Keepalived双机热备

# Keepalived介绍

Keepalived是linux下一个轻量级高可用的解决方案，它和HeartBeat实现的功能一样，都可以实现服务或者网络的高可用性。区别在于HeartBeat是一个专业的，功能完善的高可用软件，提供了HA软件的基本功能，比如心跳检测盒资源接管，监控集群中的系统服务。与Heartbeat相比，Keepalived相对比较简单，容易配置。Keepalive主要是通过虚拟路由冗余来实现高可用。

Keepalived起初是为LVS设计的。专门用来监控集群系统服各个服务节点的状态。根据TCP/IP参考模型的第三，第四，和第五层交换机制检测每个服务节点的状态。后来，Keepalived又加入了VRRP(虚拟路由冗余协议)功能。用于解决静态路由出现单点故障问题。

## VRRP协议和工作原理

VRRP可以将两台或者多台物理路由器设备虚拟成一个虚拟路由器，这个虚拟路由器通过虚拟ip对外进行服务。在虚拟路由器的内部有多个物理路由器协同工作，同一时间只有一个物理路由器对外进行服务，这台物理服务器称为主路由器（master）。当主路由器失效后，BACKUP角色的备份路由器重新进行选举，产生新的主路由器对外进行服务。

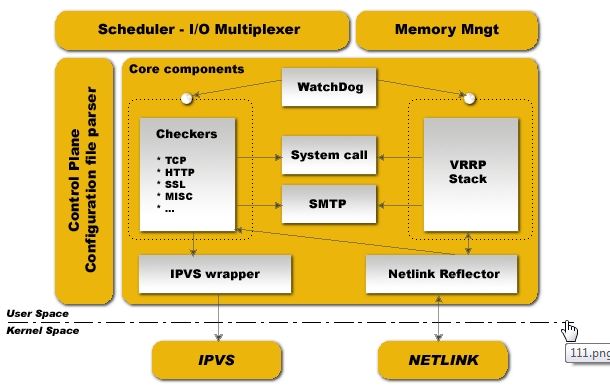
每个虚拟路由器都有一个唯一标识，称为VRID。一个VRID与一组ip地址构成一个虚拟路由器。在VRRP协议中，所有的报文都是通过ip多播的形式发送的，而在虚拟路由器中，只有处于MASTER角色的路由器会一直发送VRRP数据包，处于BACKUP角色的路由器只接受MASTER发送过来的报文信息，用来监控MASTER运行状态。

## Keepalived工作原理

Keepalived工作在TCP/IP模型的第三，第四和第五层（网络层，传输层，应用层）。Keepalived运行机制如下：

* 网络层：通过发送icmp报文来判断节点是否失效。
* 传输层：利用TCP协议的端口连接和扫描技术来判断集群节点是否正常。
* 应用层：用户自定义Keepalived来判断。

## Keepalived的体系结构



从图中可以看出，Keepalived的体系结构从整体上分为两层，分别是用户空间层（User Space）和内核空间层（Kernel Space）.下面介绍Keepalived两层结构的详细组成及实现的功能。

### 内核空间层

内核空间层处于最底层，它包括IPVS和NETLINK两个模块。IPVS模块是Keepalived引入的一个第三方模块，通过IPVS可以实现基于IP的负载均衡集群。IPVS默认包含在LVS集群软件中。而对于LVS集群软件，相信做运维的朋友并不陌生：在LVS集群中，IPVS安装在一个叫做Director Server的服务器上，同时在Director Server上虚拟出一个IP地址来对外提供服务，而用户必须通过这个虚拟IP地址才能访问服务。这个虚拟IP一般称为LVS的VIP，即Virtual IP。访问的请求首先经过VIP到达Director Server，然后由Director Server从服务器集群节点中选取一个服务节点响应用户的请求。

Keepalived最初就是为LVS提供服务的，由于Keepalived可以实现对集群节点的状态检测，而IPVS可以实现负载均衡功能，因此，Keepalived借助于第三方模块IPVS就可以很方便地搭建一套负载均衡系统。在这里有个误区，由于Keepalived可以和IPVS一起很好地工作，因此很多初学者都以为Keepalived就是一个负载均衡软件，这种理解是错误的。

NETLINK模块主要用于实现一些高级路由框架和一些相关的网络功能，完成用户空间层Netlink Reflector模块发来的各种网络请求。

### 用户空间层

用户空间层位于内核空间层之上，Keepalived的所有具体功能都在这里实现。用户空间层，Keepalived又分为四个部分，分别是Scheduler I/O Multiplexer、Memory Management、Control Plane和Core components。其中，Scheduler I/O Multiplexer是一个I/O复用分发调度器，它负责安排Keepalived所有内部的任务请求。Memory Management是一个内存管理机制，这个框架提供了访问内存的一些通用方法。Control Plane是Keepalived的控制面板，可以实现对配置文件进行编译和解析，Keepalived的配置文件解析比较特殊，它并不是一次解析所有模块的配置，而是只有在用到某模块时才解析相应的配置。最后详细说一下Core components，这个部分是Keepalived的核心组件，包含了一些列功能模块，主要有WatchDog、Checkers、VRRP Stack、IPVS wrapper和Netlink Reflector，下面介绍每个模块所实现的功能如下：

WatchDog是计算机可靠性领域中一个极为简单又非常有效的检测工具，它的工作原理是针对被监视的目标设置一个计数器和一个阈值，WatchDog会自己增加此计数值，然后等待被监视的目标周期性地重置该计数值。一旦被监控目标发生错误，就无法重置此计数值，WatchDog就会检测到，于是就采取对应的恢复措施，例如重启或关闭。在Linux中很早就引入了WatchDog功能，而Keepalived正是通过WatchDog的运行机制来监控Checkers和VRRP进程的。

Checkers：这是Keepalived最基础的功能，也是最主要的功能，可实现对服务器运行状态检测和故障隔离。

VRRP Stack：这是Keepalived后来引入的VRRP功能，可以实现HA集群中失败切换（Failover）功能。Keepalived通过VRRP功能再结合LVS负载均衡软件即可部署一套高性能的负载均衡集群系统。

IPVS wrapper：这是IPVS功能的一个实现。IPVS wrapper模块可以将设置好的IPVS规则发送到内核空间并提交给IPVS模块，最终实现IPVS模块的负载均衡功能。

Netlink Reflector：用来实现高可用集群中Failover时虚拟IP（VIP）的设置和切换。Netlink Reflector的所有请求最后都发送到内核空间的NETLINK模块来完成。

# Keepalived安装和配置

下载地址：<http://www.keepalived.org>

## Keepalived安装过程

|  |
| --- |
| tar zxvf keepalived-1.2.19.tar.gz  cd keepalived-1.2.19  uname –r #--with-kernel-dir只有在使用LVS的时候才需要  ./configure --sysconf=/etc --with-kernel-dir=/usr/src/kernels/2.6.32-358.el6.x86\_64    make && make install  ln -s /usr/local/sbin/keepalived /sbin/  chkconfig --add keepalived  chkconfig --level 35 keepalived on |

## Keepalived全局配置

指定了Keepalived配置文件的路径为/etc/keepalived/keepalived.conf，Keepalived的所有配置均在这个配置文件中完成。由于Keepalived.conf文件中可配置的选项比较多，这里根据配置文件所实现的功能，将Keepalived配置分为三类，分别是：全局配置(Global Configuration)、VRRPD配置和LVS配置。

Keepalived的配置文件都是以块（block）的形式组织的，每个块的内容都包含在{}中，以“#”和“!”开头的行都是注释。全局配置就是对整个Keepalived都生效的配置，基本内容如下：

|  |
| --- |
| ! Configuration File for keepalived  global\_defs {  notification\_email {  dba.gao@gmail.com  ixdba@163.com  }  notification\_email\_from Keepalived@localhost  smtp\_server 192.168.200.1  smtp\_connect\_timeout 30  router\_id LVS\_DEVEL  } |

全局配置以“global\_defs”作为标识，在“global\_defs”区域内的都是全局配置选项，其中：

notification\_email：用于设置报警邮件地址，可以设置多个，每行一个。注意，如果要开启邮件报警，需要开启本机的Sendmail服务。

notification\_email\_from：用于设置邮件的发送地址。

smtp\_server用于设置邮件的smtp server地址。

smtp\_connect\_timeout：用于设置连接smtp server的超时时间。

router\_id：表示运行Keepalived服务器的一个标识，是发邮件时显示在邮件主题中的信息。

## Keepalived的VRRPD配置

VRRPD配置是Keepalived所有配置的核心，主要用来实现Keepalived的高可用功能。从结构上来看，VRRPD配置又可分为VRRP同步组配置和VRRP实例配置。

同步组：就是将所有VRRP实例都加入到同步组中，这样任何一个实例出现问题，都会导致Keepalived进行主备切换。

|  |
| --- |
| vrrp\_sync\_group G1 {  group {  VI\_1  VI\_2  VI\_5  }  notify\_backup "/usr/local/bin/vrrp.back arg1 arg2"  notify\_master "/usr/local/bin/vrrp.mast arg1 arg2"  notify\_fault "/usr/local/bin/vrrp.fault arg1 arg2"  }  vrrp\_sync\_group G2 {  group {  VI\_3  VI\_4  }  }  其中，G1同步组包含VI\_1、VI\_2、VI\_5三个VRRP实例，G2同步组包含VI\_3、VI\_4两个VRRP实例。这五个实例将在vrrp\_instance段进行定义。另外，在vrrp\_sync\_group段中还出现了notify\_master、notify\_backup、notify\_fault和notify\_stop四个选项，这是Keepalived配置中的一个通知机制，也是Keepalived包含的四种状态。下面介绍每个选项的含义：   * notify\_master：指定当Keepalived进入Master状态时要执行的脚本，这个脚本可以是一个状态报警脚本，也可以是一个服务管理脚本。Keepalived允许脚本传入参数，因此灵活性很强。 * notify\_backup：指定当Keepalived进入Backup状态时要执行的脚本，同理，这个脚本可以是一个状态报警脚本，也可以是一个服务管理脚本。 * notify\_fault：指定当Keepalived进入Fault状态时要执行的脚本，脚本功能与前两个类似。 * notify\_stop：指定当Keepalived程序终止时需要执行的脚本。 |

VRRP实例的配置：也就是配置Keepalived的高可用功能。

|  |
| --- |
| vrrp\_instance VI\_1 {  state MASTER  interface eth0  virtual\_router\_id 51  priority 100  advert\_int 1  mcast\_src\_ip <IPADDR>  garp\_master\_delay 10    track\_interface {  eth0  eth1  }  authentication {  auth\_type PASS  auth\_pass qwaszx  }  virtual\_ipaddress {  #<IPADDR>/<MASK> brd <IPADDR> dev <STRING> scope <SCOPT> label <LABEL>  192.168.200.16  192.168.200.17 dev eth1  192.168.200.18 dev eth2  }  virtual\_routes {  #src <IPADDR> [to] <IPADDR>/<MASK> via|gw <IPADDR> dev <STRING> scope <SCOPE>  src 192.168.100.1 to 192.168.109.0/24 via 192.168.200.254 dev eth1  192.168.110.0/24 via 192.168.200.254 dev eth1  192.168.111.0/24 dev eth2  192.168.112.0/24 via 192.168.100.254  192.168.113.0/24 via 192.168.100.252 or 192.168.100.253  }  nopreempt  preemtp\_delay 300  }  配置详解：   * vrrp\_instance：是VRRP实例开始的标识，后跟VRRP实例名称。 * state：用于指定Keepalived的角色，MASTER表示此主机是主服务器，BACKUP表示此主机是备用服务器。 * interface：用于指定HA监测网络的接口。 * virtual\_router\_id是虚拟路由标识，这个标识是一个数字，同一个vrrp实例使用唯一的标识，即在同一个vrrp\_instance下，MASTER和BACKUP必须是一致的。 * priority用于定义节点优先级，数字越大表示节点的优先级就越高。在一个vrrp\_instance下，MASTER的优先级必须大于BACKUP的优先级。 * advert\_int用于设定MASTER与BACKUP主机之间同步检查的时间间隔，单位是秒。 * mcast\_src\_ip用于设置发送多播包的地址，如果不设置，将使用绑定的网卡所对应的IP地址。 * garp\_master\_delay用于设定在切换到Master状态后延时进行Gratuitous arp请求的时间。 * track\_interface用于设置一些额外的网络监控接口，其中任何一个网络接口出现故障，Keepalived都会进入FAULT状态。 * authentication用于设定节点间通信验证类型和密码，验证类型主要有PASS和AH两种，在一个vrrp\_instance下，MASTER与BACKUP必须使用相同的密码才能正常通信。 * virtual\_ipaddress用于设置虚拟IP地址（VIP），又叫做漂移IP地址。可以设置多个虚拟IP地址，每行一个。之所以称为漂移IP地址，是因为Keepalived切换到Master状态时，这个IP地址会自动添加到系统中，而切换到BACKUP状态时，这些IP又会自动从系统中删除。Keepalived通过“ip address add”命令的形式将VIP添加进系统中。要查看系统中添加的VIP地址，可以通过“ip add”命令实现。“virtual\_ipaddress”段中添加的IP形式可以多种多样，例如可以写成 “192.168.16.189/24 dev eth1” 这样的形式，而Keepalived会使用IP命令“ip addr add 192.168.16.189/24 dev eth1”将IP信息添加到系统中。因此，这里的配置规则和IP命令的使用规则是一致的。 * virtual\_routes和virtual\_ipaddress段一样，用来设置在切换时添加或删除相关路由信息。使用方法和例子可以参考上面的示例。通过“ip route”命令可以查看路由信息是否添加成功，此外，也可以通过上面介绍的notify\_master选项来代替virtual\_routes实现相同的功能。 * nopreempt设置的是高可用集群中的不抢占功能。在一个HA Cluster中，如果主节点死机了，备用节点会进行接管，主节点再次正常启动后一般会自动接管服务。这种来回切换的操作，对于实时性和稳定性要求不高的业务系统来说，还是可以接受的，而对于稳定性和实时性要求很高的业务系统来说，不建议来回切换，毕竟服务的切换存在一定的风险和不稳定性，在这种情况下，就需要设置nopreempt这个选项了。设置nopreempt可以实现主节点故障恢复后不再切回到主节点，让服务一直在备用节点工作，直到备用节点出现故障才会进行切换。在使用不抢占时，只能在“state”状态为“BACKUP”的节点上设置，而且这个节点的优先级必须高于其他节点。 * preemtp\_delay用于设置抢占的延时时间，单位是秒。有时候系统启动或重启之后网络需要经过一段时间才能正常工作，在这种情况下进行发生主备切换是没必要的，此选项就是用来设置这种情况发生的时间间隔。在此时间内发生的故障将不会进行切换，而如果超过“preemtp\_delay”指定的时间，并且网络状态异常，那么才开始进行主备切换。 |

## Keepalived的LVS配置

由于Keepalived属于LVS的扩展项目，因此， Keepalived可以与LVS无缝整合，轻松搭建一套高性能的负载均衡集群系统。LVS段的配置以“virtual\_server”作为开始标识，此段内容有两部分组成，分别是real\_server段和健康检测段。

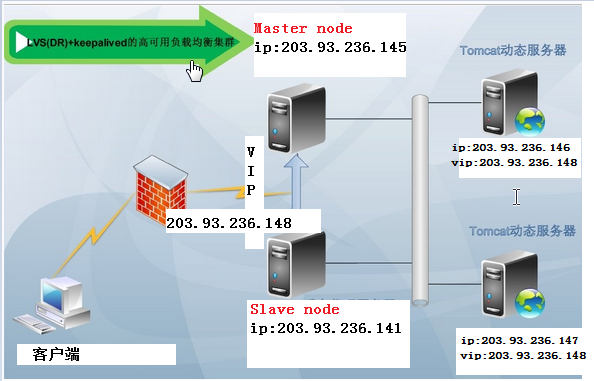
real\_server配置：

|  |
| --- |
| virtual\_server 192.168.12.200 80 {  delay\_loop 6  lb\_algo rr  lb\_kind DR  persistence\_timeout 50  persistence\_granularity <NETMASK>  protocol TCP  ha\_suspend  virtualhost <string>  sorry\_server <IPADDR> <PORT>  }  配置解析：   * virtual\_server：设置虚拟服务器的开始，后面跟虚拟IP地址和服务端口，IP与端口之间用空格隔开。 * delay\_loop：设置健康检查的时间间隔，单位是秒。 * lb\_algo：设置负载调度算法，可用的调度算法有rr、wrr、lc、wlc、lblc、sh、dh等，常用的算法有rr和wlc。 * lb\_kind：设置LVS实现负载均衡的机制，有NAT、TUN和DR三个模式可选。 * persistence\_timeout：会话保持时间，单位是秒。这个选项对动态网页是非常有用的，为集群系统中的session共享提供了一个很好的解决方案。有了这个会话保持功能，用户的请求会一直分发到某个服务节点，直到超过这个会话的保持时间。需要注意的是，这个会话保持时间是最大无响应超时时间，也就是说，用户在操作动态页面时，如果在50秒内没有执行任何操作，那么接下来的操作会被分发到另外的节点，但是如果用户一直在操作动态页面，则不受50秒的时间限制。 * persistence\_granularity：此选项是配合persistence\_timeout的，后面跟的值是子网掩码，表示持久连接的粒度。默认是255.255.255.255，也就是一个单独的客户端IP。如果将掩码修改为255.255.255.0，那么客户端IP所在的整个网段的请求都会分配到同一个real server上。 * protocol：指定转发协议类型，有TCP和UDP两种可选。 * ha\_suspend：节点状态从Master到Backup切换时，暂不启用real server节点的健康检查。 * virtualhost：在通过HTTP\_GET/ SSL\_GET做健康检测时，指定的Web服务器的虚拟主机地址。 * sorry\_server：相当于一个备用节点，在所有real server失效后，这个备用节点会启用。 * real\_server段开始的标识，用来指定real server节点   下面是real\_server段的一个配置示例：  real\_server 192.168.12.132 80 {  weight 3  inhibit\_on\_failure  notify\_up <STRING> | <QUOTED-STRING>  notify\_down <STRING> | <QUOTED-STRING>  } |
| weight：用来配置real server节点的权值。权值大小用数字表示，数字越大，权值越高。设置权值的大小可以为不同性能的服务器分配不同的负载，为性能高的服务器设置较高的权值，而为性能较低的服务器设置相对较低的权值，这样才能合理地利用和分配了系统资源。  inhibit\_on\_failure：表示在检测到real server节点失效后，把它的“weight”值设置为0，而不是从IPVS中删除。  notify\_up：此选项与上面介绍过的notify\_maser有相同的功能，后跟一个脚本，表示在检测到real server节点服务处于UP状态后执行的脚本。  notify\_down：表示在检测到real server节点服务处于DOWN状态后执行的脚本。  健康检测段允许多种检查方式，常见的有HTTP\_GET、SSL\_GET、TCP\_CHECK、SMTP\_CHECK、MISC\_CHECK。首先看TCP\_CHECK检测方式示例：  TCP\_CHECK {  connect\_port 80  connect\_timeout 3  nb\_get\_retry 3  delay\_before\_retry 3  }  下面介绍每个选项的含义介。  connect\_port：健康检查的端口，如果无指定，默认是real\_server指定的端口。  connect\_timeout：表示无响应超时时间，单位是秒，这里是3秒超时。  nb\_get\_retry：表示重试次数，这里是3次。  delay\_before\_retry：表示重试间隔，这里是间隔3秒。  下面是HTTP\_GET和SSL\_GET检测方式的示例：  HTTP\_GET |SSL\_GET  {  url {  path /index.html  digest e6c271eb5f017f280cf97ec2f51b02d3  status\_code 200  }  connect\_port 80  bindto 192.168.12.80  connect\_timeout 3  nb\_get\_retry 3  delay\_before\_retry 2  }  下面介绍每个选项的含义。  url：用来指定HTTP/SSL检查的URL信息，可以指定多个URL。  path：后跟详细的URL路径。  digest：SSL检查后的摘要信息，这些摘要信息可以通过genhash命令工具获取。例如：genhash -s 192.168.12.80 -p 80 -u /index.html。  status\_code：指定HTTP检查返回正常状态码的类型，一般是200。  bindto：表示通过此地址来发送请求对服务器进行健康检查。  下面是MISC\_CHECK检测方式的示例:  MISC\_CHECK  {  misc\_path /usr/local/bin/script.sh  misc\_timeout 5  ! misc\_dynamic  }  MISC健康检查方式可以通过执行一个外部程序来判断real server节点的服务状态，使用非常灵活。以下是常用的几个选项的含义。  misc\_path：用来指定一个外部程序或者一个脚本路径。  misc\_timeout：设定执行脚本的超时时间。  misc\_dynamic：表示是否启用动态调整real server节点权重，“!misc\_dynamic”表示不启用，相反则表示启用。在启用这功能后，Keepalived的healthchecker进程将通过退出状态码来动态调整real server节点的“weight”值，如果返回状态码为0，表示健康检查正常，real server节点权重保持不变；如果返回状态码为1，表示健康检查失败，那么就将real server节点权重设置为0；如果返回状态码为2~255之间任意数值，表示健康检查正常，但real server节点的权重将被设置为返回状态码减2，例如返回状态码为10，real server节点权重将被设置为8（10-2）。 |

注意：在配置Keepalived.conf时，需要特别注意配置文件的语法格式，因为Keepalived在启动时并不检测配置文件的正确性，即使没有配置文件，Keepalived也照样能够启动，所以一定要保证配置文件正确。

# Keepalived应用实例

## LVS+Keepalived实现负载均衡



第一步：网卡设置

第二步：LVS+Keepalived的安装

第三步：配置/etc/keepalived/keepalived.conf,对于备LVS机器只需要将MASTER改成SLAVE，优先级低于100即可。

|  |
| --- |
| ! Configuration File for keepalived  global\_defs {  notification\_email {  h861336327@163.com  }  notification\_email\_from sns-lvs@gmail.com  smtp\_server 127.0.0.1  router\_id LVS\_DEVEL  }  vrrp\_instance VI\_1 {  state MASTER  interface eth0  virtual\_router\_id 51  priority 100  advert\_int 1  authentication {  auth\_type PASS  auth\_pass 1111  }  virtual\_ipaddress {  203.93.236.148  }  }  virtual\_server 203.93.236.148 80 {  delay\_loop 6  lb\_algo wrr  lb\_kind DR  persistence\_timeout 60  protocol TCP  real\_server 203.93.236.146 80 {  weight 3  TCP\_CHECK {  connect\_timeout 10  nb\_get\_retry 3  delay\_before\_retry 3  connect\_port 80  }  }  real\_server 203.93.236.14780 {  weight 3  TCP\_CHECK {  connect\_timeout 10  nb\_get\_retry 3  delay\_before\_retry 3  connect\_port 80  }  }  } |

## 通过vrrp\_script实现对集群资源的监控

vrrp\_script模块专门用于对集群中服务资源进行监控。于此模块一起时的还有track\_script模块。在此模块中可以引入监控脚本实现对服务，端口等多方面的监控。track\_script模块主要用来调用vrrp\_script模块是Keepalived执行对集群服务资源的检测。

## Keepalived集群中MASTE和BACKUP角色的选取

在Keepalived集群中，其实并没有严格意义上的主、备节点，虽然可以在Keepalived配置文件中设置“state”选项为“MASTER”状态，但是这并不意味着此节点一直就是Master角色。控制节点角色的是Keepalived配置文件中的“priority”值，但并它并不控制所有节点的角色，另一个能改变节点角色的是在vrrp\_script模块中设置的“weight”值，这两个选项对应的都是一个整数值，其中“weight”值可以是个负整数，一个节点在集群中的角色就是通过这两个值的大小决定的。

在一个一主多备的Keepalived集群中，“priority”值最大的将成为集群中的Master节点，而其他都是Backup节点。在Master节点发生故障后，Backup节点之间将进行“民主选举”，通过对节点优先级值“priority”和““weight”的计算，选出新的Master节点接管集群服务。

在vrrp\_script模块中，如果不设置“weight”选项值，那么集群优先级的选择将由Keepalived配置文件中的“priority”值决定，而在需要对集群中优先级进行灵活控制时，可以通过在vrrp\_script模块中设置“weight”值来实现。下面列举一个实例来具体说明。

假定有A和B两节点组成的Keepalived集群，在A节点keepalived.conf文件中，设置“priority”值为100，而在B节点keepalived.conf文件中，设置“priority”值为80，并且A、B两个节点都使用了“vrrp\_script”模块来监控mysql服务，同时都设置“weight”值为10，那么将会发生如下情况。

在两节点都启动Keepalived服务后，正常情况是A节点将成为集群中的Master节点，而B自动成为Backup节点，此时将A节点的mysql服务关闭，通过查看日志发现，并没有出现B节点接管A节点的日志，B节点仍然处于Backup状态，而A节点依旧是Master状态，在这种情况下整个HA集群将失去意义。

下面就分析一下产生这种情况的原因，这也就是Keepalived集群中主、备角色选举策略的问题。下面总结了在Keepalived中使用vrrp\_script模块时整个集群角色的选举算法，由于“weight”值可以是正数也可以是负数，因此，要分两种情况进行说明。

1. “weight”值为正数时

在vrrp\_script中指定的脚本如果检测成功，那么Master节点的权值将是“weight值与”priority“值之和，如果脚本检测失败，那么Master节点的权值保持为“priority”值，因此切换策略为：

Master节点“vrrp\_script”脚本检测失败时：如果Master节点“priority”值小于Backup节点“weight值与”priority“值之和，将发生主、备切换。

Master节点“vrrp\_script”脚本检测成功时：如果Master节点“weight”值与“priority”值之和大于Backup节点“weight”值与“priority”值之和，主节点依然为主节点，不发生切换。

2. “weight”值为负数时

在“vrrp\_script”中指定的脚本如果检测成功，那么Master节点的权值仍为“priority”值，当脚本检测失败时，Master节点的权值将是“priority“值与“weight”值之差，因此切换策略为：

Master节点“vrrp\_script”脚本检测失败时，如果Master节点“priority”值与“weight”值之差小于Backup节点“priority”值，将发生主、备切换。

Master节点“vrrp\_script”脚本检测成功时，如果Master节点“priority”值大于Backup节点“priority”值时，主节点依然为主节点，不发生切换。

在熟悉了Keepalived主、备角色的选举策略后，再来分析一下刚才实例，由于A、B两个节点设置的“weight”值都为10，因此符合选举策略的第一种，在A节点停止Mysql服务后，A节点的脚本检测将失败，此时A节点的权值将保持为A节点上设置的“priority”值，即为100，而B节点的权值将变为“weight”值与“priority”值之和，也就是90（10+80），这样就出现了A节点权值仍然大于B节点权值的情况，因此不会发生主、备切换。

对于“weight”值的设置，有一个简单的标准，即“weight”值的绝对值要大于Master和Backup节点“priority”值之差。对于上面A、B两个节点的例子，只要设置“weight”值大于20即可保证集群正常运行和切换。由此可见，对于“weight值的设置，要非常谨慎，如果设置不好，将导致集群角色选举失败，使集群陷于瘫痪状态。

## Keepalived高可用说明

对于lvs的负载 均衡：Keepalived中编译时候添加lv\_vs的功能（负载均衡），并且Keepalived最初就是为了LVS服务的，具有对集群节点状态检测(网卡)。

对于应用服务器：Keepalived实现不了程序级别的高可用，需要我们通过shell脚本来实现。(track\_script模块主要用来调用vrrp\_script模块是Keepalived执行对集群服务资源的检测)。

**启动过程分析**：分别启动应用服务和Keepalived服务(会启动三个进程，父进程，vrrp子进程，healthcheckers子进程)。首先会运行track\_script(脚本)检查，根据用户节点类型，选举出master和backup。主节点的网卡接口绑定虚拟ip，当主节点失效时候，vip会移除原主节点。并出从节点中选举出主节点绑定vip。