

版本 V1.1

时间 2012-06-16

修改 2012-06-17

版权 GPL

作者 itnihao

本文档来自 itnihao 对 keepalive 的学习和理解，如有谬误之处，欢迎给我发邮件交流

本文档涉及以下内容

一、keepalive 软件概述

二、vrrp 协议概述

三、keepalive 软件安装

四、keepalive 配置详解

五、keepalive+lvs 实现高可用集群

六、一些故障排除

## 一、keepalive 软件概述

什么是 Keepalived 呢，keepalived 观其名可知，保持存活，在网络里面就是**保持在线**了，也就是所谓的高可用或热备，用来防止单点故障(单点故障是指一旦某一点出现故障就会导致整个系统架构的不可用)的发生，那说到 keepalived 时不得不说的一个协议就是 VRRP 协议，可以说这个协议就是 keepalived 实现的基础，那么首先我们来看看 VRRP 协议

## 二、vrrp 协议概述

Vrrp 协议来自 h3c 官方的文档，摘抄如下，希望加深对这个协议的理解

### VRRP 技术白皮书

关键词：VRRP、虚拟路由器

摘 要：本文介绍 VRRP 的基本原理和典型应用，以及 H3C 公司 VRRP 特性解决方案的特点和组网情况。

缩略语：

缩略语	英文全名	中文解释
VRRP	Virtual Router Redundancy Protocol	虚拟路由器冗余协议
NQA	Network Quality Analyzer	网络质量分析
BFD	Bidirectional Forwarding Detection	双向转发检测
IRDP	ICMP Router Discovery Protocol	ICMP 路由发现协议
VRID	Virtual Router ID	虚拟路由器号

目 录

1 概述

1.1 产生背景

1.2 技术优点

2 VRRP 协议介绍

2.1 相关术语

2.2 虚拟路由器简介

2.3 VRRP 工作过程

欢迎交流

itnihao 的运维技术博客 <http://itnihao.blog.51cto.com>

- 2.3.1 Master 路由器的选举
- 2.3.2 Master 路由器状态的通告
- 2.3.3 认证方式
- 3 Comware 实现的技术特色
- 3.1 监视上行链路
- 3.2 Backup 监视 Master 工作状态
- 4 典型组网案例
- 4.1 主备备份
- 4.2 负载分担
- 4.3 Master 使用 BFD/NQA 监视上行链路
- 4.4 Backup 使用 BFD 监视 Master 状态
- 5 附录
- 5.1 参考文献

## 1 概述

### 1.1 产生背景

随着 Internet 的发展，人们对网络可靠性的要求越来越高。特别是对于终端用户来说，能够实时与网络其他部分保持联系是非常重要的。一般来说，主机通过设置默认网关来与外部网络联系，如图1所示：

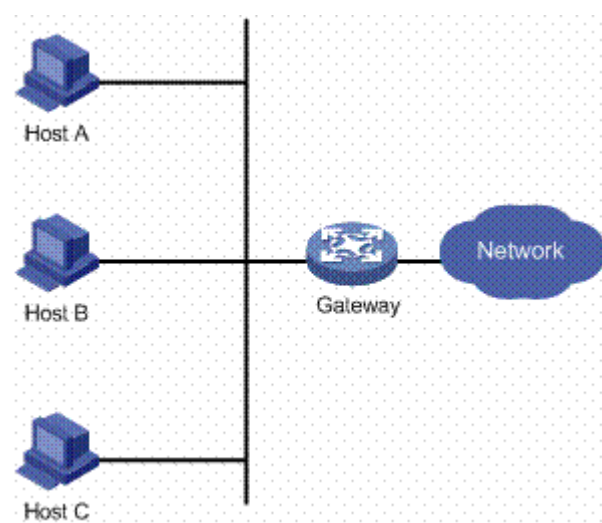


图1 常用局域网组网方案

主机将发送给外部网络的报文发送给网关，由网关传递给外部网络，从而实现主机与外部网络的通信。正常的情况下，主机可以完全信赖网关的工作，但是当网关坏掉时，主机与外部的通信就会中断。要解决网络中断的问题，可以依靠再添加网关的方式解决，不过由于大多数主机只允许配置一个默认网关，此时需要网络管理员进行手工干预网络配置，才能使得主机使用新的网关进行通信；有时，人们运用动态路由协议的方法来解决网络出现故障这一问题，如运行 RIP、OSPF 等，或者使用 IRDP。然而，这些协议由于配置过于复杂，或者安全性能不好等原因都不能满足用户的需求。

为了更好地解决网络中断的问题，网络开发者提出了 VRRP，它既不需要改变组网情况，也不需要主机上做任何配置，只需要在相关路由器上配置极少的几条命令，就能实现下一跳网关的备份，并且不会给主机带来任何负担。和其他方法比较起来，VRRP 更加能够满足用户的需求。

## 1.2 技术优点

VRRP 是一种容错协议，它保证当主机的下一跳路由器出现故障时，由另一台路由器来代替出现故障的路由器进行工作，从而保持网络通信的连续性和可靠性。

VRRP 具有如下优点：

- 简化网络管理。在具有多播或广播能力的局域网（如以太网）中，借助 VRRP 能在某台设备出现故障时仍然提供高可靠的缺省链路，有效避免单一链路发生故障后网络中断的问题，而无需修改动态路由协议、路由发现协议等配置信息，也无需修改主机的默认网关配置。
- 适应性强。VRRP 报文封装在 IP 报文中，支持各种上层协议。
- 网络开销小。VRRP 只定义了一种报文——VRRP 通告报文，并且只有处于 Master 状态的路由器可以发送 VRRP 报文。

## 2 VRRP 协议介绍

### 2.1 相关术语

- 虚拟路由器：由一个 Master 路由器和多个 Backup 路由器组成。主机将虚拟路由器当作默认网关。
- VRID：虚拟路由器的标识。有相同 VRID 的一组路由器构成一个虚拟路由器。
- Master 路由器：虚拟路由器中承担报文转发任务的路由器。
- Backup 路由器：Master 路由器出现故障时，能够代替 Master 路由器工作的路由器。
- 虚拟 IP 地址：虚拟路由器的 IP 地址。一个虚拟路由器可以拥有一个或多个 IP 地址。
- IP 地址拥有者：接口 IP 地址与虚拟 IP 地址相同的路由器被称为 IP 地址拥有者。
- 虚拟 MAC 地址：一个虚拟路由器拥有一个虚拟 MAC 地址。虚拟 MAC 地址的格式为 00-00-5E-00-01-{VRID}。通常情况下，虚拟路由器回应 ARP 请求使用的是虚拟 MAC 地址，只有虚拟路由器做特殊配置的时候，才回应接口的真实 MAC 地址。
- 优先级：VRRP 根据优先级来确定虚拟路由器中每台路由器的地位。
- 非抢占方式：如果 Backup 路由器工作在非抢占方式下，则只要 Master 路由器没有出现故障，Backup 路由器即使随后被配置了更高的优先级也不会成为 Master 路由器。
- 抢占方式：如果 Backup 路由器工作在抢占方式下，当它收到 VRRP 报文后，会将自己的优先级与通告报文中的优先级进行比较。如果自己的优先级比当前的 Master 路由器的优先级高，就会主动抢占成为 Master 路由器；否则，将保持 Backup 状态。

### 2.2 虚拟路由器简介

VRRP 将局域网内的一组路由器划分在一起，形成一个 VRRP 备份组，它在功能上相当于一台虚拟路由器，使用虚拟路由器号进行标识。以下使用虚拟路由器代替 VRRP 备份组进行描述。

虚拟路由器有自己的虚拟 IP 地址和虚拟 MAC 地址，它的外在表现形式和实际的物理路由器完全一样。局域网内的主机将虚拟路由器的 IP 地址设置为默认网关，通过虚拟路由器与外部网络进行通信。

虚拟路由器是工作在实际的物理路由器之上的。它由多个实际的路由器组成，包括一个 **Master** 路由器和多个 **Backup** 路由器。**Master** 路由器正常工作时，局域网内的主机通过 **Master** 与外界通信。当 **Master** 路由器出现故障时，**Backup** 路由器中的一台设备将成为新的 **Master** 路由器，接替转发报文的工作，如图2所示。

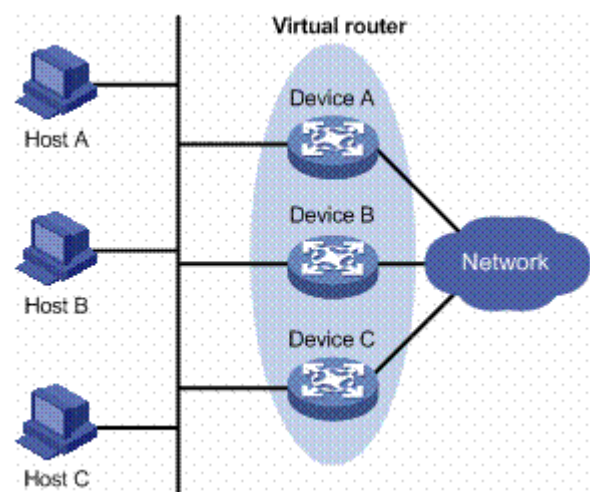


图2 虚拟路由器示意图

## 2.3 VRRP 工作过程

VRRP 的工作过程为：

- (1) 虚拟路由器中的路由器根据优先级选举出 **Master**。**Master** 路由器通过发送免费 ARP 报文，将自己的虚拟 MAC 地址通知给与它连接的设备或者主机，从而承担报文转发任务；
- (2) **Master** 路由器周期性发送 VRRP 报文，以公布其配置信息（优先级等）和工作状况；
- (3) 如果 **Master** 路由器出现故障，虚拟路由器中的 **Backup** 路由器将根据优先级重新选举新的 **Master**；
- (4) 虚拟路由器状态切换时，**Master** 路由器由一台设备切换为另外一台设备，新的 **Master** 路由器只是简单地发送一个携带虚拟路由器的 MAC 地址和虚拟 IP 地址信息的免费 ARP 报文，这样就可以更新与它连接的主机或设备中的 ARP 相关信息。网络中的主机感知不到 **Master** 路由器已经切换为另外一台设备。
- (5) **Backup** 路由器的优先级高于 **Master** 路由器时，由 **Backup** 路由器的工作方式（抢占方式和非抢占方式）决定是否重新选举 **Master**。

由此可见，为了保证 **Master** 路由器和 **Backup** 路由器能够协调工作，VRRP 需要实现以下功能：

- **Master** 路由器的选举；
- **Master** 路由器状态的通告；
- 同时，为了提高安全性，VRRP 还提供了认证功能；

下面将从上述三个方面详细介绍 VRRP 的工作过程。

### 2.3.1 Master 路由器的选举

VRRP 根据优先级来确定虚拟路由器中每台路由器的角色（Master 路由器或 Backup 路由器）。优先级越高，则越有可能成为 Master 路由器。

初始创建的路由器工作在 Backup 状态，通过 VRRP 报文的交互获知虚拟路由器中其他成员的优先级：

- 如果 VRRP 报文中 Master 路由器的优先级高于自己的优先级，则路由器保持在 Backup 状态；
- 如果 VRRP 报文中 Master 路由器的优先级低于自己的优先级，采用抢占工作方式的路由器将抢占成为 Master 状态，周期性地发送 VRRP 报文，采用非抢占工作方式的路由器仍保持 Backup 状态；
- 如果在一定时间内没有收到 VRRP 报文，则路由器切换为 Master 状态。

VRRP 优先级的取值范围为0到255（数值越大表明优先级越高），可配置的范围是1到254，优先级0为系统保留给路由器放弃 Master 位置时候使用，255则是系统保留给 IP 地址拥有者使用。当路由器为 IP 地址拥有者时，其优先级始终为255。因此，当虚拟路由器内存在 IP 地址拥有者时，只要其工作正常，则为 Master 路由器。

### 2.3.2 Master 路由器状态的通告

Master 路由器周期性地发送 VRRP 报文，在虚拟路由器中公布其配置信息（优先级等）和工作状况。Backup 路由器通过接收到 VRRP 报文的情况来判断 Master 路由器是否工作正常。

Master 路由器主动放弃 Master 地位（如 Master 路由器退出虚拟路由器）时，会发送优先级为0的 VRRP 报文，致使 Backup 路由器快速切换变成 Master 路由器。这个切换的时间称为 Skew time，计算方式为： $(256 - \text{Backup 路由器的优先级}) / 256$ ，单位为秒。

当 Master 路由器发生网络故障而不能发送 VRRP 报文的时候，Backup 路由器并不能立即知道其工作状态。Backup 路由器等待一段时间之后，如果还没有接收到 VRRP 报文，那么会认为 Master 路由器无法正常工作，而把自己升级为 Master 路由器，周期性发送 VRRP 报文。如果此时多个 Backup 路由器竞争 Master 路由器的位置，将通过优先级来选举 Master 路由器。Backup 路由器默认等待的时间称为 Master\_Down\_Interval，取值为： $(3 \times \text{VRRP 报文的发送时间间隔}) + \text{Skew time}$ ，单位为秒。

在性能不够稳定的网络中，Backup 路由器可能因为网络堵塞而在 Master\_Down\_Interval 期间没有收到 Master 路由器的报文，而主动抢占为 Master 位置，如果此时原 Master 路由器的报文又到达了，就会出现虚拟路由器的成员频繁的进行 Master 抢占现象。为了缓解这种现象的发生，特制定了延迟等待定时器。它可以使得 Backup 路由器在等待了 Master\_Down\_Interval 后，再等待延迟等待时间。如在此期间仍然没有收到 VRRP 报文，则此 Backup 路由器才会切换为 Master 路由器，对外发送 VRRP 报文。

### 2.3.3 认证方式

VRRP 提供了三种认证方式：

- 无认证：不进行任何 VRRP 报文的合法性认证，不提供安全性保障。
- 简单字符认证：在一个有可能受到安全威胁的网络中，可以将认证方式设置为简单字符认证。发送 VRRP 报文的路由器将认证字填入到 VRRP 报文中，而收到 VRRP 报文的路由器会将收到的 VRRP 报文中的认证字和本地配置的认证字进行比较。如果认证字相同，则认为接收到的报文是合法的 VRRP 报文；否则认为接收到的报文是一个非法报文。
- MD5 认证：在一个非常不安全的网络中，可以将认证方式设置为 MD5 认证。发送 VRRP 报文的路由器利用认证字和 MD5 算法对 VRRP 报文进行加密，加密后的报文保存在 Authentication Header（认证头）中。收到 VRRP 报文的路由器会利用认证字解密报文，检查该报文的合法性。

## 3 Comware 实现的技术特色

### 3.1 监视上行链路

VRRP 网络传输功能有时需要额外的技术来完善其工作。例如，Master 路由器到达某网络的链路突然断掉时，主机无法通过此 Master 路由器远程访问该网络。此时，可以通过监视指定接口上行链路功能，解决这个问题。当 Master 路由器发现上行链路出现故障后，主动降低自己的优先级（使 Master 路由器的优先级低于 Backup 路由器），并立即发送 VRRP 报文。Backup 路由器接收到优先级比自己低的 VRRP 报文后，等待 Skew\_Time 切换为新的 Master 路由器。从而，使得能够到达此网络的 Backup 路由器充当 VRRP 新的 Master 路由器，协助主机完成网络通讯。

VRRP 可以直接监视连接上行链路的接口状态。当连接上行链路的接口 down 时，将 Master 路由器降低指定的优先级。VRRP 优先级最低可以降低到1。

VRRP 可以利用 NQA 技术监视上行链路连接的远端主机或者网络状况。例如，Master 设备上启动 NQA 的 ICMP-echo 探测功能，探测远端主机的可达性。当 ICMP-echo 探测失败时，它可以通知本设备探测结果，达到降低 VRRP 优先级的目的。

VRRP 也可以利用 BFD 技术监视上行链路连接的远端主机或者网络状况。由于 BFD 的精度可以到达10ms，通过 BFD 能够快速检测到链路状态的变化，达到快速抢占的目的。例如，可以在 Master 路由器上使用 BFD 技术监视上行设备的物理状态，在上行设备坏掉之后，快速检测到该变化，并降低 Master 路由器的优先级，致使 Backup 路由器等待 Skew time 后，抢占成为新的 Master 路由器。

### 3.2 Backup 监视 Master 工作状态

Backup 路由器在 Master 路由器坏掉之后，正常情况下需要等待 Master\_Down\_Interval 才能切换为新的 Master 的位置，这段时间内主机将无法通信，因为此时没有 Master 设备替它转发报文。为了解决这个网络故障，Backup 设备提供了一个监听 Master 工作状态的功能，使得 Master 路由器坏掉之后 Backup 能够立即切换成为新的 Master 路由器，维持网络通讯。

Backup 路由器监视 Master 路由器采用的是具有快速检测功能的 BFD 技术。在 Backup 设备上使用该技术监视 Master 路由器的状态，一旦 Master 路由器发生故障，Backup 就可以自动切换成为新的 Master 路由器，将切换时间缩短到毫秒级。

## 4 典型组网案例

### 4.1 主备备份

主备备份方式表示业务仅由 Master 路由器承担。当 Master 路由器出现故障时，才会由选举出来的 Backup 路由器接替它工作。如图3中所示。



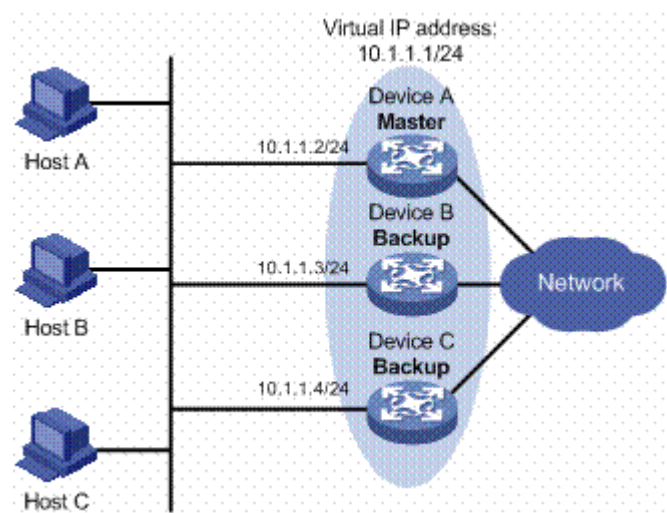


图3 主备份 VRRP

初始情况下，Device A 是 Master 路由器并承担转发任务，Device B 和 Device C 是 Backup 路由器且都处于就绪监听状态。如果 Device A 发生故障，则虚拟路由器内处于 Backup 状态的 Device B 和 Device C 路由器将根据优先级选出一个新的 Master 路由器，这个新 Master 路由器继续为网络内的主机转发数据。

## 4.2 负载分担

在路由器的一个接口上可以创建多个虚拟路由器，使得该路由器可以在一个虚拟路由器中作为 Master 路由器，同时其他的虚拟路由器中作为 Backup 路由器。

负载分担方式是指多台路由器同时承担业务，因此负载分担方式需要两个或者两个以上的虚拟路由器，每个虚拟路由器都包括一个 Master 路由器和若干个 Backup 路由器，各虚拟路由器的 Master 路由器可以各不相同，如图4中所示。

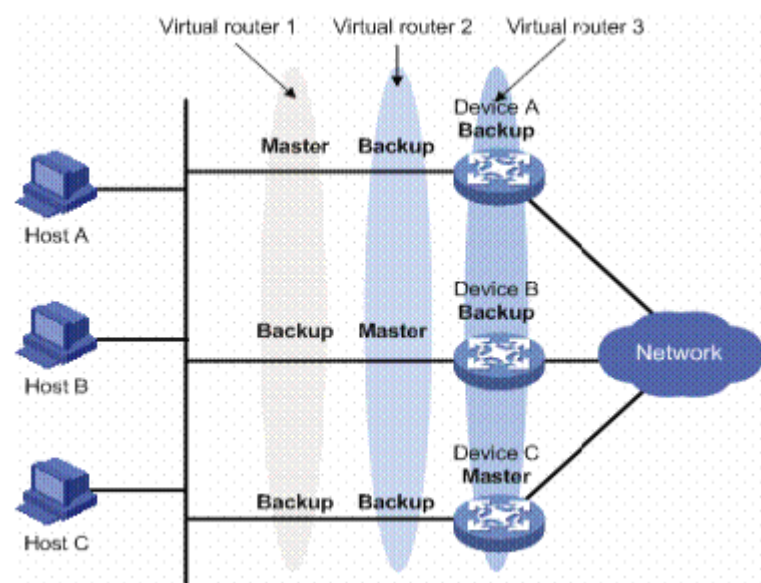


图4 负载分担 VRRP

在图4中，有三个虚拟路由器存在：

- 虚拟路由器1: Device A 作为 Master 路由器, Device B 和 Device C 作为 Backup 路由器。
- 虚拟路由器2: Device B 作为 Master 路由器, Device A 和 Device C 作为 Backup 路由器。
- 虚拟路由器3: Device C 作为 Master 路由器, Device A 和 Device B 作为 Backup 路由器。

为了实现业务流量在 Device A、Device B 和 Device C 之间进行负载分担, 需要将局域网内的主机的默认网关分别设置为虚拟路由器1、2和3。在配置优先级时, 需要确保三个虚拟路由器中各路由器的 VRRP 优先级形成一定的交叉, 使得一台路由器尽可能不同时充当2个 Master 路由器。

### 4.3 Master 使用 BFD/NQA 监视上行链路

VRRP 可以通过 BFD 或 NQA 等快速检测协议监视一些上行敏感链路, 使得 Master 路由器快速地发现网络故障, 降低自身的优先级, 从而保证上行链路工作正常的 Backup 路由器能够接替它的工作。

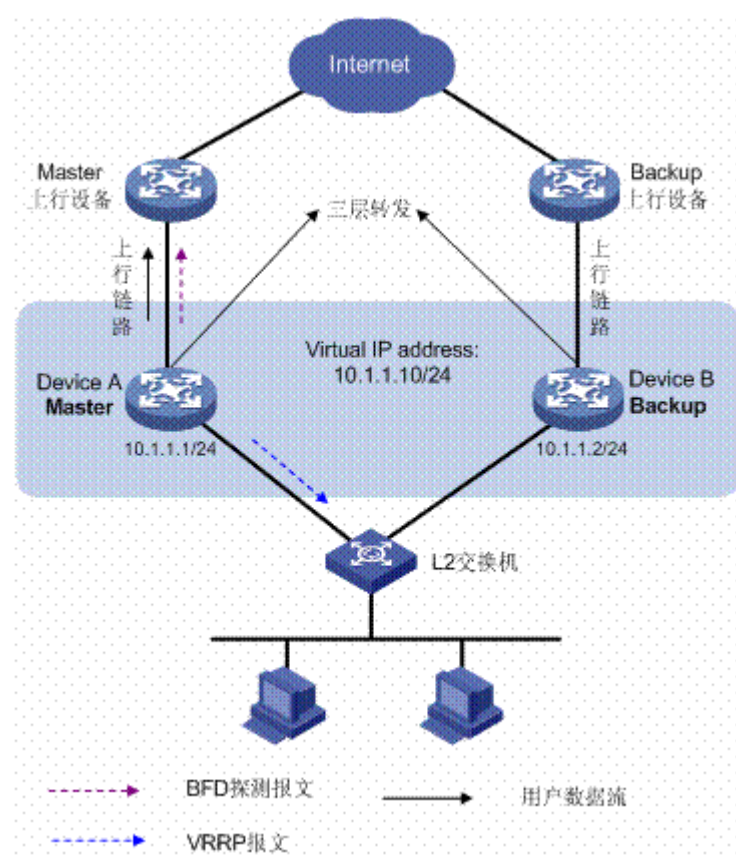


图5 Master 监视上行链路

如图5所示, 初始情况下, Device A 作为 Master 路由器, 承担转发任务; Device B 为 Backup 路由器, 处于就绪监听状态。Device A 使用 BFD 监视上行到达 Internet 的链路状态。如果 Device A 的上行链路发生故障, Device A 可以在毫秒级感知到网络变化, 立即发送低优先级的 VRRP 报文给 Device B。如果此时 Device B 的优先级高于报文中的优先级, 那么它将在 Skew Time 时间之后切换为新的 Master 路由器, 之后由这个新的 Master 路由器为网络内的主机转发数据。

### 4.4 Backup 使用 BFD 监视 Master 状态

为了保证网络传输的稳定性, 可以在 Backup 设备上使用 BFD 技术监视 Master 的状态, 使得 Master 设



备发生故障时，Backup 设备能够立即切换为新的 Master 设备。

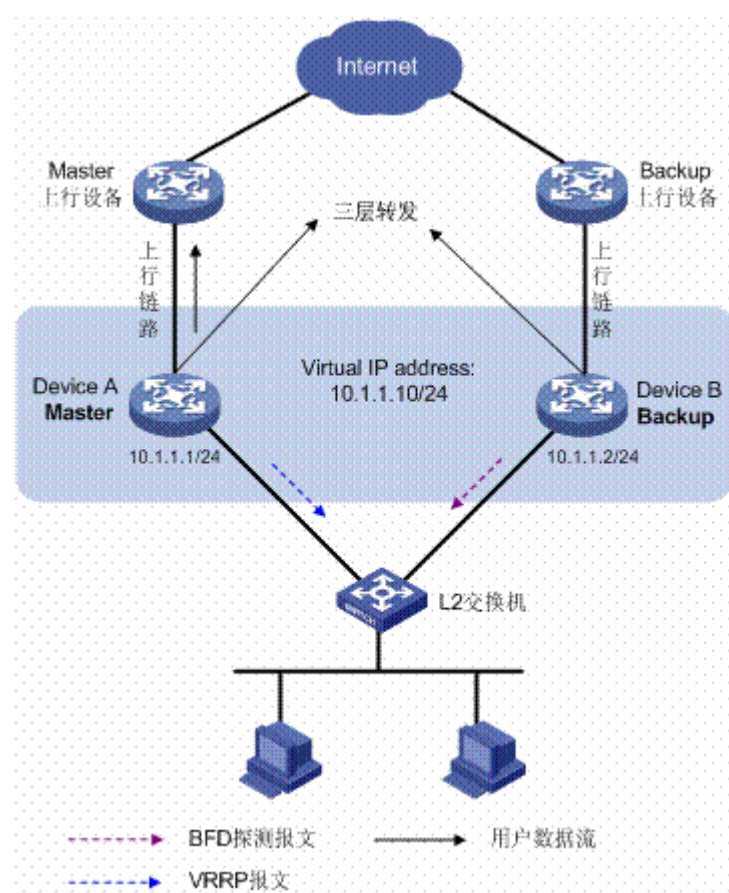


图6 Backup 监视 Master 状态

如图6中所示，初始情况下，Device A 作为 Master 路由器，承担转发任务；Device B 是 Backup 路由器，处于就绪监听状态。Device B 使用 BFD 监视 Device A 上 IP 地址10.1.1.1的可达性。如果 Device A 发生故障，Device B 可以立即通过 BFD 感知到对端的变化，主动切换成为新的 Master 设备，之后这个新 Master 路由器将为网络内的主机转发数据。

## 5 附录

### 5.1 参考文献

- RFC 3768: Virtual Router Redundancy Protocol (VRRP)

以上 vrrp 协议来自

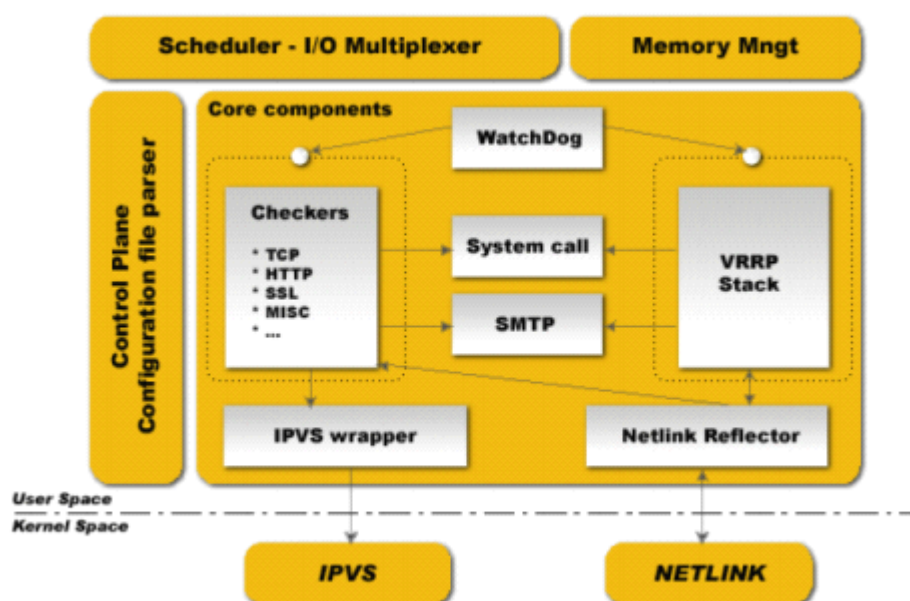
[http://www.h3c.com.cn/Products\\_Technology/Technology/Dependability/Other\\_technology/Tech\\_nology\\_book/200802/335873\\_30003\\_0.htm#\\_Toc189476503](http://www.h3c.com.cn/Products_Technology/Technology/Dependability/Other_technology/Tech_nology_book/200802/335873_30003_0.htm#_Toc189476503)

除了这篇文章，还可用参考 <http://bbs.ywlm.net/thread-790-1-1.html>

## 三、keepalive 软件安装

Keepalive 是一个高度模块化设计的软件，源代码的结构似乎很容易看出这一点，源码里面如下目录

```
check core etc include libipvs-2.4 libipvs-2.6 Makefile.in vrrp
```



Core keepalived 的核心程序,比如全局配置的解析,进程的启动等等;

Vrrp keepalived 的 vrrpd 子进程以及相关的代码。

Check keepalived 的 healthcheck 子进程的目录, 包括了所有的健康检测方式以及对应配置的解析, lvs 的配置解析也在这个里面。

Libipfwc iptables 库,主要用来配置 LVS 中的 firewall-mak。

Libipvs\* 也是试用 LVS 需要使用到的。

### 多进程模式

Keepalived 采用多进程的设计模式, 每个进程负责不同的功能, 我们在使用 lvs 的机器上通常可以看到如下进程:

111Keepalived&lt; 父进程: 内存管理, 监控子进程

112\\_Keepalived&lt; VRRPf 子进程

113\\_Keepalived&lt; healthchecker 子进程

可以通过某些命令行参数来控制 u 开启某些进程, 比如不运行 LVS 的机器上, 只开启 vrrp 就可以的话, 可以试用 -P 参数, 如果只运行了 healthcheck 子进程可以试用 -C 参数来实现。

### 控制面板

所谓的面板就是对配置文件的编译和解析, Keepalived 的配置文件解析比较另类, 并不是一次统统解析所有的配置, 只有用到某模块的时候才解析相应的配置, 在每个模块里面都可用看到 XXX\_parser.c 这样的文件, 就是做这个作用的。

### WatchDg

这种框架提供了对子进程 (VRRP 和 healthchecker) 的监控

### IPVS 封装

Keepalived 里面所有对 LVS 的相关操作并不直接使用 ipvsadm 这样的用户端程序, 而是直接使用 IPVS 提供的函数进程操作, 这些代码都在 check/ipwrapper.c 中

### 安装过程

```
#!/bin/bash
mkidr keepalived
```

```
cd keepalived
wget http://www.keepalived.org/software/keepalived-1.2.2.tar.gz
tar xvf keepalived-1.2.2.tar.gz
cd keepalived-1.2.2
./configure --prefix=/usr --bindir=/usr/bin --sbindir=/usr/bin --libexecdir=/usr/libexec
--localstatedir=/var --libdir=/lib64 --infodir=/usr/share/info --sysconfdir=/etc
--mandir=/usr/local/share/man --with-kernel-dir=/usr/src/kernels/2.6.32-220.el6.x86_64
make && make install
```

注意: --with-kernel-dir 需要指定正确的内核位置, 不然会出现以下的提示 LVS选项为 No

```
Keepalived configuration
-----
Keepalived version      : 1.2.2
Compiler                 : gcc
Compiler flags           : -g -O2
Extra Lib                : -lpopt -lssl -lcrypto -lnl
Use IPVS Framework       : No
IPVS sync daemon support : No
Use VRRP Framework      : Yes
Use Debug flags          : No
```

查看 configure 的可选参数如下

```
[root@localhost keepalived-1.2.2]# ./configure --help
Installation directories:
  --prefix=PREFIX          install architecture-independent files in PREFIX [/usr/local]
  --exec-prefix=EPREFIX    install architecture-dependent files in EPREFIX [PREFIX]

By default, `make install' will install all the files in
`/usr/local/bin', `/usr/local/lib' etc. You can specify
an installation prefix other than `/usr/local' using `--prefix',
for instance `--prefix=$HOME'.
For better control, use the options below.
Fine tuning of the installation directories:
  --bindir=DIR             user executables [EPREFIX/bin]
  --sbindir=DIR            system admin executables [EPREFIX/sbin]
  --libexecdir=DIR         program executables [EPREFIX/libexec]
  --sysconfdir=DIR         read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR     modifiable architecture-independent data [PREFIX/com]
  --localstatedir=DIR      modifiable single-machine data [PREFIX/var]
  --libdir=DIR             object code libraries [EPREFIX/lib]
  --includedir=DIR         C header files [PREFIX/include]
  --oldincludedir=DIR      C header files for non-gcc [/usr/include]
  --datarootdir=DIR        read-only arch.-independent data root [PREFIX/share]
  --datadir=DIR            read-only architecture-independent data [DATAROOTDIR]
  --infodir=DIR            info documentation [DATAROOTDIR/info]
  --localedir=DIR          locale-dependent data [DATAROOTDIR/locale]
  --mandir=DIR            man documentation [DATAROOTDIR/man]
  --docdir=DIR            documentation root [DATAROOTDIR/doc/PACKAGE]
  --htmldir=DIR           html documentation [DOCDIR]
```

```
--dvidir=DIR          dvi documentation [DOCDIR]
--pdfdir=DIR          pdf documentation [DOCDIR]
--psdir=DIR           ps documentation [DOCDIR]

Optional Features:
--disable-option-checking ignore unrecognized --enable/--with options
--disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
--disable-lvs-syncd     do not use LVS synchronization daemon
--disable-lvs           do not use the LVS framework
--disable-vrrp          do not use the VRRP framework
--enable-debug          compile with debugging flags
--enable-profile        compile with profiling flags

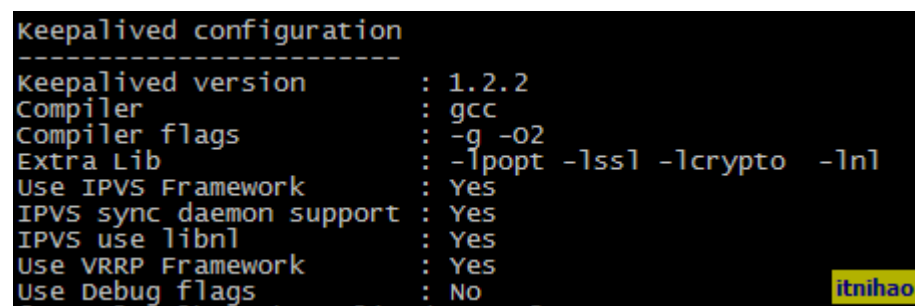
Optional Packages:
--with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
--without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)
--with-kernel-dir=DIR   path to linux kernel source directory
--with-kernel-version=VER forced value for linux kernel version (VER=2.4|2.6)

Some influential environment variables:
CC          C compiler command
CFLAGS      C compiler flags
LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
            nonstandard directory <lib dir>
LIBS        libraries to pass to the linker, e.g. -l<library>
CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
            you have headers in a nonstandard directory <include dir>
CPP         C preprocessor

Use these variables to override the choices made by `configure' or to help
it to find libraries and programs with nonstandard names/locations.

Report bugs to the package provider.
```

Configure 正确后会得到以下提示



```
Keepalived configuration
-----
Keepalived version      : 1.2.2
Compiler                : gcc
Compiler flags          : -g -O2
Extra Lib               : -lpopt -lssl -lcrypto -lnl
Use IPVS Framework      : Yes
IPVS sync daemon support : Yes
IPVS use libnl          : Yes
Use VRRP Framework      : Yes
Use Debug flags         : No
```

注意

Use IPVS Framework      ipvs 框架--也即 L V S 的核心代码框架，如果不使用 L V S，可用在 configure 时指定参数--disable-lvs，这样的话，这里看到的的就是 No 不是 Yes,如下所示：

```
Keepalived version      : 1.2.2
Compiler                 : gcc
Compiler flags           : -g -O2
Extra Lib                : -lpopt -lssl -lcrypto -lnl
Use IPVS Framework       : No
IPVS sync daemon support : No
Use VRRP Framework      : Yes
Use Debug flags          : No
```

IPVS sync daemon support    IPVS 同步进程，很显然，如果前面那项是 No 的话，那么这里肯定也是 No，当然如果前面这项是 Yes，即使用 LVS，而不想使用 LVS 的同步进程(sync daemon)，可以在 configure 的时候指定--disable-lvs-syncd

IPVS use libnl

Use VRRP Framework      VRRP 框架，是 Keepalived 的核心进程 vrrpd

Use Debug flags

## 四、keepalive 配置详解

Keep 的配置文件可以分为三类、

1.全局配置文件

2.VRRPD 配置

3.LVS 配置

全局配置：对整个 Keepalive 配置生效的，不管是否使用 LVS

VRRPD 是 Keepalived 的核心

LVS 配置只在使用 Keepalived 来配置和管理 LVS 时才需要使用，如果仅仅使用 Keepalived 做 HA，LVS 的配置完全是不需要的。

配置文件都是以块（block）形式组织的，每个块都在{和}包围的范围内。#和！开头的行都是注释

```
#全局配置--包括两个子配置，全局定义（global definition）和静态地址路由（static ipaddress/routes）
! Configuration File for keepalived
global_defs {
    #全局配置
    notification_email {
        itnihao@qq.com      #发生事件切换的时候，发送的邮箱，可以有多个，每行一个
                           #需要接收邮箱1
        itnihao@admin.com   #需要接收邮箱2
    }
    notification_email_from itnihao@server.com #发件箱
    smtp_server 127.0.0.1   #指定邮件服务的地址
    smtp_connect_timeout 30 #连接 smtp 超时间隔
    router_id LVS_DEVEL     #运行 keepalive 机器的标示，注意每个机器需要唯一的标示
}

#静态地址和路由
所谓的静态（static）就是说不会随 vrrpd instance 的开/关而变化，VIP 就不是 static 的，会随着 vrrpd
而添加/删除。整个配置可以用来给服务器配置静态的 IP 地址/路由，当然如果服务器的配置里面已经有这
```

些配置，这里就不需要设置了。

```
static_ipaddress
```

```
{
```

```
    192.168.1.1/24 brd + dev eth0 scope global
```

#keepalive 最终运行的命令是 ip addr add 192.168.1.1/24 brd + dev eth0 scope global

#这里的配置要符合 linux 下的命令规则

```
}
```

```
static_routes
```

```
{
```

```
    src $SRC_IP to $DST_IP dev $SRC_DEVICE
```

#源地址，目的地址

```
    src $SRC_IP to $DST_IP via $GW dev $SRC_DEVICE
```

```
}
```

#=====全局配置到此结束=====

#VRRPD 的配置包括2部分：VRRPD 同步组(synchroizstion group)和 VRRPD 实例(VRRPD instance)

```
VRRPD Sync Groups
```

不适用 Sync Group 的话，如果机器(或者说 router)有两个网段，一个内网，一个外网，每个网段开启一个 VRRPD 实例，假设 VRRPD 配置为检查内网，那么当外网出现问题时，VRRPD 认为自己仍然健康，那么不会发送 Master 和 Backup 的切换，从而导致了问题。Sync group 就是为了解决这个问题，可以把两个实例放进一个 Sync Group，这样的话，group 里面任何一个实例出现问题都会发生切换。

```
vrrp_syncv_group VG_1 {
```

```
group {
```

```
    inside_network
```

```
    outside_network
```

```
}
```

```
notify_master /path/to/to_master.sh
```

```
notify_backup /path/to/to_backup.sh
```

```
notify_fault "/path/fault.sh VG_1"
```

```
notify /path/to/notify.sh
```

```
smtp_alter
```

```
}
```

#notify\_mater 指定当切换到 Master 时，指定的脚本，这个脚本可以传入参数（引号引起），其他2个类推

#notify 指定有3个参数，这些参数由 Keepalived 提供：\$1(GROUP-INSTANCE), \$2(group 或者 instance 的名字), \$3(MASTER\_BACKUP-FAULT)

#smtp\_alter 使用 global\_defs 里面定义的邮件地址和 smtp 服务器在切换后发送邮件通知

#VIP

```
vrrp_instance VI_1 {
```

```
    state MASTER
```

#两种角色，MASTER or BACKUP 备份服务器此处是 BACKUP

#state 指定 instance 的初始(initial)状态，在两台 router 都启动后，马上会发生竞选，高 priority 的会竞选为 Master，所以这里的额 state 并不表示这台就是一直是 Master

```
    interface eth0
```

#实例绑定的网卡

```
    dont_track_primary
```

#忽略 VRRP 的 interface 错误（默认不设置）

```
    track_interface { #设置额外的监控，里面的任意一个网卡出现问题，都会进入 FAULT 状态
```



```

        eth0
        eth1
    }

    mcast_src_ip <IPADDR> #发送多播包的地址，如果不设置，默认使用绑定的网卡的 primary IP
    Garp_master_delay 10  #在切换到 MASTER 状态后，延迟进行 gratuitous ARP 请求
    virtual_router_id 51  #VRID 标记 (0...255)
    priority 100          #优先级，另一台改为90
    advert_int 1          #检查间隔，默认1s
    nopreempt             #不抢占，只在优先级高的机器上设置即可，优先级低的机器不设置

    lvs_sync_daemon_interface #lvs syncd 绑定的网卡
    authentication {       #认证
        auth_type PASS     #认证的方式，支持 PASS 和 AH
        auth_pass 1111     #认证的密码
    }

    virtual_ipaddress {    #指定漂移地址 (VIP)
        192.168.16.200     #如果有多个 VIP，继续换行填写
    }

    virtual_routes {       #发生切换的时候添加/删除路由
        src 192.168.16.1 to 192.168.1.0/24 via 192.168.16.21 dev eth1
        192.168.2.0/24 via 192.168.16.21 dev eth1
        192.168.3.0/24 dev eth2
        192.168.4.0/24 via 192.168.16.21

        preempt_deay      #抢占延迟，默认为5分钟
        Debug              #debug 级别
    }
}

#=====VRRPD 配置到此结束=====
#LVS 配置-包括两部分：虚拟主机组(virtual server group)和虚拟主机(virtual server)，这些配置都会传递给 ipvsadm 作为参数
虚拟主机组
    这个配置段是可选的，目的是为了 Let 一台 RealServer 上的某个 service 可以属于多个 Virtual Server，并且只做一次监控检查。
virtual_server_group <STRING> {
#VIP port
    <IPADDR> <PORT>
    <IPADDR> <PORT>
    fwmark <INT>
}
虚拟主机 可以用下面三种方式中的任意一种配置
1.virtual_server IPport
2.virtual_server fwmark int
3.virtual_server group sting

```

```

virtual_server 192.168.16.200 80 { #设置 VIP port
    delay_loop 2                #每个2秒检查一次 real_server 状态
    lb_algo wrr                 #LVS 调度算法    #lb_algo rr|wrr|lc|wlc|sh|dh|lbic
    lb_kind DR                  #LVS 集群模式    #lb_kind NAT|DR|TUN
    persistence_timeout 60      #会话保持时间
    #persistence_granularity <NETMASK> #lvs 会话保持度, ipvsadm中的-M 参数, 默认是0xffffffff,
    #即根据每个客户端做会话保持
    #virtualhost <sting>        #HTTP_GET 做健康检查时, 检查的 Web 服务器的虚拟主机(即 Host:头)
    protocol TCP                 #使用协议 TCP 或者 UDP #protocol TCP|UDP
    ha_suspend                   #suspendhealthchecker's activity
    #sorry_server @IP PORT      #备用机, 当所有的 realserver 失效后使用

    real_server 192.168.16.253 80 {
        weight 3                #权重, 默认为1; 0为失效
        inhibit_on_failure      #在服务器健康检查失败时, 将 weight 设置为0, 而不是直接从 IPVS 里面
        #删除
        # notify_down "/root/realserver.sh start" #检测到服务 down 后执行的脚本
        # notify_up "/root/realserver.sh stop"   #检测到服务器 up 后执行的脚本
        TCP_CHECK {              #tcp 健康检查
            connect_timeout 10    #连接超时时间
            nb_get_retry 3        #重连次数
            delay_before_retry 3  #重连间隔时间
            connect_port 80       #健康检查端口
        }
    }

    real_server 192.168.16.252 80 {
        weight 3                #权重
        # notify_down "/root/realserver.sh start" #检测到服务 down 后执行的脚本
        # notify_up "/root/realserver.sh start"   #检测到服务 up 后执行的脚本
        # TCP 方式的监控检查
        TCP_CHECK {
            connect_port 80       #健康检查端口
            bindto 192.168.16.200
            connect_timeout 10    #连接超时时间
            nb_get_retry 3        #重连次数
            delay_before_retry 3  #重连间隔时间
        }
    }
}

#MISC 健康检查方式, 执行一个程序
MISC_CHECK 检测脚本
{
    misc_path /path_to_script/script.sh #外部程序脚本路径
    misc_timeout <INT>                  #脚本执行的超时时间
    misc_dynamic
}

```

如果设置了 misc\_dynamic 的话,healthchecker 程序的退出状态码会用来动态调整服务器的权重(weight)

返回0: 健康检查 OK, 权重不被修改

返回1: 健康检查失败, 权重设为0

返回2-255, 健康检查 OK, 权重设置为: 退出状态码-2, 比如返回255, 那么 weight=255-2=253

```
(or misc_path "/path_to_script/script.sh"
}
```

```
#HTTP_GET|SSL_GET #健康检测的方式 TCP_CHECK |SSL_GET|HTTP_GET|
{
    url {
        #HTTP/SSL 检查的 URL, 这里可以指定多个 url
        # You can add multiple url block
        path /
        digest la23ddd32ddddd #SSL 检查后的摘要信息 (genhash 工具算出)
        status_code 200 #HTTP 检查的返回状态码
    }

    connect_port 80 #监控检查端口
    bindto 192.168.16.200 #以此地址发送请求对服务器进行监控检查
    connect_timeout num #连接超时时间
    nb_get_retry num #重试次数
    delay_before_retry num #重连间隔时间
}
}
```

#-----扩展参数-----

```
vrrp_sync_group string {
group {
string
}
notify_master /path_to_script/script_master.sh
(or notify_master " /path_to_script/script_master.sh <arg_list>")
notify_backup /path_to_script/script_backup.sh
(or notify_backup " /path_to_script/script_backup.sh <arg_list>")
notify_fault /path_to_script/script_fault.sh
(or notify_fault " /path_to_script/script_fault.sh <arg_list>")
}
vrrp_instance string {
state MASTER|BACKUP
interface string
mcast_src_ip @IP
lvs_sync_daemon_interface string
```

```
virtual_router_id num
priority num
advert_int num
smtp_alert
authentication {
auth_type PASS|AH
auth_pass string
}
virtual_ipaddress { # Block limited to 20 IP addresses
@IP
@IP
}
virtual_ipaddress_excluded { # Unlimited IP addresses number
@IP
@IP
}
notify_master /path_to_script/script_master.sh
(or notify_master “ /path_to_script/script_master.sh <arg_list>” )
notify_backup /path_to_script/script_backup.sh
(or notify_backup “ /path_to_script/script_backup.sh <arg_list>” )
notify_fault /path_to_script/script_fault.sh
(or notify_fault “/path_to_script/script_fault.sh <arg_list>” )
}
```

```
real_server @IP PORT
{
    weight num    权重
    MISC_CHECK    检测脚本
    {
        misc_path /path_to_script/script.sh  脚本路径
        (or misc_path “/path_to_script/script.sh
    }
}

real_server @IP PORT
{
    weight num          #权重
    HTTP_GET|SSL_GET    #健康检测的方式 TCP_CHECK |SSL_GET|HTTP_GET|
    {
        url
        { # You can add multiple url block
            path alphanum          路径
            digest alphanum        校验码
        }
    }
}
```

```

    }
    connect_port num          连接端口
    connect_timeout num       连接超时
    nb_get_retry num          重试时间间隔
    delay_before_retry num     两次成功 retry 之间的时间间隔
  }
}

```

一些需要注意的事项:

全局定义块

1. email 通知。作用：有故障，发邮件报警。这是可选项目，建议不用，用 nagios 全面监控代替之。
2. Lvs 负载均衡器标识 (lvs\_id)。在一个网络内，它应该是唯一的。
3. 花括号“{}”。用来分隔定义块，因此必须成对出现。如果写漏了，keepalived 运行时，不会得到预期的结果。由于定义块内存在嵌套关系，因此很容易遗漏结尾处的花括号，这点要特别注意。

VRRP 定义块

1、 同步 vrrp 组 vrrp\_sync\_group。作用：确定失败切换 (FailOver) 包含的路由实例个数。即在有2个负载均衡器的场景，一旦某个负载均衡器失效，需要自动切换到另外一个负载均衡器的实例是哪些？

2、 实例组 group。至少包含一个 vrrp 实例。

3、 Vrrp 实例 vrrp\_instance。实例名出自实例组 group 所包含的那些名字。

(1) 实例状态 state。只有 MASTER 和 BACKUP 两种状态，并且需要大写这些单词。其中 MASTER 为工作状态，BACKUP 为备用状态。当 MASTER 所在的服务器失效时，BACKUP 所在的系统会自动把它的状态有 BACKUP 变换成 MASTER；当失效的 MASTER 所在的系统恢复时，BACKUP 从 MASTER 恢复到 BACKUP 状态。

(2) 通信接口 interface。对外提供服务的网络接口，如 eth0,eth1。当前主流的服务器都有2个或2个以上的接口，在选择服务接口时，一定要核实清楚。

(3) lvs\_sync\_daemon\_inteface。负载均衡器之间的监控接口，类似于 HA HeartBeat 的心跳线。但它的机制优于 Heartbeat，因为它没有“裂脑”这个问题，它是以优先级这个机制来规避这个麻烦的。在 DR 模式中，lvs\_sync\_daemon\_inteface 与服务接口 interface 使用同一个网络接口。

(4) 虚拟路由标识 virtual\_router\_id。这个标识是一个数字，并且同一个 vrrp 实例使用唯一的标识。即同一个 vrrp\_stance, MASTER 和 BACKUP 的 virtual\_router\_id 是一致的，同时在整个 vrrp 内是唯一的。

(5) 优先级 priority。这是一个数字，数值愈大，优先级越高。在同一个 vrrp\_instance 里，MASTER 的优先级高于 BACKUP。若 MASTER 的 priority 值为150，那么 BACKUP 的 priority 只能是140或更小的数值。

(6) 同步通知间隔 advert\_int。MASTER 与 BACKUP 负载均衡器之间同步检查的时间间隔，单位为秒。

(7) 验证 authentication。包含验证类型和验证密码。类型主要有 PASS、AH 两种，通常使用的类型为 PASS，据说 AH 使用时有问题。验证密码为明文，同一 vrrp 实例 MASTER 与 BACKUP 使用相同的密码才能正常通信。

4、 虚拟 ip 地址 virtual\_ipaddress。可以有多个地址，每个地址占一行，不需要指定

子网掩码。注意：这个 ip 必须与我们在 lvs 客户端设定的 vip 相一致！

- 虚拟服务器 virtual\_server 定义块

虚拟服务器定义是 keepalived 框架最重要的项目了，是 keepalived.conf 必不可少的部分。

1、 虚拟服务器 virtual\_server. 这个 ip 来自于 vrrp 定义块的第“4”步，后面一个空格，然后加上端口号。定义一个 vip，可以实现多个 tcp 端口的负载均衡功能。

(1) delay\_loop. 健康检查时间间隔，单位是秒。

(2) lb\_algo. 负载均衡调度算法，互联网应用常使用 wlc 或 rr。

(3) lb\_kind. 负载均衡转发规则。一般包括 DR,NAT,TUN3种。

(4) persistence\_timeout. 会话保持时间，单位是秒。这个选项对动态网站很有用处：当用户从远程用帐号进行登陆网站时，有了这个会话保持功能，就能把用户的请求转发给同一个应用服务器。在这里，我们来做一个假设，假定现在有一个 lvs 环境，使用 DR 转发模式，真实服务器有3个，负载均衡器不启用会话保持功能。当用户第一次访问的时候，他的访问请求被负载均衡器转给某个真实服务器，这样他看到一个登陆页面，第一次访问完毕；接着他在登陆框填写用户名和密码，然后提交；这时候，问题就可能出现了一登陆不能成功。因为没有会话保持，负载均衡器可能会把第2次的请求转发到其他的服务器。

(5) 转发协议 protocol. 一般有 tcp 和 udp 两种。

2、 真实服务器 real\_server. 也即服务器池。Real\_server 的值包括 ip 地址和端口号。多个连续的真实 ip，转发的端口相同，是不是可以以范围表示？需要进一步实验。如写成 real\_server 61.135.20.1-10 80。

(1) 权重 weight. 权重值是一个数字，数值越大，权重越高。使用不同的权重值的目的在于为不同性能的机器分配不同的负载，性能较好的机器，负载分担大些；反之，性能差的机器，则分担较少的负载，这样就可以合理的利用不同性能的机器资源。

(2) Tcp 检查 tcp\_check.

## 五、keepalive+lvs 实现高可用集群

### 5.1. 服务模块概述

通过 keepalive+lvs 构建高可用集群，实现对 web 服务的负载均衡，从而提供系统的总容量。本例只实现了基本的 web 服务负载均衡，在集群环境中，还会有 cache 集群，mysql 集群，rsync 数据同步，cacti+nagios 的监控等复杂架构

以下系统采用 centos6.2X86\_64 环境，系统环境的调优此例不涉及到，在此都是默认关闭 iptables 的，对 lvs 而言，建议关闭 iptables，而改用其他安全策略，如 tcpwrap，shell 脚本等，对 web 服务器，可以根据需要设置 iptables 规则。

模块以及 ip 信息如下所示

模块名称	作用	Ip 地址	服务类型	备注
Keepalive(master) Lvs+DR	提供前端负载均衡分发（主）	192.168.16.20 (vip)	调度器 (主)	用户访问 vip 地址
		192.168.16.21 (dip1)		
Keepalive(backup) Lvs+DR	提供前端负载均衡分发（备）	192.168.16.20 (vip)	调度器 (备)	用户访问 vip 地址
		192.168.16.22 (dip2)		
Node1	Web 服务节点 1	192.168.16.2	http 服务	访问的实际节点
Node2	Web 服务节点 2	192.168.16.3		
Node3	Web 服务节点 3	192.168.16.4		

### 5.2 服务的配置

#### 5.2.1 node1-3 的 web 服务安装配置以及



```

yum install httpd &&
chkconfig --add httpd &&
chkconfig --level 345 httpd on
echo "this is 0.2" >/var/www/html/index.html
service httpd restart &&
同理设置 显示 this is 0.3 .....
设置 iptables 等，此处省略 N 多文字.....读者自己完善

```

最终能出现以下，说明 2-4 的 web 服务配置访问正常了

```

bash-2.03$ curl http://192.168.16.2
this is 0.2
bash-2.03$ curl http://192.168.16.3
this is 0.3
bash-2.03$ curl http://192.168.16.4
this is 0.4
bash-2.03$

```

itnihao

Web 服务配置完后，在 192.168.16.2-4 上面做以下的操作，注意要执行先设置 arp 参数

```

echo "2" > /proc/sys/net/ipv4/conf/all/arp_announce
echo "1" > /proc/sys/net/ipv4/conf/all/arp_ignore
echo "1" > /proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" > /proc/sys/net/ipv4/conf/lo/arp_announce
ifconfig lo:0 192.168.16.20 netmask 255.255.255.255 broadcast 192.168.16.20 up
route add -host 192.168.16.20 dev lo:0

```

注意，这里的设置会在重启机器后失效，因此，可以把以上命令添加开机启动项

```

[root@localhost ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1A:B4:71
          inet addr:192.168.16.2  Bcast:192.168.16.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:163358 errors:0 dropped:0 overruns:0 frame:0
          TX packets:124677 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11520437 (10.9 MiB)  TX bytes:12111516 (11.5 MiB)
          Interrupt:59 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:58 errors:0 dropped:0 overruns:0 frame:0
          TX packets:58 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8987 (8.7 KiB)  TX bytes:8987 (8.7 KiB)

lo:0      Link encap:Local Loopback
          inet addr:192.168.16.20  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:16436  Metric:1

```

itnihao

在三台机器上面看，都有相同的 ip 192.168.16.20 (lo:0)

```

[root@localhost ~]# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.16.20 0.0.0.0 255.255.255.255 UH 0 0 0 lo
192.168.16.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
169.254.0.0 0.0.0.0 255.255.0.0 U 0 0 0 eth0
0.0.0.0 192.168.16.1 0.0.0.0 UG 0 0 0

```

itnihao

## 5.2.2master 的安装

```

# uname -a
Linux localhost.localdomain 2.6.32-220.el6.x86_64 #1 SMP Tue Dec 6 19:48:22 GMT 2011 x86_64
x86_64 x86_64 GNU/Linux

```

```
#安装 lvs
yum install popt popt-devel popt-static libnl libnl-devel
cd /usr/local/src
mkdir lvs_install && cd lvs_install
wget http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.26.tar.gz
tar xvf ipvsadm-1.26.tar.gz
cd ipvsadm-1.26
make
make install
cd ../
#安装 keepalived
wget http://www.keepalived.org/software/keepalived-1.2.2.tar.gz
tar xvf keepalived-1.2.2.tar.gz
cd keepalived-1.2.2
./configure --prefix=/usr --bindir=/usr/bin --sbindir=/usr/bin --libexecdir=/usr/libexec
--localstatedir=/var --libdir=/lib64 --infodir=/usr/share/info --sysconfdir=/etc
--mandir=/usr/local/share/man --with-kernel-dir=/usr/src/kernels/2.6.32-220.el6.x86_64
make
make install
cd ../
chkconfig --add keepalived
chkconfig --level 345 keepalived on
```

### Keepalived master 的配置文件如下

```
global_defs {
    notification_email {
        itnihao@qq.com
    }
    notification_email_from itnihao@163.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id lvs_drl
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass itnihao
    }
}
```

```
}
virtual_ipaddress {
    192.168.16.20
}
}

virtual_server 192.168.16.20 80
{
    delay_loop 2
    lb_algo rr
    lb_kind DR
    nat_mask 255.255.255.0
    persistence_timeout 50
    protocol TCP

    real_server 192.168.16.2 80
    {
        weight 1
        TCP_CHECK
        {
            connect_timeout 10
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }

    real_server 192.168.16.3 80
    {
        weight 1
        TCP_CHECK
        {
            connect_timeout 10
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }

    real_server 192.168.16.4 80
    {
        weight 1
        TCP_CHECK
        {
```

```
        connect_timeout 10
        nb_get_retry 3
        delay_before_retry 3
        connect_port 80
    }
}
```

开启 keepalived master 的服务

Service keepalived restart

### =====5.2.3 backup 的安装=====

```
# uname -a
Linux localhost.localdomain 2.6.32-220.el6.x86_64 #1 SMP Tue Dec 6 19:48:22 GMT 2011 x86_64
x86_64 x86_64 GNU/Linux
#安装 lvs
yum install popt popt-devel popt-static libnl libnl-devel
cd /usr/local/src
mkdir lvs_install && cd lvs_install
wget http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.26.tar.gz
tar xvf ipvsadm-1.26.tar.gz
cd ipvsadm-1.26
make
make install
cd ../
#安装 keepalived
wget http://www.keepalived.org/software/keepalived-1.2.2.tar.gz
tar xvf keepalived-1.2.2.tar.gz
cd keepalived-1.2.2
./configure --prefix=/usr --bindir=/usr/bin --sbindir=/usr/bin --libexecdir=/usr/libexec
--localstatedir=/var --libdir=/lib64 --infodir=/usr/share/info --sysconfdir=/etc
--mandir=/usr/local/share/man --with-kernel-dir=/usr/src/kernels/2.6.32-220.el6.x86_64
make
make install
cd ../
chkconfig --add keepalived
chkconfig --level 345 keepalived on
```

Keepalived backup 的配置文件如下

```
global_defs {
    notification_email {
        itnihao@qq.com
    }
}
```

```
notification_email_from itnihao@163.com
smtp_server 127.0.0.1
smtp_connect_timeout 30
router_id lvs_dr2
}
```

```
vrrp_instance VI_1 {
    state BACKUP    #注意此处为 backup
    interface eth0
    virtual_router_id 51
    priority 99     #优先级比 master 低
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass itnihao
    }
    virtual_ipaddress {
        192.168.16.20
    }
}
```

```
virtual_server 192.168.16.20 80
{
    delay_loop 2
    lb_algo rr
    lb_kind DR
    nat_mask 255.255.255.0
    persistence_timeout 50
    protocol TCP

    real_server 192.168.16.2 80
    {
        weight 1
        TCP_CHECK
        {
            connect_timeout 10
            nb_get_retry 3
            delay_before_retry 3
            connect_port 80
        }
    }

    real_server 192.168.16.3 80
    {
```

```
weight 1
TCP_CHECK
{
    connect_timeout 10
    nb_get_retry 3
    delay_before_retry 3
    connect_port 80
}
}

real_server 192.168.16.4 80
{
    weight 1
    TCP_CHECK
    {
        connect_timeout 10
        nb_get_retry 3
        delay_before_retry 3
        connect_port 80
    }
}
}
```

开启 keepalived backup 的服务

Service keepalived restart

在 master 查看负载 web 主机

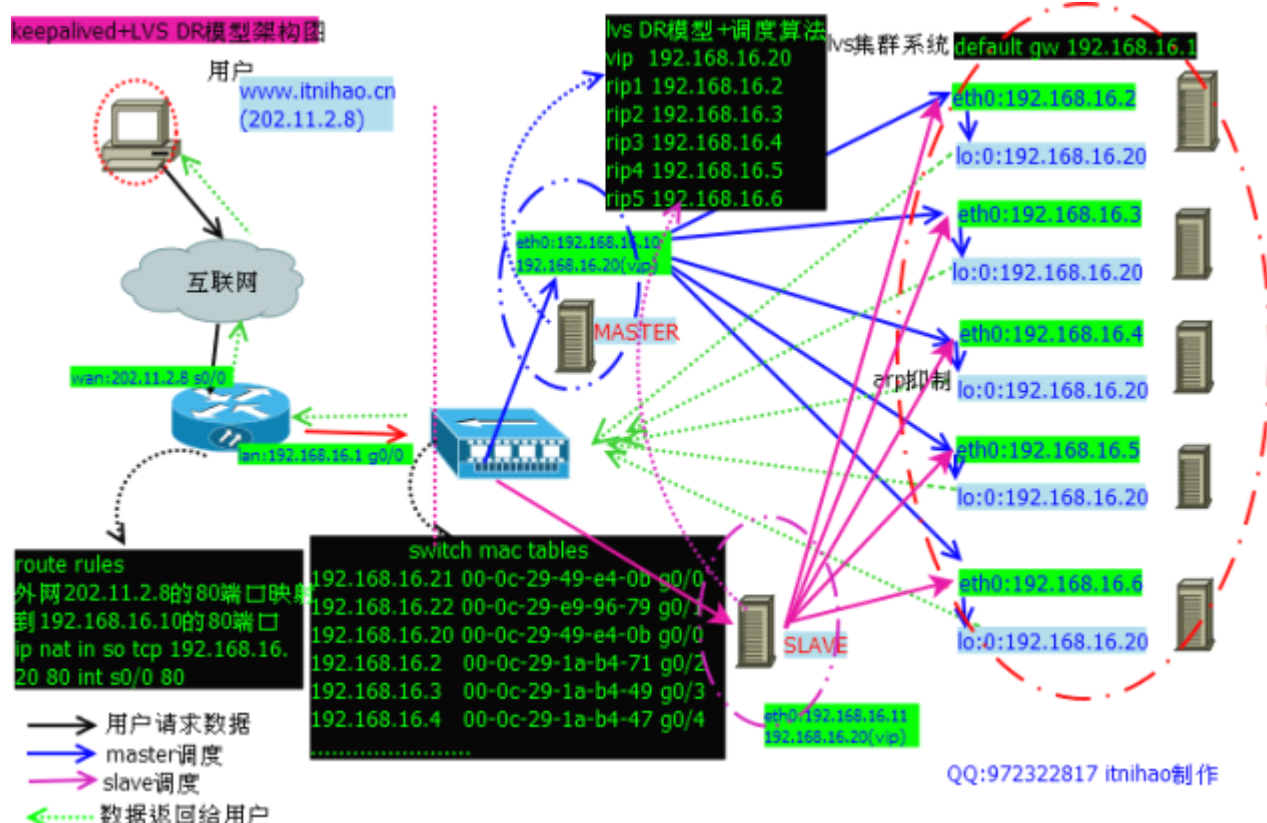
```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:49:e4:0b brd ff:ff:ff:ff:ff:ff
    inet 192.168.16.20/32 scope global eth0
    inet 192.168.16.21/24 brd 192.168.16.255 scope global eth0
    inet6 fe80::20c:29ff:fe49:e40b/64 scope link
        valid_lft forever preferred_lft forever
[root@localhost ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 192.168.16.20:80 rr persistent 50
-> 192.168.16.2:80 Route 1 0 0
-> 192.168.16.3:80 Route 1 0 0
-> 192.168.16.4:80 Route 1 0 0
[root@localhost ~]#
```

在 backup 上面查看 web 主机



```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:e9:96:79 brd ff:ff:ff:ff:ff:ff
    inet 192.168.16.22/24 brd 192.168.16.255 scope global eth0
    inet6 fe80::20c:29ff:fee9:9679/64 scope link
        valid_lft forever preferred_lft forever
[root@localhost ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 192.168.16.20:80 rr persistent 50
-> 192.168.16.2:80 Route 1 0 0
-> 192.168.16.3:80 Route 1 0 0
-> 192.168.16.4:80 Route 1 0 0
[root@localhost ~]#
```

好了。以上对服务的设置已经完毕，下面我们对这个系统用拓扑图来描述



当用户访问 `www.itnihao.cn` 这个域名，通过 `dns` 查询到 `ip` 为 `202.11.2.8`，于是，用户请求 `202.11.2.8` 这个 `ip`，在 `202.11.2.8` 路由上面，有 `nat` 映射规则，将请求翻译到内部 `192.168.16.10`，经过交换机，在 `mac tables` 里面发现 `192.168.16.20` 在接口 `g0/0` 上面，于是将数据投递给 `master` 机器，`master` 接收请求，发现访问的是 `192.168.16.20:80` 根据已经设置的 `lvs` 调度算法，将请求发送给后端的 `realserver`，假如根据调度算法，发送给了 `192.168.16.2`，此时，`192.168.16.2` 发现请求的数据包目的地是 `192.168.16.20`，于是在回复的时候用 `lo:0` 接口 `192.168.16.20` 作为响应，此处涉及 `arp` 响应，此处略过，请参考博文附录 `arp` 的详解

## 六、一些故障排除

### 6.1 master 宕机的演示

停止 `master` 的 `keepalived` 服务

在 `master` 上面查看日志如下

```

[root@localhost keepalived]# tail -f /var/log/messages
Jun 14 17:53:25 localhost keepalived: Terminating on signal
Jun 14 17:53:25 localhost keepalived: Stopping Keepalived v1.2.2 (06/10,2012)
Jun 14 17:53:25 localhost keepalived_healthcheckers: Terminating Healthchecker child process on signal
Jun 14 17:53:25 localhost keepalived_vrrp: Terminating VRRP child process on signal
Jun 14 17:53:25 localhost keepalived_vrrp: VRRP_Instance(VI_1) removing protocol VIPs.

```

在 backup 上面查看，master 已经切换了

```

ssages
Jun 14 15:10:18 localhost keepalived_vrrp: VRRP_Instance(VI_1) Transition to MASTER STATE
Jun 14 15:10:19 localhost keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER STATE
Jun 14 15:10:19 localhost keepalived_vrrp: VRRP_Instance(VI_1) setting protocol VIPs.
Jun 14 15:10:19 localhost keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 192.168.16.20
Jun 14 15:10:19 localhost keepalived_healthcheckers: Netlink reflector reports IP 192.168.16.20 added
Jun 14 15:10:24 localhost keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 192.168.16.20

```

```

bash-2.03$ arp -a

Interface: 192.168.16.90 --- 0x10005
  Internet Address      Physical Address        Type
  192.168.16.1          00-22-aa-40-1a-c2      dynamic
  192.168.16.2          00-0c-29-1a-b4-71      dynamic
  192.168.16.3          00-0c-29-64-b4-49      dynamic
  192.168.16.4          00-0c-29-59-b3-47      dynamic
  192.168.16.20         00-0c-29-e9-96-79      dynamic
  192.168.16.21         00-0c-29-49-e4-0b      dynamic
  192.168.16.22         00-0c-29-e9-96-79      dynamic
bash-2.03$ curl 192.168.16.20
this is 0.2
bash-2.03$

```

此时访问依然 OK 的

在切换的瞬间，192.168.16.20 的网络会中断一下

```

Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Request timed out.
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64
Reply from 192.168.16.20: bytes=32 time<1ms TTL=64

```

恢复 master

在 master 日志里面可以看到已经恢复为 master 了

```

Jun 14 18:01:19 localhost keepalived_healthcheckers: Activating healthchecker for service [192.168.16.4]:80
Jun 14 18:01:20 localhost keepalived_vrrp: VRRP_Instance(VI_1) Transition to MASTER STATE
Jun 14 18:01:21 localhost keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER STATE
Jun 14 18:01:21 localhost keepalived_vrrp: VRRP_Instance(VI_1) setting protocol VIPs.
Jun 14 18:01:21 localhost keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 192.168.16.20
Jun 14 18:01:21 localhost keepalived_healthcheckers: Netlink reflector reports IP 192.168.16.20 added
Jun 14 18:01:26 localhost keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 192.168.16.20

```

Backup 里面看到恢复为 backup 的

```

Jun 14 15:18:10 localhost keepalived_vrrp: VRRP_Instance(VI_1) Received higher prio advert
Jun 14 15:18:10 localhost keepalived_vrrp: VRRP_Instance(VI_1) Entering BACKUP STATE
Jun 14 15:18:10 localhost keepalived_vrrp: VRRP_Instance(VI_1) removing protocol VIPs.
Jun 14 15:18:10 localhost keepalived_healthcheckers: Netlink reflector reports IP 192.168.16.20 removed

```

关闭其中的一台正在连接的服务器 192.168.16.2.

```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet 192.168.16.20/32 brd 192.168.16.20 scope global lo:0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 10
    link/ether 00:0c:29:1a:b4:71 brd ff:ff:ff:ff:ff:ff
    inet 192.168.16.2/24 brd 192.168.16.255 scope global eth0
[root@localhost ~]# service httpd stop
Stopping httpd:
[root@localhost ~]#
```

Master 上面看到 192.168.16.2 已经移除了

```
Jun 14 18:18:09 localhost keepalived_healthcheckers: SMTP alert successfully sent.
Jun 14 18:18:23 localhost keepalived_healthcheckers: TCP connection to [192.168.16.2]:80 failed !!!
Jun 14 18:18:23 localhost keepalived_healthcheckers: Removing service [192.168.16.2]:80 from VS [192.168.16.20]:80
Jun 14 18:18:23 localhost keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connected.
Jun 14 18:18:23 localhost keepalived_healthcheckers: SMTP alert successfully sent.
```

```
root@localhost ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
root LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward weight ActiveConn InActConn
TCP 192.168.16.20:80 rr persistent 50
-> 192.168.16.3:80 Route 1 0 0
-> 192.168.16.4:80 Route 1 1 0
```

Slave 上面看到 192.168.16.2 也移除了

```
Jun 14 15:35:00 localhost keepalived_healthcheckers: SMTP alert successfully sent.
Jun 14 15:35:11 localhost keepalived_healthcheckers: TCP connection to [192.168.16.2]:80 failed !!!
Jun 14 15:35:11 localhost keepalived_healthcheckers: Removing service [192.168.16.2]:80 from VS [192.168.16.20]:80
Jun 14 15:35:11 localhost keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connected.
Jun 14 15:35:11 localhost keepalived_healthcheckers: SMTP alert successfully sent.
```

```
[root@localhost ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward weight ActiveConn InActConn
TCP 192.168.16.20:80 rr persistent 50
-> 192.168.16.3:80 Route 1 0 0
-> 192.168.16.4:80 Route 1 0 0
```

可以看到服务已经自动切换了

```
this is 0.2
this is 0.2
this is 0.2
this is 0.2
this is 0.2
curl: (56) Recv failure: Connection was reset
curl: (7) couldn't connect to host
curl: (7) couldn't connect to host
this is 0.4
this is 0.4
this is 0.4
this is 0.4
this is 0.4
this is 0.4
```

注意：为什么这里一直访问后端的一台服务器，其实就是会话一致起的作用，在切换服务器的时候，连接会稍微中断一会儿，但不影响继续访问。如果此时的 session 存在于单台服务器上面，则用户的会话会丢失，需要重新建立连接(比如重新登录等动作)。

访问脚本如下

```
bash-2.03$ cat curl.sh
#!/bin/bash
while :;
do
curl 192.168.16.20
done
```

从以上故障切换实例可以看到，keepalive+lvs 组合可以实现秒级别的切换，可以满足高可用集群的需求，对于 session 一致性的实现，可以通过设置 `persistence_timeout` 参数达到目的。然而在服务器切换时如要实现会话的一致性，则需要通过其他方法来实现。

参数文档: <http://hi.baidu.com/gvc800/blog/item/88f486dfefd8fd096227984c.html>  
<http://wenku.baidu.com/view/85alc6lca8114431b90dd8b1.html>  
<http://zhumeng8337797.blog.163.com/blog/static/100768914201191762253640/>  
<http://bbs.ywlm.net/thread-790-1-1.html>  
《Keepalived 权威指南.》  
《keepalived the definitive guid》

文档暂写到此处, 有不足和错误之处, 欢迎指正, 谢谢!

附录: LVS 服务端脚本 (本案例中没有用到这个脚本)

```
#!/bin/sh
#
# lvs      Start lvs
#
# chkconfig: 2345 05 06
# description: Starts, stops and saves lvs
#

SNS_VIP=192.168.16.20
SNS_RIP1=192.168.16.2
SNS_RIP2=192.168.16.3
SNS_RIP2=192.168.16.4

. /etc/rc.d/init.d/functions

#logger $0 called with $1
retval=0

start()
{
    #set squid vip
    /sbin/ipvsadm --set 30 5 60
    /sbin/ifconfig eth0:0 $SNS_VIP netmask 255.255.255.255 broadcast $SNS_VIP up
    /sbin/route add -host $SNS_VIP dev eth0:0
    /sbin/ipvsadm -A -t $SNS_VIP:80 -s wlc
    /sbin/ipvsadm -a -t $SNS_VIP:80 -r $SNS_RIP1:80 -g -w 1
    /sbin/ipvsadm -a -t $SNS_VIP:80 -r $SNS_RIP2:80 -g -w 1
    /sbin/ipvsadm -a -t $SNS_VIP:80 -r $SNS_RIP3:80 -g -w 1
    touch /var/lock/subsys/ipvsadm > /dev/null 2 >&1
    echo "ipvsadm started"
}

stop()
```

```
{
    /sbin/ipvsadm -C
    /sbin/ipvsadm -Z
    ifconfig eth0:0 down
    route del $SNS_VIP
    rm -rf /var/lock/subsys/ipvsadm > /dev/null 2 >&1
    echo "ipvsadm stoped"
}

status()
{
    if [ ! -e /var/lock/subsys/ipvsadm ];then
        echo "ipvsadm stoped"
        exit 1
    else
        echo "ipvsadm OK"
    fi
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status
        ;;
    restart)
        stop
        start
        ;;
    *)
        echo $"Usage: $0 {start|stop|status}"
        retval=1
esac
exit $retval
```

Realserver 脚本，在以上的手动添加命令中，可以使用此脚本

```
#!/bin/bash
SNS_VIP=192.168.16.2
. /etc/rc.d/init.d/functions
```

```
case "$1" in
start)
ifconfig lo:0 $SNS_VIP netmask 255.255.255.255 broadcast $SNS_VIP
/sbin/route add -host $SNS_VIP dev lo:0
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
sysctl -p >/dev/null
echo "RealServer Start Ok"
;;
stop)
ifconfig lo:0 down
route del $LVS_VIP >/dev/null 2 >&1
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
echo "RealServer Stopped"
;;
*)
echo "Usage: $0 {start|stop}"
exit 1
esac
exit 0
```

## arp\_announce 和 arp\_ignore 详细解说

lvs 在 DR 模式下需要关闭 arp，设置参数的意思可以参考下文

arp\_announce 和 arp\_ignore

用来屏蔽 arp 请求，比较难理解，先看看 linux 核心2.6的定义：

```
=====arp_announce=====

arp_announce - INTEGER Define different restriction levels for announcing the
local source IP address from IP packets in ARP requests sent on interface:

0 - (default) Use any local address, configured on any interface
1 - Try to avoid local addresses that are not in the target's subnet for this
interface. This mode is useful when target hosts reachable via this interface
require the source IP address in ARP requests to be part of their logical network
configured on the receiving interface. When we generate the request we will check
all our subnets that include the target IP and will preserve the source address
if it is from such subnet. If there is no such subnet we select source address
according to the rules for level
```



2 - Always use the best local address for this target. In this mode we ignore the source address in the IP packet and try to select local address that we prefer for talks with the target host. Such local address is selected by looking for primary IP addresses on all our subnets on the outgoing interface that include the target IP address. If no suitable local address is found we select the first local address we have on the outgoing interface or on all other interfaces, with the hope we will receive reply for our request and even sometimes no matter the source IP address we announce. The max value from conf/{all,interface}/arp\_announce is used. Increasing the restriction level gives more chance for receiving answer from the resolved target while decreasing the level announces more valid sender's information.

#对网络接口上,本地 IP 地址的发出的,ARP 回应,作出相应级别的限制:确定不同程度的限制,宣布对来自本地源 IP 地址发出 Arp 请求的接口

0 - (默认)在任意网络接口(eth0,eth1,lo)上的任何本地地址

1 -尽量避免不在该网络接口子网段的本地地址做出 arp 回应.当发起 ARP 请求的源 IP 地址是被设置应该经由路由达到此网络接口的时候很有用.此时会检查来访 IP 是否为所有接口上的子网段内 ip 之一.如果改来访 IP 不属于各个网络接口上的子网段内,那么将采用级别2的方式来进行处理.

2 - 对查询目标使用最适当的本地地址.在此模式下将忽略这个 IP 数据包的源地址并尝试选择与能与该地址通信的本地地址.首要的是选择所有的网络接口的子网中外出访问子网中包含该目标 IP 地址的本地地址.如果没有合适的地址被发现,将选择当前的发送网络接口或其他的可能接受到该 ARP 回应的网络接口来进行发送.

=====arp\_ignore=====

arp\_ignore - INTEGER Define different modes for sending replies in response to received ARP requests that resolve local target IP addresses:

0 - (default): reply for any local target IP address, configured on any interface

1 - reply only if the target IP address is local address configured on the incoming interface

2 - reply only if the target IP address is local address configured on the incoming interface and both with the sender's IP address are part from same subnet on this interface

3 - do not reply for local addresses configured with scope host,only resolutions for global and link addresses are replied

4-7 - reserved

8 - do not reply for all local addresses

The max value from conf/{all,interface}/arp\_ignore is used when ARP request is received on the {interface}

定义对目标地址为本地 IP 的 ARP 询问不同的应答模式0

0 - (默认值): 回应任何网络接口上对任何本地 IP 地址的 arp 查询请求

1 - 只回答目标 IP 地址是来访网络接口本地地址的 ARP 查询请求

2 -只回答目标 IP 地址是来访网络接口本地地址的 ARP 查询请求,且来访 IP 必须在该网络接口的子网段内

3 - 不回答网络界面的 arp 请求,而只对设置的唯一和连接地址做出回应

4-7 - 保留未使用

8 -不回应所有(本地地址)的 arp 查询

arp\_ignore 设置为1,这个比较好理解,当别人的 arp 请求过来的时候,如果接收的设备上面没有这个

ip, 就不响应, 默认是0, 只要这台机器上面任何一个设备上面有这个 ip, 就响应 arp 请求, 并发送 mac 地址应答。

arp\_announce 这个就比较难解释了, 先看一段英文的:

Assume that a linux box X has three interfaces - eth0, eth1 and eth2. Each interface has an IP address IP0, IP1 and IP2. When a local application tries to send an IP packet with IP0 through the eth2. Unfortunately, the target node's mac address is not resolved. The linux box X will send the ARP request to know the mac address of the target (or the gateway). In this case what is the IP source address of the "ARP request message"? The IP0 - the IP source address of the transmitting IP or IP2 - the outgoing interface? Until now (actually just 3 hours before) ARP request uses the IP address assigned to the outgoing interface (IP2 in the above example). However the linux's behavior is a little bit different. Actually the selection of source address in ARP request is totally configurable by the proc variable "arp\_announce"

If we want to use the IP2 not the IP0 in the ARP request, we should change the value to 1 or 2. The default value is 0 - allow IP0 is used for ARP request.

其实就是路由器的的问题, 因为路由器一般是动态学习 ARP 包的 (一般动态配置 DHCP 的话), 当内网的机器要发送一个到外部的 ip 包, 那么它就会请求 路由器的 Mac 地址, 发送一个 arp 请求, 这个 arp 请求里面包括了自己的 ip 地址和 Mac 地址, 而 linux 默认是使用 ip 的源 ip 地址作为 arp 里面的 源 ip 地址, 而不是使用发送设备上面的, 这样在 lvs 这样的架构下, 所有发送包都是同一个 VIP 地址, 那么 arp 请求就会包括 VIP 地址和设备 Mac, 而路由器收到这个 arp 请求就会更新自己的 arp 缓存, 这样就会造成 ip 欺骗了, VIP 被抢夺, 所以就会有问题。

arp 缓存为什么会更新了, 什么时候会更新呢, 为了减少 arp 请求的次数, 当主机接收到询问自己的 arp 请求的时候, 就会把源 ip 和源 Mac 放入自己的 arp 表里面, 方便接下来的通讯。如果收到不是询问自己的包 (arp 是广播的, 所有人都收到), 就会丢掉, 这样不会造成 arp 表里面无用数据太多导致 有用的记录被删除。

What happens when a host receives an ARP request packet? The ARP request is received and processed by all the hosts in the network, since it is a broadcast packet. The following steps are carried out when a ARP request packet is received by a host: If the IP address to be resolved is for this host, then the ARP module sends an ARP reply packet with its Ethernet MAC address. If the IP address to be resolved is for this host, then the ARP module updates its ARP cache with the source Ethernet MAC address to source IP address mapping present in the ARP request packet. If the entry is already present in the cache, it is overwritten. If it is not present, it is added. If the IP address to be resolved is not for this host, then the ARP module discards the ARP request packet. Will a host update its ARP cache upon receiving any ARP request? A host will update its ARP cache, only if the ARP request is for its IP address. Otherwise, it will discard the ARP request. What is the disadvantage if a host updates its ARP cache upon receiving any ARP request? The host will exhaust the ARP cache with a lot of unused ARP entries, if it updates the ARP cache for any ARP request.

如果路由器使用静态 ARP 表, 客户端也使用静态网关 ARP 的话, 基本就不用管这两个值了。也看了一下 vpn, 用了 pppoe 协议, 也是不需要处理 arp 请求的。

查看某个设备上面绑定了多少个 ip: >ip addr show dev eth0

绑定多个 ip (临时, 看操作系统不同加在不同的地方): >ip addr add x.x.x.x/32 dev eth0

临时修改 arp\_announce 和 arp\_ignore:

```
echo 1 > /proc/sys/net/ipv4/conf/eth0/arp_ignore
```

```
echo 2 > /proc/sys/net/ipv4/conf/eth0/arp_announce
```

永久修改:

if /etc/sysctl.conf is used in the system, we have this config in /etc/sysctl.conf

```
net.ipv4.conf.eth0.arp_ignore = 1
```

```
net.ipv4.conf.eth0.arp_announce = 2
```

在 lvs 环境中, 需要设定以下的参数

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

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

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

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

参考文档

[http://blog.sina.com.cn/s/blog\\_6caddb500100qp5v.html](http://blog.sina.com.cn/s/blog_6caddb500100qp5v.html)

<http://hi.baidu.com/li32768/blog/item/83bb13cb803be198c81768d3.html>