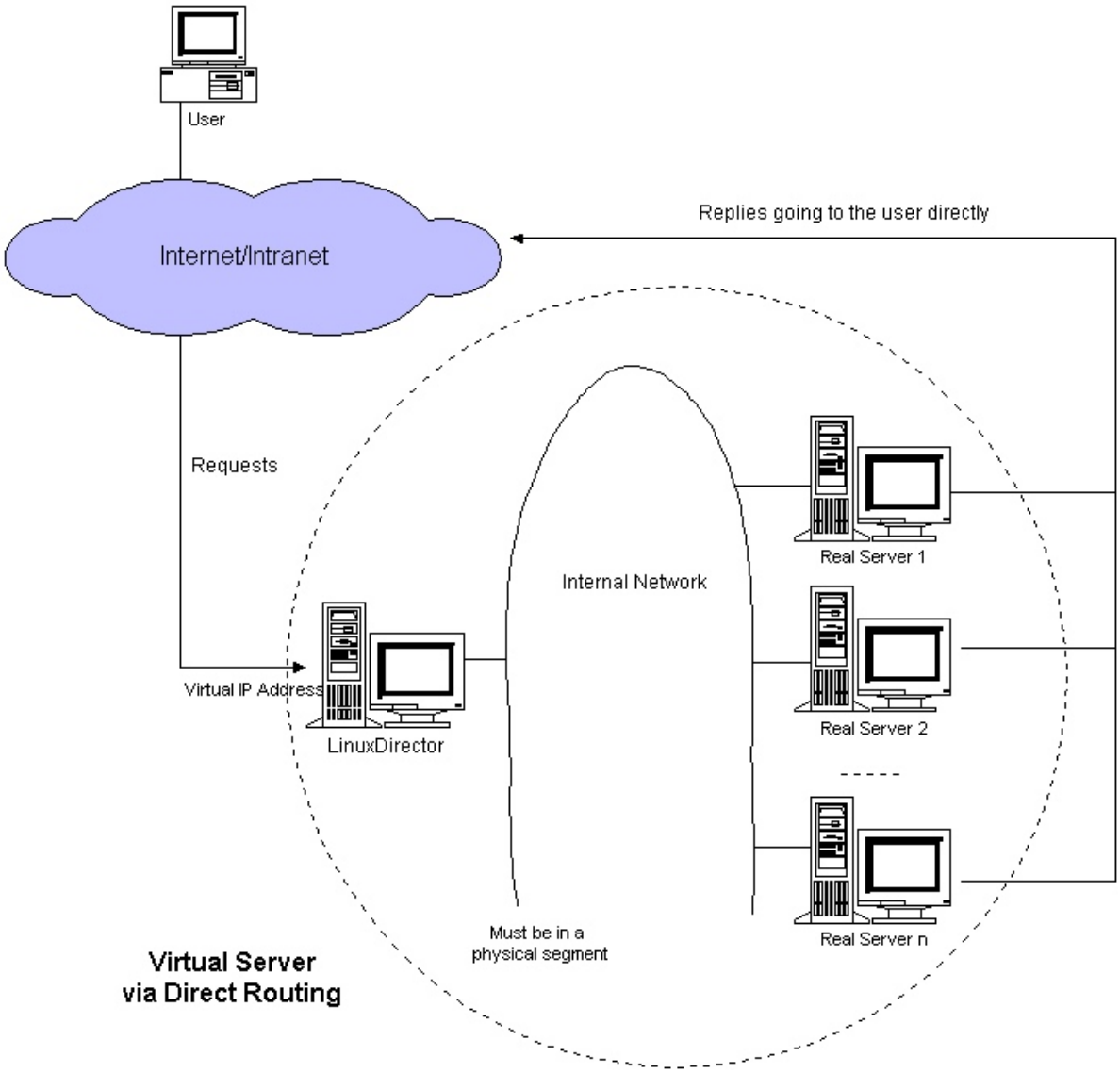
```
# 真实情况下，客户端节点也是无法连接企业内部网络
[root@client ~]# curl -HHost:lvs.oldxu.com http://172.16.1.100
curl: (7) Failed to connect to 172.16.1.100: 网络不可达

# 需要通过访问路由的公网IP
[root@client ~]# curl -HHost:lvs.oldxu.com http://10.0.0.200
Web Page RS-Node2
[root@client ~]# curl -HHost:lvs.oldxu.com http://10.0.0.200
Web Page RS-Node1
```

03 LVS DR 模型实践

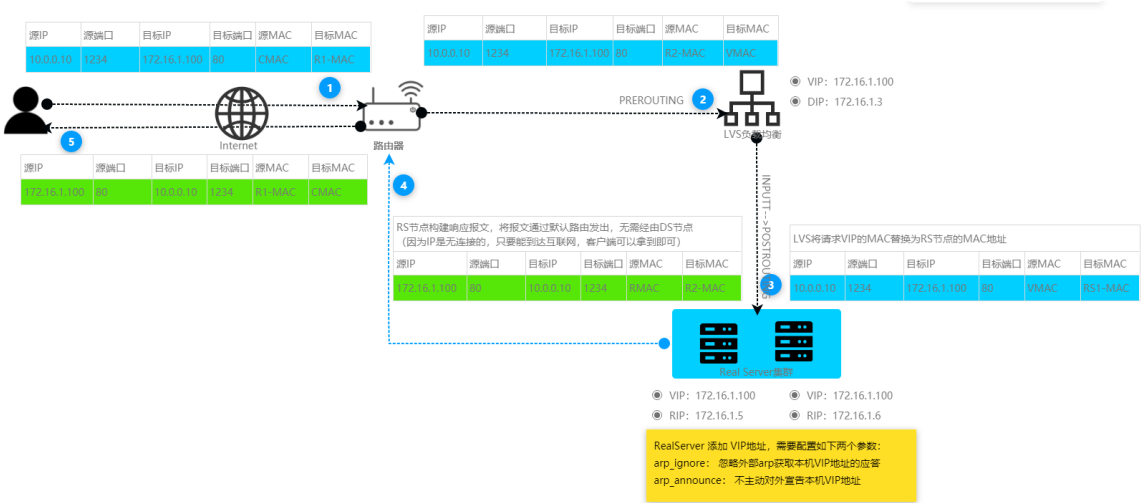
1.LVS DR模型原理

通过修改请求报文的目标`MAC`地址，然后根据算法挑选出合适的 `RS` 节点，进行转发。（请求进入`DS Server` 时做 `MAC` 地址替换，后端返回数据报文时无需经过`DS Server`节点，直接返回给客户端即可。）

1.1 DR模型基础图解



1.2 DR模型底层实现



- 1.路由器如何找到 `VIP` 以及 `MAC` 地址呢?
  - 路由器通过 `ARP` 广播获取 `VMAC`，然后封装 `CIP、VIP、CMAC、VMAC`，通过交换机转发至目标主机
- 2. `RS` 处理请求直接返回给 `CIP`，不经过 `DS`，那么 `RS` 如何将数据包回传给 `CIP`

    - 由于CIP请求的是VIP，而响应是通过RIP响应给CIP，所以数据报文一定会被丢弃。那么就需要在所有的RS的接口上配置VIP的地址。由RS上的VIP响应给CIP即可。
- 3.所有RS节点都配置VIP，那么路由器在广播的时候，岂不是所有的VIP都会响应?
    - 方式1：在路由器上静态绑定VIP与VMAC的关系。（但可能没有操作权限）
    - 方式2：在所有RS节点上配置ARP抑制，简单来说就是路由器广播获取VMAC时，所有的RS都不应答，其次所有的RS都不对外宣布自己的VIP
- 5.VIP、DIP、RIP需要在同一网段中吗?
    - 一般来说DIP和RIP在同一物理网络中，并不一定在同一网段中。

1.3.DR模型访问流程

- 1、当用户请求到达DS节点，此时请求的数据报文会先到内核空间的PREROUTING链。 此时报文的源IP为CIP， 目标IP为VIP。
- 2、 PREROUTING检查发现数据包的目标IP是本机，将数据包送至INPUT链。
- 3、 IPVS比对数据包请求的服务是否为集群服务，是则将请求报文中的源MAC修改为DMAC，将目标MAC修改RMAC，然后将数据包通过POSTROUTING链发出。此时的源IP和目的IP均未修改，仅将源MAC修改为DMAC， 目标MAC修改为RMAC
- 4、由于DS和RS在同一个网络中，所以是通过二层来传输。 POSTROUTING链检查目标MAC为RIP的MAC地址，那么此时数据包将通过DIP发送RS节点
- 5、 RS拆解数据报文发现请求的IP地址是本机，则会接收该数据报文，而后构建响应报文向外发出，此时的源IP是VIP， 目标IP是CIP
- 6、响应报文最终送达至客户端

1.4.DR模型特性

- 1、请求报文必须由DS节点转发，但响应报文必须不经过DS节点
- 2、 RS不能将网关指向DS节点的DIP
- 3、 DS和RS节点必须位于同一物理网络中
- 4、 DR模型不支持地址转换，也不支持端口映射
- 5、 RS可以是常见的操作系统Windows、Linux、MacOS
- 6、 RS在lo接口上配置VIP eth1（arp）

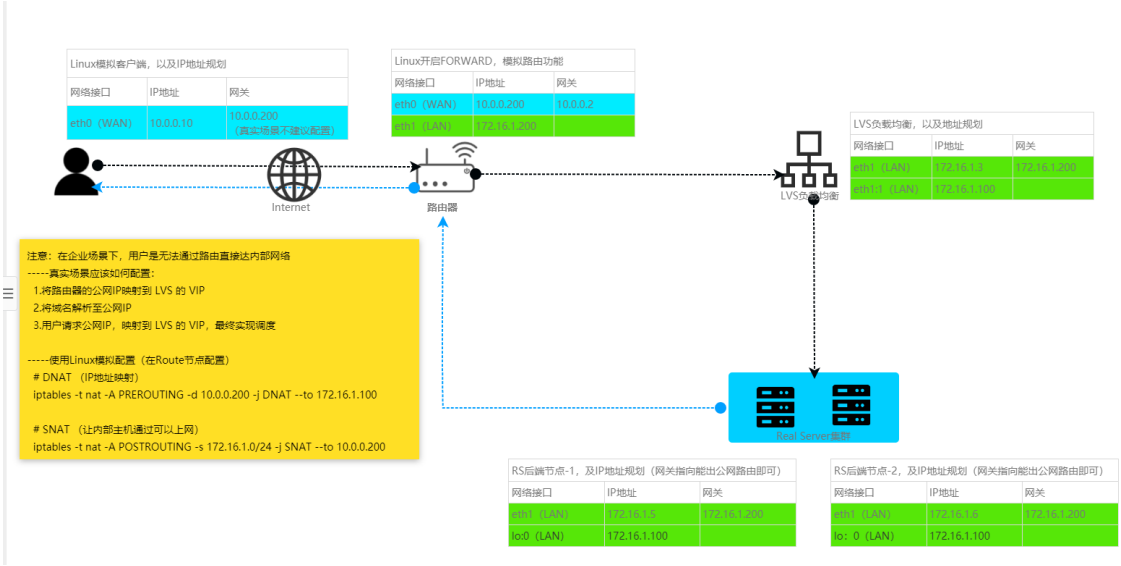
1.5.DR模型ARP

- arp_ignore（控制系统在收到外部的arp请求时，是否需要应答。）
    - 0 默认值，将本机所有接口的所有信息像每个连接的网络进行通告。
    - 1 只应答本地主机访问网络接口（eth0-->lo），才给予响应
- arp_announce（控制系统是否对外宣布自己的地址。）

- 0 默认值，把本机所有接口的所有信息向每个接口的网络进行通告；
- 1 "尽量避免" 将接口信息向非直接连接网络进行通告；
- 2 "必须避免" 将接口信息向非本网络进行通告；

2.LVS DR模型实践

2.1.DR架构规划



2.2.DR 路由器配置

1.将 `Linux` 服务器配置为路由器，先配置其 `IP` 地址

- `eth0` 配置信息如下

```
[root@route ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.0.200
PREFIX=24
GATEWAY=10.0.0.2          # 指向能出公网的IP
DNS1=223.5.5.5
```

- `eth1` 配置信息如下

```
[root@route ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=172.16.1.200
PREFIX=24
```

2.在 `Route` 节点启用 `FORWARD` 转发功能，实现路由功能。

```
[root@route ~]# echo "net.ipv4.ip_forward = 1" >>
/etc/sysctl.conf
[root@route ~]# sysctl -p
```

2.4.DR RS配置

1.配置 `RS` 节点 `eth1` 网卡为 `LAN` 模式，然后将网关指向能出公网的路由即可。（所有 `RS` 节点都需要操作）

```
[root@rs01 ~]# ifdown eth0      # 关闭 eth0网关，真实生产环境也仅有一块网卡
[root@rs01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=172.16.1.5         # 不同的 RS 节点地址不一样
GATEWAY=172.16.1.200      # 所有的 RS 节点都需要能出公网网关
PREFIX=24
DNS=223.5.5.5
```

2.配置 `RS` 节点 `VIP` 地址

```
[root@rs01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-lo:0
DEVICE=lo:0
IPADDR=172.16.1.3
NETMASK=255.0.0.0
ONBOOT=yes
NAME=loopback
```

3.重启 `eth1`、`lo:0` 网络，使其生效。

```
[root@rs01 ~]# ifdown eth1 && ifup eth1
[root@rs01 ~]# ifdown lo:0 && ifup lo:0
```

4.配置 `arp` ，不对外宣告本机 `VIP` 地址，也不响应其他节点发起 `ARP` 请求 本机的 `VIP`

```
# echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
# echo "1" >/proc/sys/net/ipv4/conf/default/arp_ignore
# echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore

# echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
# echo "2" >/proc/sys/net/ipv4/conf/default/arp_announce
# echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
```

5.检查 `RS` 节点路由信息

```
[root@rs01 ~]# route -n
Destination     Gateway         Genmask         Flags Metric Ref
   Use Iface
0.0.0.0         172.16.1.200    0.0.0.0         UG    100    0
     0 eth1
172.16.1.0      0.0.0.0         255.255.255.0   U     100    0
     0 eth1
```

4.配置后端所有 `RS` 的 `web` 服务，注意 `RS1` 和 `RS2` 页面不一样，方便验证效果。

- `rs1` 节点配置

```
[root@rs01 ~]# yum install nginx -y
[root@rs01 ~]# cat /etc/nginx/conf.d/lvs.oldxu.com.conf
server {
    listen  80;
    server_name lvs.oldxu.com;
    root /opt;

    location / {
        index index.html;
    }
}
[root@rs01 ~]# echo "Web Page RS-Node1" > /opt/index.html
[root@rs01 ~]# systemctl start nginx

# 本机测试访问
```

```
[root@rs01 ~]# curl -HHost:lvs.oldxu.com http://172.16.1.5
Web Page RS-Node1
```

- `rs2` 节点配置

```
[root@rs02 ~]# yum install nginx -y
[root@rs02 ~]# cat /etc/nginx/conf.d/lvs.oldxu.com.conf
server {
    listen  80;
    server_name lvs.oldxu.com;
    root /opt;

    location / {
        index index.html;
    }
}
[root@rs02 ~]# echo "Web Page RS-Node2" > /opt/index.html
[root@rs02 ~]# systemctl start nginx

# 本机测试访问
[root@rs02 ~]# curl -HHost:lvs.oldxu.com http://172.16.1.6
Web Page RS-Node2
```

2.5.DR DS配置

1.编辑网卡配置，将 `DS` 节点的 `eth1` 网关指向路由节点。

```
[root@lvs-01 ~]# ifdown eth0
[root@lvs-01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=172.16.1.3
PREFIX=24
GATEWAY=172.16.1.200    # 填写一个网关即可
DNS1=223.5.5.5
```

2.新增 `VIP` 地址的网卡配置，将 `VIP` 绑定到 `eth1:1` 网卡上。

```
[root@lb01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1:1
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth1:1
DEVICE=eth1:1
ONBOOT=yes
IPADDR=172.16.1.100
PREFIX=24
```

3.重启 `DS` 节点 `eth1、eth1:1` 的网卡

```
[root@lvs-01 ~]# ifdown eth1 && ifup eth1
[root@lvs-01 ~]# ifdown eth1:1 && ifup eth1:1
```

4.开启 `DS` 节点的内核转发功能。（`DR` 模型不需要，因为不需要 `DS` 节点转发 `RS` 请求）

```
[root@lvs-01 ~]# echo "net.ipv4.ip_forward = 1" >>
/etc/sysctl.conf
[root@lvs-01 ~]# sysctl -p
```

5.`DS` 节点负载均衡配置

```
# 定义 LVS 集群
[root@lvs-01 ~]# ipvsadm -C
[root@lvs-01 ~]# ipvsadm -A -t 172.16.1.100:80 -s rr

# 添加 RS1、RS2集群节点，采用官网模式
[root@lvs-01 ~]# ipvsadm -a -t 172.16.1.100:80 -r 172.16.1.5:80 -
g
[root@lvs-01 ~]# ipvsadm -a -t 172.16.1.100:80 -r 172.16.1.6:80 -
g

# 查看集群状态信息
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 rr
  -> 172.16.1.5:80                Route   1      0          0
  -> 172.16.1.6:80                Route   1      0          0
```

2.5 DR Client测试

1.配置 `Client` 节点 `eth0` 网络。

```
[root@client ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.0.100
PREFIX=24
GATEWAY=10.0.0.200        # 真实场景不可能将客户端网关指向企业的路由器上
```

2.重启 `Client` 节点网络

```
[root@client ~]# systemctl restart network
```

3.使用 `Client` 测试访问效果

```
[root@client ~]# curl -HHost:lvs.oldxu.com http://172.16.1.100
Web Page RS-Node1
[root@client ~]# curl -HHost:lvs.oldxu.com http://172.16.1.100
Web Page RS-Node2
```

4.模拟真实场景，首先删除 `Client` 节点指向 `Route` 的网关信息，然后配置 `Route` 打开 `IP` 映射DNAT、以及SNAT（共享上网功能）。

- 删除 `Client` 节点的网关配置

```
[root@client ~]# sed -i '/GATEWAY/d' /etc/sysconfig/network-scripts/ifcfg-eth0
[root@client ~]# systemctl restart network
```

- 配置 `Route` 路由节点的 `DNAT` 以及 `SNAT`

```
# DNAT （地址映射，企业环境使用）
[root@route ~]# iptables -t nat -A PREROUTING -d 10.0.0.200 -j
DNAT --to 172.16.1.100
# DNAT （端口映射，虚拟环境使用）
[root@route ~]# iptables -t nat -A PREROUTING -d 10.0.0.200 -p
tcp --dport 80 -j DNAT --to 172.16.1.100:80

#SNAT （让内部主机通过路由可以上网）
[root@route ~]# iptables -t nat -A POSTROUTING -s 172.16.1.0/24 -
j SNAT --to 10.0.0.200
```

5.最后用 `Client` 再次测试

```
# 真实情况下，客户端节点也是无法连接企业内部网络
[root@client ~]# curl -HHost:lvs.oldxu.com http://172.16.1.100
curl: (7) Failed to connect to 172.16.1.100: 网络不可达

# 需要通过访问路由的公网IP
[root@client ~]# curl -HHost:lvs.oldxu.com http://10.0.0.200
Web Page RS-Node2
[root@client ~]# curl -HHost:lvs.oldxu.com http://10.0.0.200
Web Page RS-Node1
```

3.LVS DR模型脚本实践

在网络规划没有问题的情况下，通过脚本来实现 `LVS DR` 模型。（注意修改脚本 中的IP地址）

3.1.DS配置脚本

```
[root@lvs-01 ~]# cat lvs_dr.sh
#!/usr/bin/bash
VIP=172.16.1.100
RS1=172.16.1.5
RS2=172.16.1.6
PORT=80
DEV=eth1:1

case $1 in
    start)
    cat  >/etc/sysconfig/network-scripts/ifcfg-${DEV} <<-EOF
    TYPE=Ethernet
    BOOTPROTO=none
    DEFROUTE=yes
```

```
    NAME=${DEV}
    DEVICE=${DEV}
    ONBOOT=yes
    IPADDR=${VIP}
    PREFIX=24
    EOF

    # 启动网卡
    ifup ${DEV}

    # 配置LVS规则
    ipvsadm -C
    ipvsadm -A -t ${VIP}:${PORT} -s rr
    ipvsadm -a -t ${VIP}:${PORT} -r ${RS1} -g
    ipvsadm -a -t ${VIP}:${PORT} -r ${RS2} -g
    ;;

    stop)
        ifdown ${DEV}
        rm -f /etc/sysconfig/network-scripts/ifcfg-${DEV}
        ipvsadm -C
       ;;
    *)
        echo "Usage: sh $0 { start | stop }"
    ;;
esac
```

3.2.RS配置脚本

```
[root@web01 ~]# cat lvs_rs.sh
#!/usr/bin/bash

VIP=172.16.1.100
DEV=lo:0

case $1 in
    start)
    echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
    echo "1" >/proc/sys/net/ipv4/conf/default/arp_ignore
    echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore

    echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
    echo "2" >/proc/sys/net/ipv4/conf/default/arp_announce
    echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
```

```
    cat  >/etc/sysconfig/network-scripts/ifcfg-${DEV} <<-EOF
    DEVICE=lo:0
    IPADDR=${VIP}
    NETMASK=255.0.0.0
    ONBOOT=yes
    NAME=loopback
    EOF

    ifup ${DEV} # 启动网卡
    systemctl start nginx
    ;;
    stop)
    echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
    echo "0" >/proc/sys/net/ipv4/conf/default/arp_ignore
    echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore

    echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
    echo "0" >/proc/sys/net/ipv4/conf/default/arp_announce
    echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce

        ifdown ${DEV}  # 停止网卡
        rm -f /etc/sysconfig/network-scripts/ifcfg-${DEV}
        systemctl stop nginx
        ;;
    *)
        echo "Usage: sh $0 { start | stop }"
esac
```

4.LVS 持久连接实践

4.1 什么是持久化连接

`lvs persistence 持久连接`，无论使用任何调度算法，在一段时间内（默认300s），能够实现将来自同一个地址的请求始终发往同一个`RS`

4.2.持久化连接配置

```
[root@lb01 ~]# ipvsadm -E -t 172.16.1.100:80 -p 30
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 wlc persistent 30
  -> 172.16.1.7:80                Route   1      0          0
  -> 172.16.1.8:80                Route   1      0          0
```

4.3.持久化连接测试

客户端请求后，可以登陆LVS节点查看，是否有一个连接处于长连接状态。（需要每个请求都TIME-WAIT结束，才会断开长连接）

```
[root@lb01 ~]# ipvsadm -L -n -c
IPVS connection entries
pro expire state       source             virtual
destination
TCP 01:52  FIN_WAIT    10.0.0.100:60100   172.16.1.100:80
 172.16.1.6:80
# 这个就是长连接的计数器，为0就断开了长连接
TCP 00:22  NONE        10.0.0.100:0       172.16.1.100:80
 172.16.1.6:80
```

5.LVS DR模型高可用

`LVS` 可以实现负载均衡功能，但是没有健康检查机制，如果一台 `RS` 节点故障，`LVS` 任然会将请求调度至该故障 `RS` 节点服务器；

- 那么我们可以使用 `Keepalived` 来实现解决：
  - 1.使用 `Keepalived` 可以实现 `LVS` 的健康检查机制，`RS` 节点故障，则自动剔除该故障的 `RS` 节点，如果 `RS` 节点恢复则自动加入集群。
  - 2.使用 `Keeplaived` 可以解决 `LVS` 单点故障，以此实现 `LVS` 的高可用。
  - 3.可以理解 `Keepalived` 就是为 `LVS` 而诞生的。

5.1.Master节点配置

1.安装 `Keepalived` 软件

```
[root@lvs-01 ~]# yum install keepalived -y
```

2.配置 `Keepalived` 为 `MASTER` 角色

```
[root@lb01 ~]# cat /etc/keepalived/keepalived.conf
global_defs {
    router_id lb01
}

vrrp_instance VI_1 {
    state MASTER
    priority 200

    interface eth1
    virtual_router_id 50
    advert_int 3
    authentication {
        auth_type PASS
        auth_pass 1111
 }
    virtual_ipaddress {
        172.16.1.100
    }
}

# 配置集群地址访问的IP+Port
virtual_server 172.16.1.100 80 {
    # 健康检查的时间，单位：秒
    delay_loop 6
    # 配置负载均衡的算法
    lb_algo wlc
    # 设置LVS的模式 NAT|TUN|DR
    lb_kind DR
    # 设置会话持久化的时间
    perssisstence_timeout 5
    # 设置协议
    protocol TCP

    # 负载均衡后端的真实服务节点RS-1
    real_server 172.16.1.5 80 {
        # 权重配比设置为1
        weight 1
        # 设置健康检查
        TCP_CHECK {
            # 检测后端80端口
            connect_port 80
            # 超时时间
            connect_timeout 3
            # 重试次数2次
```

```
            nb_get_retry 2
            # 间隔时间3s
            delay_beefore_retry 3
        }
    }
     # 负载均衡后端的真实服务节点RS-2
    real_server 172.16.1.6 80 {
        # 权重配比设置为1
        weight 1
        # 设置健康检查
        TCP_CHECK {
            # 检测后端80端口
            connect_port 80
            # 超时时间
            connect_timeout  3
            # 重试次数2次
            nb_get_retry 2
            # 间隔时间3s
            delay_beefore_retry 3
        }
    }
}
```

5.2.Backup节点配置

1.安装 `Keepalived` 软件

```
[root@lvs-01 ~]# yum install keepalived -y
```

2.配置 `Keepalived` 为 `BACKUP` 角色

```
[root@lb01 ~]# cat /etc/keepalived/keepalived.conf
global_defs {
    router_id lb01
}

vrrp_instance VI_1 {
    state BACKUP
    priority 150

    interface eth1
    virtual_router_id 50
    advert_int 1
    authentication {
        auth_type PASS
```

```
        auth_pass 1111
 }
    virtual_ipaddress {
        172.16.1.3
    }
}

# 配置集群地址访问的IP+Port
virtual_server 172.16.1.3 80 {
    # 健康检查的时间，单位：秒
    delay_loop 6
    # 配置负载均衡的算法
    lb_algo wlc
    # 设置LVS的模式 NAT|TUN|DR
    lb_kind DR
    # 设置会话持久化的时间
    perssisstence_timeout 5
    # 设置协议
    protocol TCP

    # 负载均衡后端的真实服务节点RS-1
    real_server 172.16.1.7 80 {
        # 权重配比设置为1
        weight 1
        # 设置健康检查
        TCP_CHECK {
            # 检测后端80端口
            connect_port 80
            # 超时时间
            connect_timeout  3
            # 重试次数2次
            nb_get_retry 2
            # 间隔时间3s
            delay_beefore_retry 3
        }
    }
     # 负载均衡后端的真实服务节点RS-2
    real_server 172.16.1.7 80 {
        # 权重配比设置为1
        weight 1
        # 设置健康检查
        TCP_CHECK {
            # 检测后端80端口
            connect_port 80
            # 超时时间
```

```
            connect_timeout  3
            # 重试次数2次
            nb_get_retry 2
            # 间隔时间3s
            delay_beefore_retry 3
        }
    }
}
```

5.3 高可用架构测试

当启动两台 `LVS` 节点的 `Keepalived` 软件后，会发现两台节点都有 `LVS` 的规则，但仅标记为 `Master` 节点的服务器有 `VIP` 地址；
当有新的请求进入时，所有的请求都会发送至 `Master` 节点，当 `Master` 节点故障，`VIP` 地址会漂移到 `Backup` 节点，由 `Backup` 节点继续对外提供服务，以此来实现 `LVS` 的高可用。

- `LVS-MASTER` 节点地址检查，`LVS` 规则检查。

```
[root@lb01 ~]# ip addr | grep 172.16.1.3
    inet 172.16.1.3/32 scope global eth1

[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.3:80 wlc
  -> 172.16.1.7:80                Route   1      0          0
  -> 172.16.1.8:80                Route   1      0          0
```

- `LVS-BACKUP` 节点地址检查，`LVS` 规则检查。

```
[root@lb02 ~]# ip addr |grep 172.16.1.3
[root@lb02 ~]#

[root@lb02 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.3:80 wlc
  -> 172.16.1.7:80                Route   1      0          0
  -> 172.16.1.8:80                Route   1      0          0
```

- 假设目前有一台 `Real Server` 故障，`Keeplaived` 检测后会自动将节点移除

```
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.3:80 wlc
  -> 172.16.1.8:80                Route   1      0          0


# 观察Keeplaived的详情
[root@lb01 ~]# systemctl status keepalived
...
1月 08 10:11:55 lb01 Keepalived_healthcheckers[25006]: TCP
connection to [172.16.1.7]:80 failed.
1月 08 10:11:56 lb01 Keepalived_healthcheckers[25006]: TCP
connection to [172.16.1.7]:80 failed.
1月 08 10:11:56 lb01 Keepalived_healthcheckers[25006]: Check on
service [172.16.1.7]:80 failed after 1 retry.
1月 08 10:11:56 lb01 Keepalived_healthcheckers[25006]: Removing
service [172.16.1.7]:80 from VS [172.16.1.3]:80
```

04 LVS调度算法详解

1.LVS调度算法概述

`LVS` 根据后端服务器的负载，或其他的计算的标准，判断挑选哪台 `RS` 来进行请求处理。调度算法主要分为”静态调度算法“、”动态调度算法“。

- 静态调度算法：`RR、WRR、SH、DH`
- 动态调度算法：`LC、WLC、SED、NQ、LBLC、LBLCR`

2.LVS静态调度算法

静态：仅根据算法本身进行调度，不考虑后端实际负载情况（起点公平）

2.1.RR调度算法

`RR: round robin` 轮询调度算法，将每一次用户的请求，轮流分配给`Real Server` 节点。

```
[root@lb01 ~]# ipvsadm -E -t 172.16.1.100:80 -s rr

[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 rr
  -> 172.16.1.7:80                Route   1      0          0
  -> 172.16.1.8:80                Route   1      0          0
```

2.2.WRR调度算法

`WRR: weighted round robin` 加权轮询调度算法，根据服务器的硬件情况、以及处理能力，为每台服务器分配不同的权值，使其能够接受相应权值的请求。

```
[root@lb01 ~]# ipvsadm -E -t 172.16.1.100:80 -s wrr
[root@lb01 ~]# ipvsadm -e -t 172.16.1.100:80 -r 172.16.1.5:80 -g
-w 5
[root@lb01 ~]# ipvsadm -e -t 172.16.1.100:80 -r 172.16.1.6:80 -g
-w 1
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 wrr
  -> 172.16.1.7:80                Route   5      0          0
  -> 172.16.1.8:80                Route   1      0          0
```

2.3.SH调度算法

`SH: Source Hashing` 源地址`hash`调度算法，将请求的源`IP`地址进行`Hash`运算，得到一个具体的数值，同时对后端服务器进行编号，按照运算结果将请求分发到对应编号的服务器上。

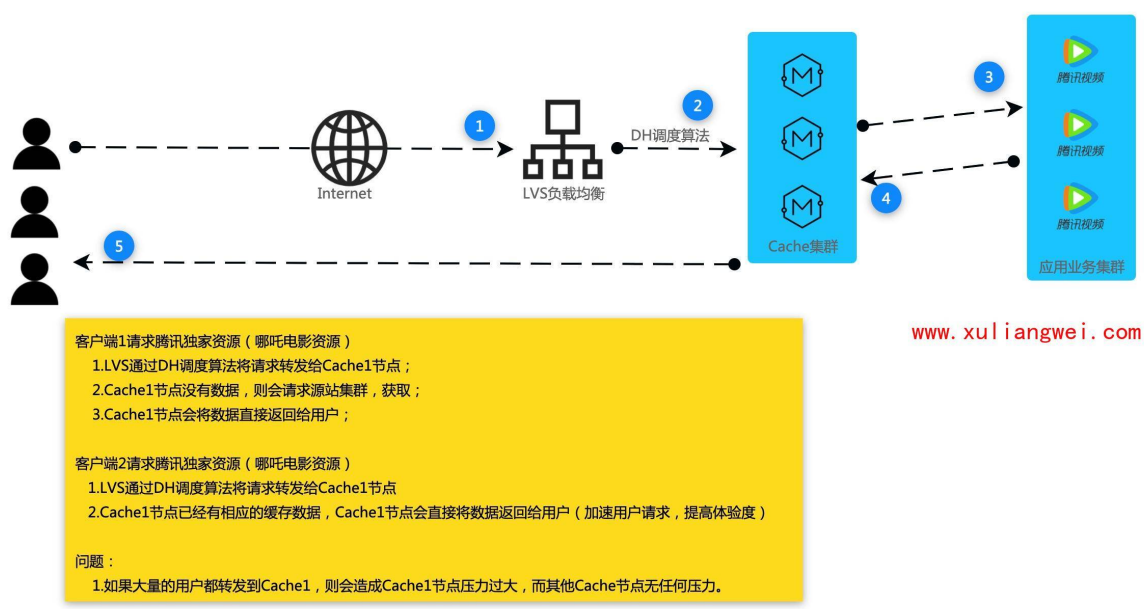
1.可以实现不同来源 `IP` 的请求进行负载分发；

2.同时还能实现相同来源 `IP` 的请求始终被派发至某一台特定的节点；

```
[root@lb01 ~]# ipvsadm -E -t 172.16.1.100:80 -s sh
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 sh    # 配置了Weight无效
  -> 172.16.1.7:80                Route   5      0          3
  -> 172.16.1.8:80                Route   1      0          1
```

2.4.DH调度算法

`DH: destination hash` 目标地址`hash`将客户端的请求，始终发往同一个`RS`。
应用场景：`LVS-cache-源站`，始终调度到指定的`cache`，加速用户体验。




```
[root@lb01 ~]# ipvsadm -E -t 172.16.1.100:80 -s dh
[root@lb01 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  172.16.1.100:80 dh
  -> 172.16.1.7:80                Route   5      0          0
  -> 172.16.1.8:80                Route   1      0          0
```

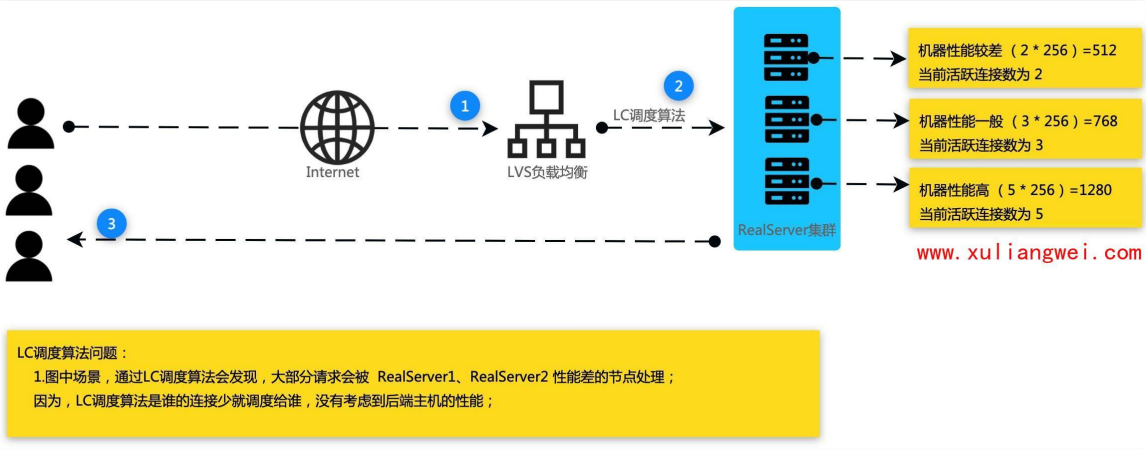
3.LVS动态调度算法

动态：根据算法及`RS`节点负载状态进行调度，较小的`RS`将被调度（保证结果公平）

3.1.LC调度算法

`LC：Least-Connection` 最少连接数调度算法，哪台`RS`连接数少就将请求调度至哪台`RS`。

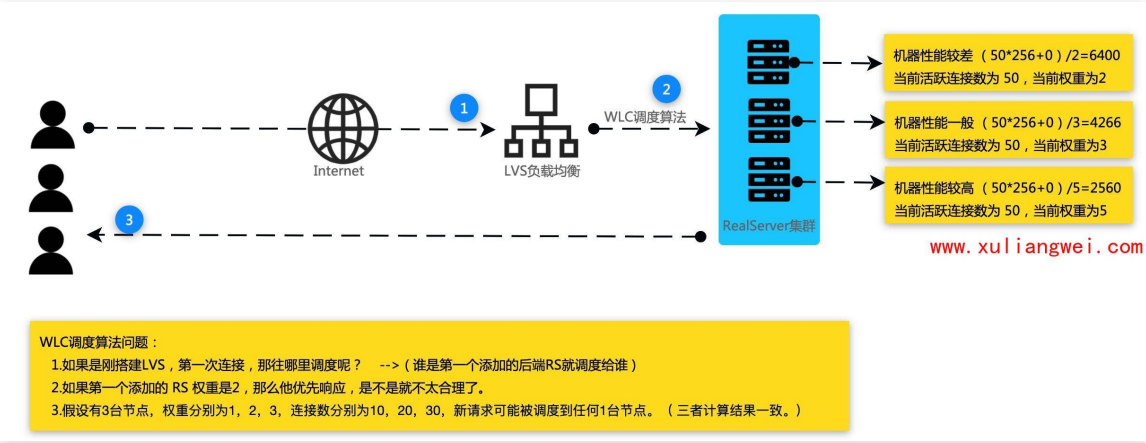
算法：`Overhead = ( Active * 256 + Inactive仅连接 )`一个活动连接相当于256个非活动连接



3.2.WLC调度算法

`WLC：Weighted Least-Connection` 加权最小连接数（默认调度算法），在服务器性能差异较大的情况下，采用"加权最少链接"调度算法优化负载均衡性能，权值较高的`RS`节点，将承受更多的连接；负载均衡可以自动问询`RS`节点服务器的负载状态，通过算法计算当前连接数最少的节点，而后将新的请求调度至该节点。

算法：`Overhead =（ Active * 256 + Inactive ）/Weight`



3.3.SED调度算法

`SED：Shortest Expected Delay` 最短期望延迟，尽可能让权重高的优先接收请求，不再考虑非活动状态，把当前处于活动状态的数目+1，通过算法计算当前连接数最少的节点，而后将新的请求调度至该节点。

算法：在WLC基础上改进，`Overhead = （ACTIVE+1）*256/Weight`

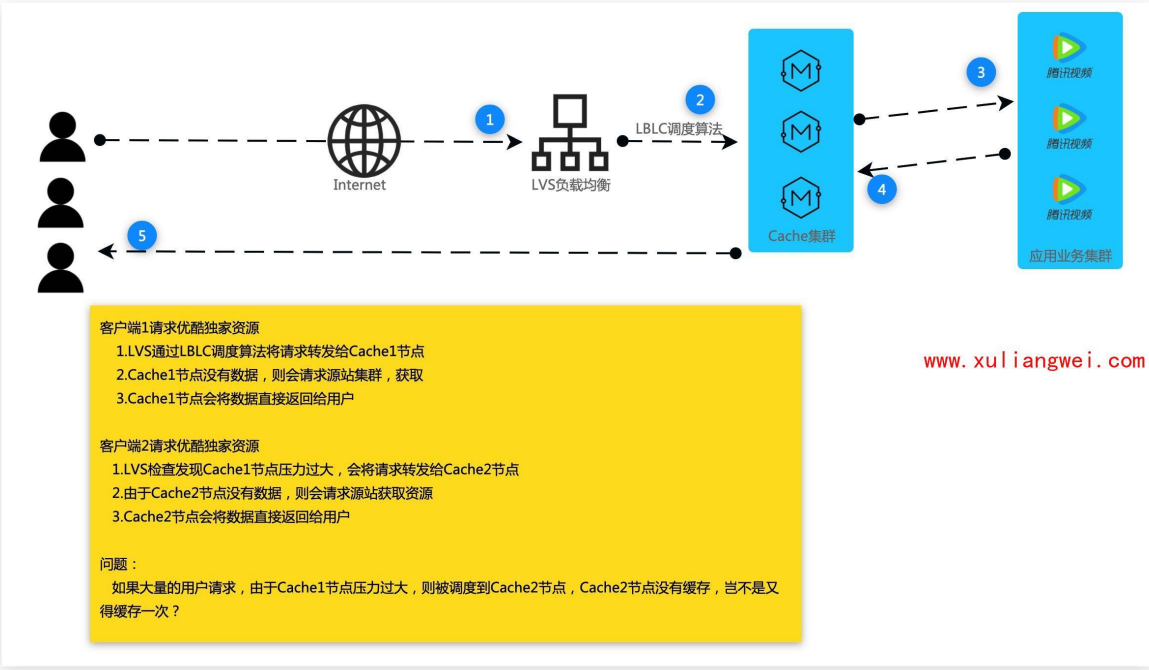
3.4.NQ调度算法

`NQ：Never Queue` 永不排队/最少队列调度

- 原理：`SED`算法由于某台服务器的权重较小，比较空闲，甚至接收不到请求，而权重大的服务器会很忙，而`NQ`算法是说不管权重多大都会被分配到请求。简单来说，就是无需队列，如果有台`Real Server`的连接数为0会直接分配过去，后续采用 `SED`算法
- 算法：`Overhead = （ACTIVE+1）*256/Weight`

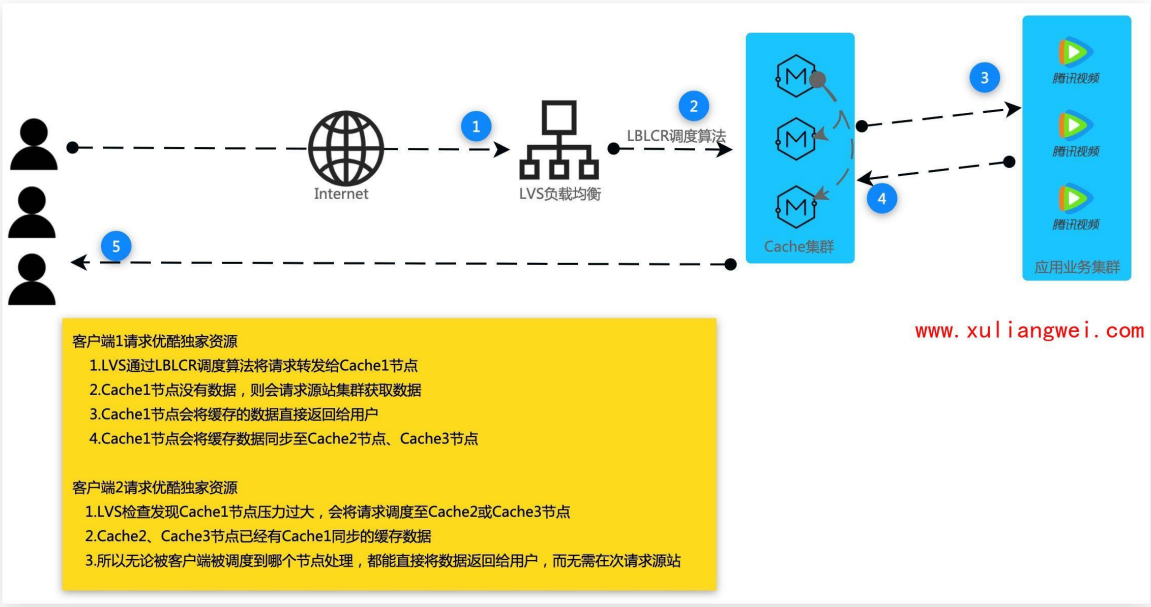
3.5.LBLC调度算法

`LBLC：Locality-Based Least-Connection` 动态目标地址`hash`调度算法，解决`DH`调度算法负载不均衡。
应用场景：`LVS-cache-源站`，此前`DH`算法始终调度到后端`Cache1`节点，会造成`Cache1`负载过高，`LBLC`会根据负载均衡动态调度到后端其他`Cache`节点。



3.6.LBLCR调度算法

`LBLCR： Locality-Based Least-Connection with Replication` 带复制功能的LBLC算法，解决LBLC负载不均衡的问题，从负载重的复制到负载轻的RS。
应用场景：`LVS-cache-源站`，此前`LBLC`算法始终调度到后端`Cache1`节点，会造成`Cache1`负载过高，会根据负载均衡动态调度到后端其他`Cache`节点，同时也会将缓存数据同步一份至`Cache1、Cache2`节点。



05.高可用Keepalived

1.Keepalived高可用基本概述

1.什么是高可用

简单理解：两台机器启动着相同的业务系统,当有一台机器宕机，另外一台服务器能快速的接管，对于访问的用户是无感知的。

专业理解：高可用是分布式系统架构设计中必要的一环，主要目的: 减少系统不能提供服务的时间。假设系统一直能够提供服务，我们说系统的可用性是100%。如果系统每运行100个时间单位，会有1个时间单位无法提供服务，我们说系统的可用性是99%。[高可用参考URL]

2.高可用通常使用什么软件?

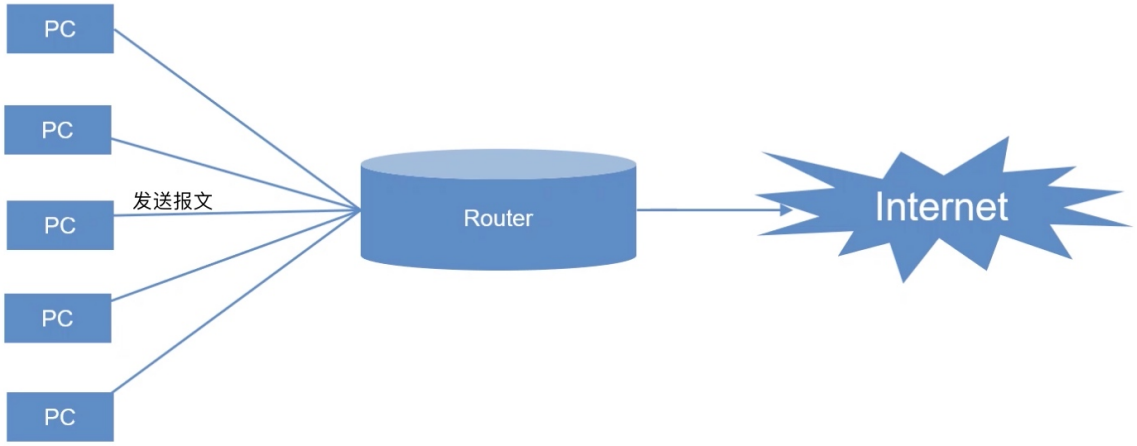
通常服务高可用我们选择使用keepalived软件实现

3.keepalived是如何实现高可用的?

keepalived软件是基于VRRP协议实现的。VRRP虚拟路由冗余协议，主要用于解决单点故障问题。

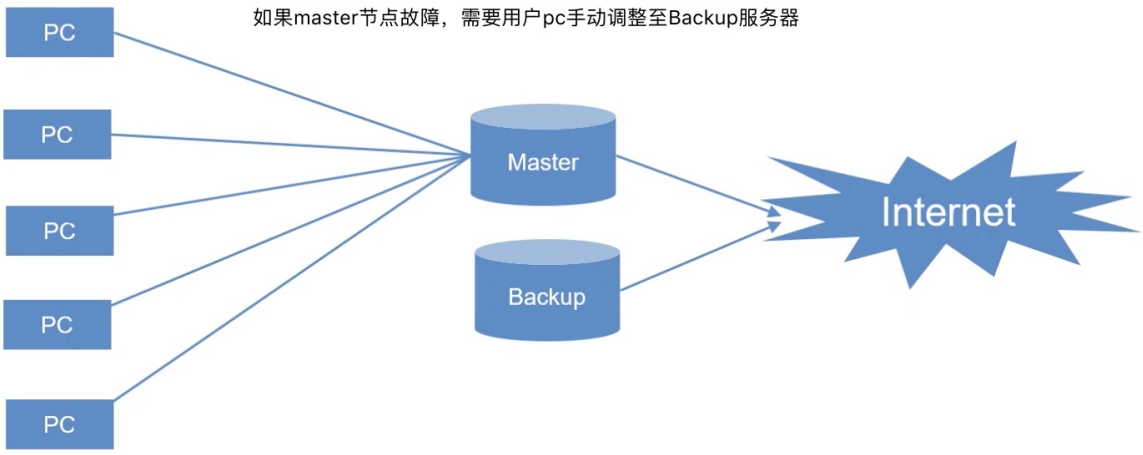
4.那VRRP是如何诞生的，VRRP的原理又是什么？

比如公司的网络是通过网关转换进行上网的，那如果该路由器故障了，网关无法转发报文了，此时所有人都将无法上网，这么时候怎么办呢？

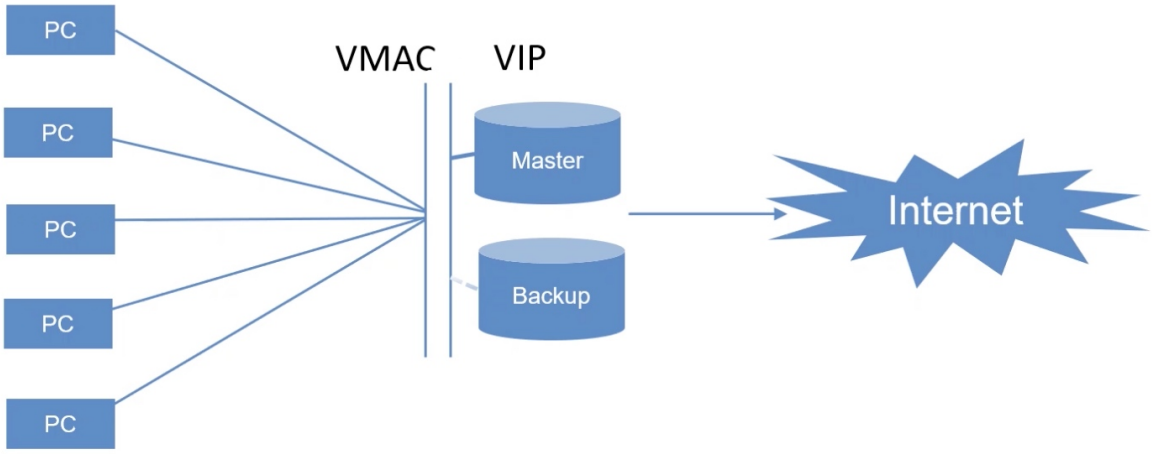


通常做法是增加一个Backup路由，然后修改用户PC电脑网关指向为Backup。
但这里有几个问题?
F1: 如果用户过多修改起来是不是会非常的麻烦?
F2: 如果用户将指向都修改为Backup，那Master如果恢复了该怎么办?

F1: 这里有人会说了，我们直接将Backup网关IP配置为Master网关IP不就可以了吗?
Q1: 这种做法不行的，为什么?
因为PC第一次是通过ARP广播寻找到Master网关的Mac地址与IP地址，PC则会将Master网关的对应IP与MAC地址写入ARP缓存表中，那么PC第二次则会直接读取ARP缓存表中的MAC地址与IP地址，然后进行数据包的转发。此时PC转发的数据包还是会发给Master。(除非PC的ARP缓存表过期，在次发起ARP广播的时候才能正确获取Bakcup的Mac地址与对应的IP地址。)

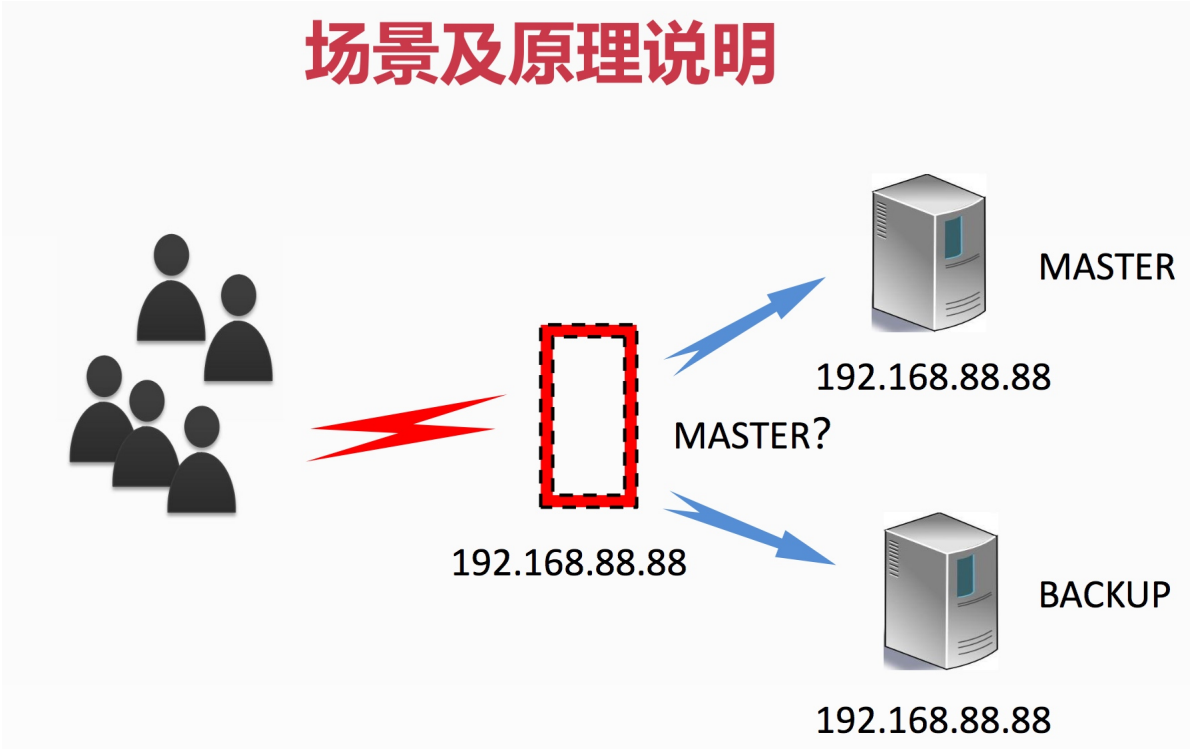


那如何才能做到出现故障自动转移，此时VRRP就应运而生，我们的VRRP其实是通过软件或硬件的形式在Master和Backup外面增加一个虚拟MAC地址(简称VMAC)与虚拟IP地址(简称VIP)。那么在这种情况下，PC请求VIP的时候，无论是Master处理还是Backup处理，PC仅会在ARP缓存表中记录VMAC与VIP的对应关系。



5.高可用keepalived使用场景

通常业务系统需要保证7x24小时不DOWN机，比如公司内部OA系统，每天公司人员都需要使用，则不允许Down机。作为业务系统来说随时随地都要求可用。



6.高可用核心概念总结

1.如何确定谁是主节点谁是备节点。(投票选举？优先级？)

2.如果Master故障，Backup自动接管，那Master恢复后会夺权吗？(抢占式、非抢占式)

3.如果两台服务器都认为自己是Master会出现什么问题？(脑裂)

2.Keepalived高可用安装配置

1.实践环境，配置实现虚IP转移

状态	IP	角色
节点1	10.0.0.5	Master
节点2	10.0.0.6	Backup
VIP	10.0.0.100	

2.在master与backup上分别安装`keepalived`

```
[root@lb01 ~]# yum install keepalived -y
[root@lb02 ~]# yum install keepalived -y
```

3.配置节点1，Master

```
[root@lb01 ~]# cat /etc/keepalived/keepalived.conf
global_defs {
    router_id lb01
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 50
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
}
    virtual_ipaddress {
        10.0.0.100
    }
}
```

4.配置节点2，Backup

```
[root@lb02 ~]# cat /etc/keepalived/keepalived.conf
global_defs {
    router_id lb02
}

vrrp_instance VI_1 {
    state BACKUP
    interface eth0
```

```
    virtual_router_id 50
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.100
    }
}
```

5.对比`keepalived`的`master`与`backup`配置的区别

Keepalived配置区别	Master配置	Backup节配置
route_id(唯一标识)	lb01	lb02
state(角色状态)	MASTER	BACKUP
priority(优先级)	150	100

6.启动Master与Backup节点的keepalived

```
#lb01
[root@lb01 ~]# systemctl enable keepalived
[root@lb01 ~]# systemctl start keepalived

#lb02
[root@lb02 ~]# systemctl enable keepalived
[root@lb02 ~]# systemctl start keepalived
```

PS：什么是广播，什么是组播?
广播：比如讲课，你说一句话，教室所有的人能收到。
组播：比如解决问题，整个教室只有特定的人群才能收到。组播是有特定的成员，是一种可控的广播，组播成员需要加入"组播组"才能收到该组播的信息。

3.Keepalived高可用地址漂移

检查keepalived的虚拟VIP地址能否漂移

1.在Master上进行如下操作

```
# Master存在vip地址
[root@lb01 ~]# ip addr |grep 10.0.0.100
    inet 10.0.0.100/24 scope global secondary eth0

# 停止Master上的keepalived, 检测vip已不存在
[root@lb01 ~]# systemctl stop keepalived
[root@lb01 ~]# ip addr |grep 10.0.0.100
```

2.在Backup上进行如下操作

```
#发现地址已经漂移至Backup端
[root@lb02 ~]# ip addr|grep 10.0.0.100
    inet 10.0.0.100/24 scope global secondary eth0
```

3.此时重新启动Master上的Keepalived,会发现VIP被强行抢占

```
[root@lb01 ~]# systemctl start keepalived
[root@lb01 ~]# ip addr |grep 10.0.0.100
    inet 10.0.0.100/24 scope global secondary eth0
```

4.通过windows查看arp缓存表，验证地址漂移后是否会自动更新MAC地址。

4.Keepalived高可用非抢占式

通常master服务故障后backup会变成master，但是当master服务又恢复的时候，master会抢占VIP，这样就会发生两次切换对业务繁忙的网站来说并不是太友好，此时我们可以配置keepalived为非抢占式(前提两台主机的硬件配置信息一致)

配置非抢占式步骤如下

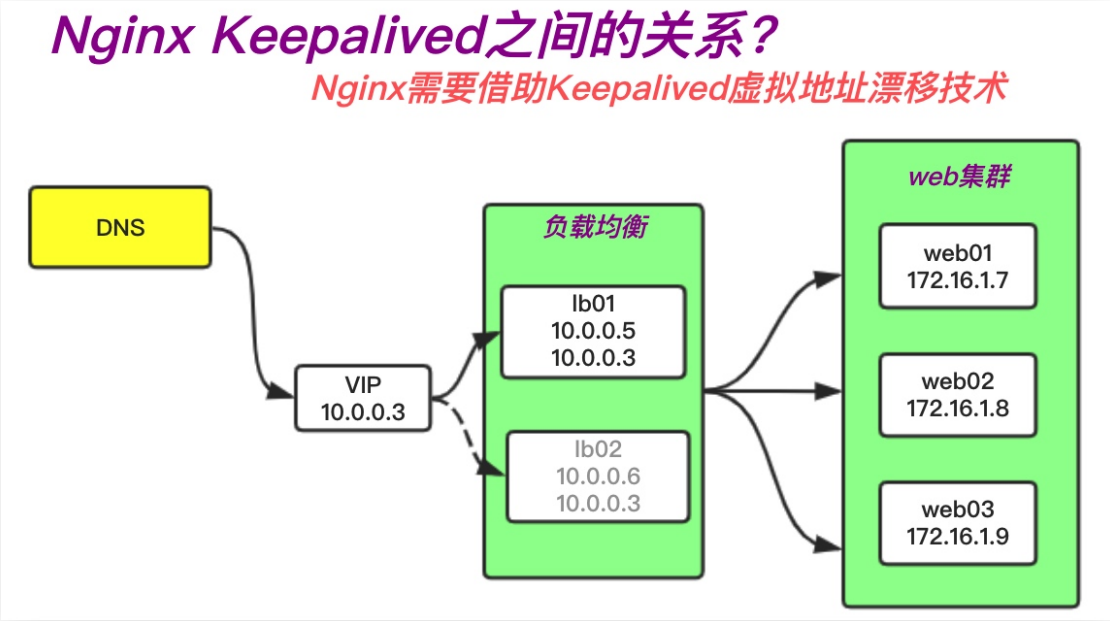
1、两个节点的state都必须配置为BACKUP(官方建议)
2、两个节点都在vrrp_instance中添加nopreempt参数
3、其中一个节点的优先级必须要高于另外一个节点的优先级。
两台服务器都角色状态启用nopreempt后，必须修改角色状态统一为BACKUP，唯一的区分就是优先级。

```
#Master
    vrrp_instance VI_1 {
        state BACKUP
        priority 150
        nopreempt
    }

#Backup
    vrrp_instance VI_1 {
        state BACKUP
        priority 100
        nopreempt
    }
```

5.Keepalived高可用与Nginx

1.Nginx与Keepalived之间是什么关系?

没关系。为什么？（Nginx仅仅是借助了Keepalived的VIP地址漂移技术，从而实现的高可用。）



2.如果Nginx如果无法访问keepalived会漂移IP地址吗？如果不能如何解决?

如果Nginx宕机，会导致用户请求失败, 但Keepalived并不会进行切换, 所以需要编写一个脚本检测Nginx的存活状态，如果不存活则 kill nginx和 keepalived

```
[root@lb01 ~]# mkdir /server/scripts
[root@lb01 ~]# vim /server/scripts/check_web.sh
#!/bin/sh
nginxpid=$(pidof nginx | wc -l)
#1.判断Nginx是否存活,如果不存活则尝试启动Nginx
```

```
if [ $nginxpid -eq 0 ];then
    systemctl start nginx
    sleep 2
    #2.等待2秒后再次获取一次Nginx状态
    nginxpid=$(pidof nginx | wc -l)
    #3.再次进行判断, 如Nginx还不存活则停止Keepalived,让地址进行漂移,并退出
脚本
    if [ $nginxpid -eq 0 ];then
        systemctl stop keepalived
   fi
fi

#给脚本增加执行权限
[root@lb01 ~]# chmod +x /server/scripts/check_web.sh
```

1.在lb01主机的keepalived配置文件中调用此脚本

```
[root@lb01 ~]# cat /etc/keepalived/keepalived.conf
global_defs {
         router_id LVS_01
}

#1.每5秒执行一次脚本, 脚本执行内容不能超过5秒,否则会被中断再次重新运行脚本
vrrp_script check_web {
   script "/server/scripts/check_web.sh"
   interval 5
}

vrrp_instance VI_1 {
    nopreempt
    state MASTER
    interface eth0
    virtual_router_id 50
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        10.0.0.100
    }

    #2.调用并运行该脚本
    track_script {
```

```
        check_web
    }
}
```