HA 集群软件 Keepalived

相关术语

一、节点（ node）

运行 HA 进程的一个独立主机，称为节点，节点是 HA 的核心组成部分，每个节点上运行着操作系统和高可用软件服务，在高可用集群中，节点有主次之分，分别称为主节点和备用/备份节点，每个节点拥有唯一的主机名，并且拥有属于自己的一组资源，例如，磁盘、文件系统、网络地址和应用服务等。主节点上一般运行着一个或多个应用服务。而备用节点一般处于监控状态。

二、资源（ resource）

资源是一个节点可以控制的实体，并且当节点发生故障时，这些资源能够被其它节点接管，HA 集群软件中，可以当做资源的实体有：

- 磁盘分区、文件系统
- IP 地址 VIP
- 应用程序服务
- NFS 文件系统

三、事件（ event）

也就是集群中可能发生的事情，例如节点系统故障、网络连通故障、网卡故障、 应用程序故障等。这些事件都会导致节点的资源发生转移， HA 的测试也是基于这些事件来进行的。

四、动作（ action）

事件发生时 HA 的响应方式，动作是由 shell 脚步控制的，例如，当某个节点发生故障后，备份节点将通过事先设定好的执行脚本进行服务的关闭或启动。进而接管故障节点的资源。

Keepalived 简介

Keepalived 是 Linux 下一个轻量级的高可用解决方案，它与 HACMP、RoseHA 实现的功能类似，都可以实现服务或者网络的高可用，但是又有差别： HACMP 是一个专业的、教材地位分析功能完善的高可用软件，它提供了 HA 软件所需的基本功能，比如心跳检测和资源接管，监测集群中的系统服务，在群集节点间转移共享 IP 地址的所有者等， HACMP 功能强大，但是部署和使用相对比较麻烦，同时也是商业化软件；与 HACMP 相比， Keepalived 主要是通过虚拟路由冗余来实现高可用功能，虽然它没有 HACMP 功能强大，但 Keepalived 部署和使用非常简单，所有配置只需一个配置文件即可完成。

Keepalived 起初是为 LVS 设计的，专门用来监控集群系统中各个服务节点的状态。它根据 layer3, 4 & 5 交换机制检测每个服务节点的状态，如果某个服务节点出

现异常，工作出现故障， Keepalived 将检测到，并将出现故障的服务节点从集群系统中剔除，而在故障节点恢复正常后， Keepalived 又可以自动将此服务节点重新加入到服务器集群中，不需要人工干涉，需要人工完成的只是修复出现故障的服务节点。

Keepalived 后来又加入了 VRRP 的功能， VRRP 是 Virtual Router Redundancy Protocol（虚拟路由器冗余协议）的缩写，它出现的目的是为了解决静态路由出现的单点故障问题，通过 VRRP 可以实现网络不间断地、稳定地运行。因此， Keepalived 一方面具有服务器状态检测和故障隔离功能，另一方面也具有 HA cluster 功能.

VRRP 协议与工作原理

一、VRRP 协议

1、VRRP，全称虚拟路由冗余协议，VRRP 的出现是为了解决静态路由的单点故障。

2、VRRP 用 IP 多播的方式（默认多播地址 224.0.0.18）实现高可用对之间通信

3、VRRP 是通过一种竞选协议机制来将路由任务交给某台 VRRP 路由器的

4、工作时主节点发包，备节点接包，当备节点接收不到主节点发的数据包的时候，就启动接管程序接管主节点的资源，备节点可以有多个，通过优先级竞选

5、VRRP 使用了加密协议加密数据，但 keepalived 官方目前还是推荐使用明文方式配置认证类型和密码

二、工作原理

在现实的网络环境中，主机之间的通信都是通过配置静态路由（默认网关）完成的，而主机之间的路由器一旦出现故障，通信就会失败，因此，在这种通信模式中，路由器就成了一个单点瓶颈，为了解决这个问题，就引入了 VRRP 协议。

它是一种主备模式的协议，通过 VRRP 可以在网络发生故障时透明地进行设备切换而不影响主机间的数据通信，这其中涉及两个概念：物理路由器和虚拟路由器。

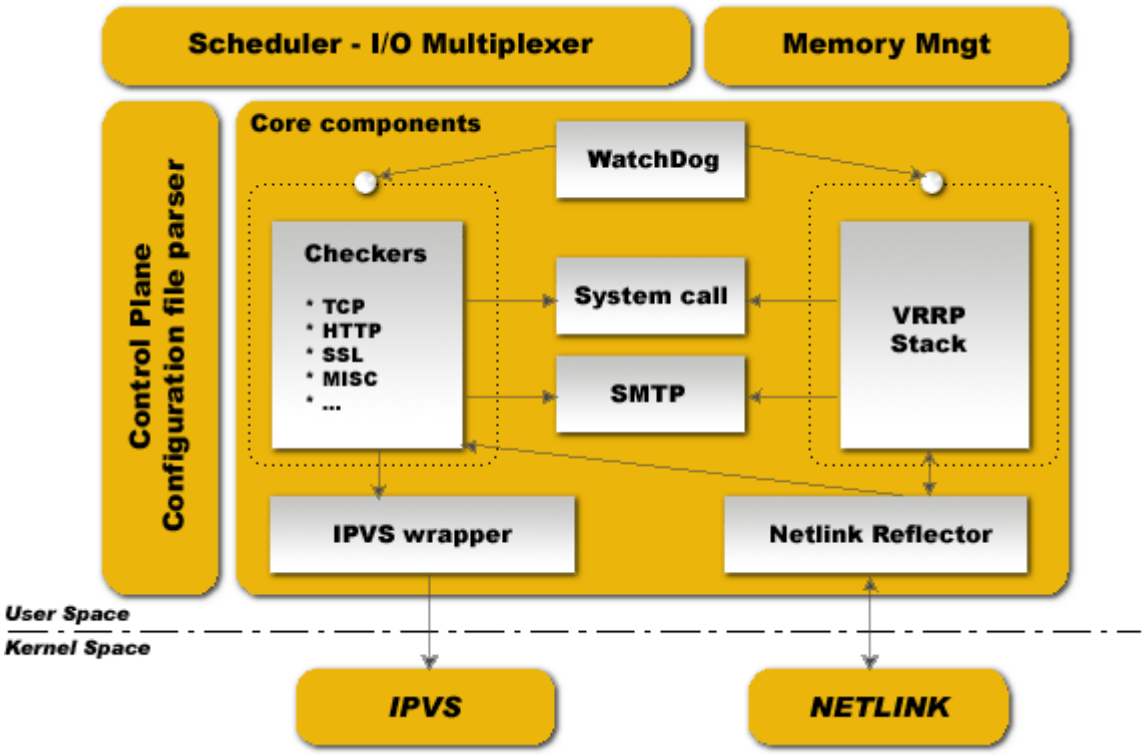
VRRP 可以将两台或多台物理路由器设备虚拟成一个虚拟路由器，这个虚拟路由器通过虚拟 IP（一个或多个）对外提供服务，而在虚拟路由器内部，是多个物理路由器协同工作，同一时间只有一台物理路由器对外提供服务，这台物理路由器被称为主路由器（处于
MASTER 角色）。一般情况下 MASTER 由选举算法产生，它拥有对外服务的虚拟 IP，提供各种网络功能，如 ARP 请求、 ICMP、数据转发等。而其他物理路由器不拥有对外的虚拟 IP，也不提供对外网络功能，仅仅接收 MASTER 的 VRRP 状态通告信息，这些路由器被统称为备份路由器（处于 BACKUP 角色）。当主路由器失效时，处于 BACKUP 角色的备份路由器将重新进行选举，产生一个新的主路由器进入 MASTER 角色继续提供对外服务，整个切换过程对用户来说完全透明。

在一个虚拟路由器中，只有处于 MASTER 角色的路由器会一直发送 VRRP 数据包，处于 BACKUP 角色的路由器只接收 MASTER 发过来的报文信息，用来监控 MASTER 运

行状态，因此，不会发生 MASTER 抢占的现象，除非它的优先级更高。而当 MASTER 不可用时，BACKUP 也就无法收到 MASTER 发过来的报文信息，于是就认定 MASTER 出现故障，接着多台 BACKUP 就会进行选举，优先级最高的 BACKUP 将成为新的 MASTER，因而也就保证了服务的持续可用性。

Keepalived 的体系结构

Keepalived 是一个高度模块化的软件，结构简单，但扩展性很强。



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。

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。

在 Keepalived 中， IPVS 模块是可配置的，如果需要负载均衡功能，可以在编译 Keepalived 时打开负载均衡功能，反之，也可以通过配置编译参数关闭。

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。

用户空间层位于内核空间层之上， Keepalived 的所有具体功能都在这里实现.

Keepalived 安装与配置

安装

建议通过 yum 方式直接安装：yum install keepalived

#如果需要 lvs 功能，还需要安装 ipvs 模块：yum install ipvsadm

配置

Keepalived 配置文件的默认路径为/etc/Keepalived/Keepalived.conf， Keepalived 的所有配置均在这个配置文件中完成。

Keepalived 配置分为三类，分别是：

- 全局配置(Global Configuration)
- VRRPD 配置
- LVS 配置


keepalived.conf

一、全局配置

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。

二、VRRPD 配置（核心配置）

VRRP 配置以“ vrrp_instance”作为标识，在这个实例中包含了若干配置选项，

分别介绍如下：
❑ 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。

❑ 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。

❑ 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 用于设置切换的延时时间，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。

❑ 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

三、LVS 配置

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。

❑ 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。

❑ 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：是 real_server 段开始的标识，用来指定 real server 节点，后面跟的是 real server 的真实 IP 地址和端口， IP 与端口之间用空格隔开。

❑ 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。

❑ 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。

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：

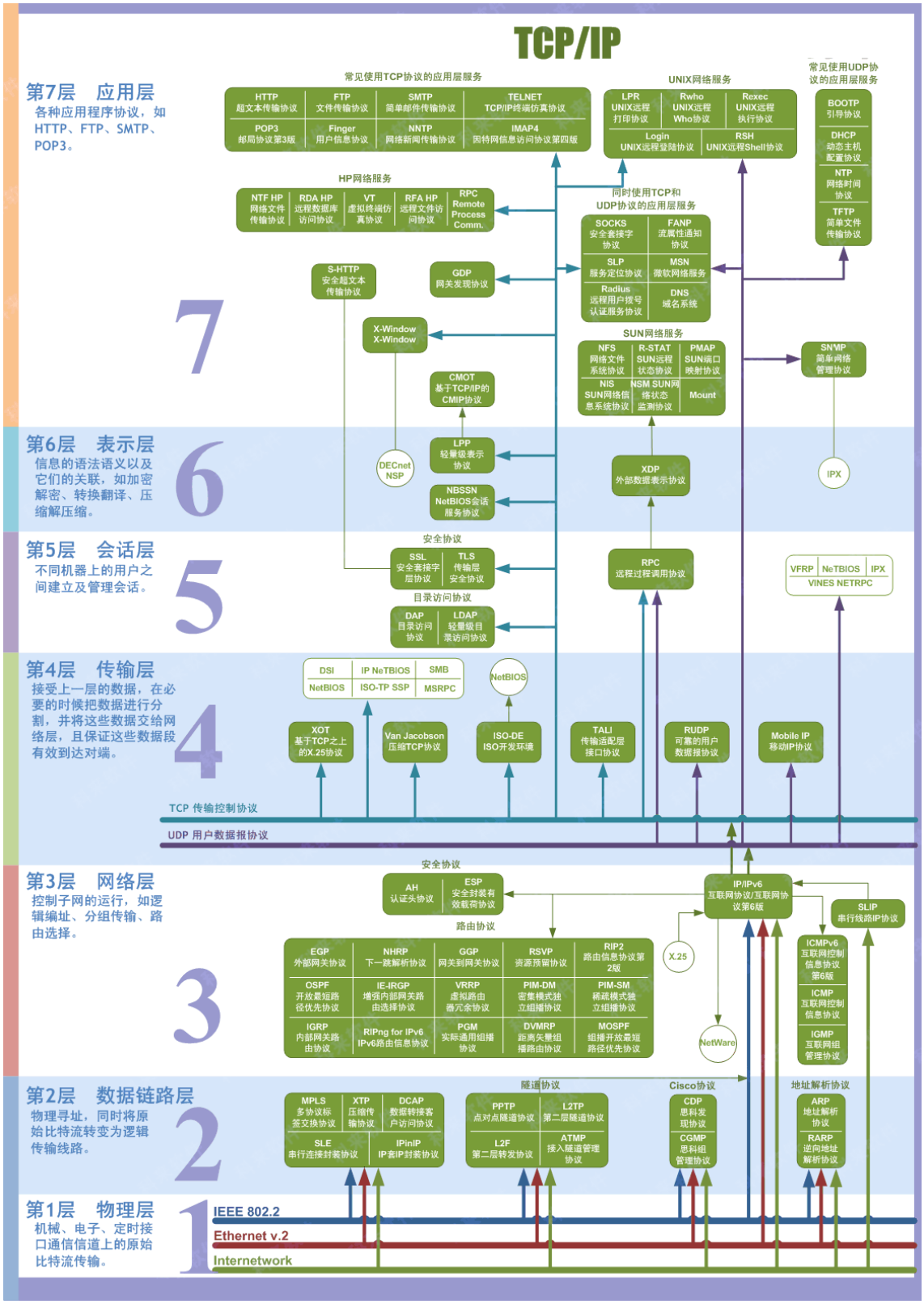
```
MISC_CHECK
{ misc_path /usr/local/bin/script.sh
misc_timeout 5
! misc_dynamic
}
```

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）。到这里为止， Keepalived 配置文件中常用的选项已经介绍完毕，在默认情况下，Keepalived 在启动时会查找/etc/Keepalived/Keepalived.conf 配置文件，如果配置文件放在其他路径下， 通过“ Keepalived -f”参数指定配置文件的路径即可。

Keepalived 提供了 vrrp_script、 notify_master、 notify_backup 等多个功能模块，通过这些模块可以实现对集群资源的托管以及集群服务的监控。

lvs 和 nginx 的对比



一、lvs 的优势：

1.抗负载能力强，因为 lvs 工作方式的逻辑是非常简单的，而且工作再网络层第 4 层，仅作请求分发用，没有流量，所以在效率上基本不需要太过考虑。lvs 一般很少出现故障，即使出现故障一般也是其他地方（如内存、CPU 等）出现问题导致 lvs 出现问题。

2.配置性地，这通常是一大劣势同时也是一大优势，因为没有太多的可配置的选项，所以除了增减服务器，并不需要经常去触碰它，大大减少了人为出错的几率。

3.工作稳定，因为其本省抗负载能力很强，所以稳定性高也是顺理成章的事，另外各种 lvs 都有完整的双机热备方案，所以一点不用担心均衡器本身会出什么问题，节点出现故障的话，lvs 会自动判别，所以系统整体式非常稳定的。

4.无流量，lvs 仅仅分发请求，而流量并不从它本身出去，所以可以利用它这点来做一些线路分流之用。没有流量同时也保住了均衡器的 IO 性能不会受到大流量的影响。

5.lvs 基本上能支持所有应用，因为绿色工作在第 4 层，所以它可以对几乎所有应用做负载均衡，包括 http、数据库、聊天室等。

另外：lvs 也不是完全能判别节点故障的，比如在 wlc 分配方式下，集群里有一个节点没有配置 vip，会使整个集群不能使用，这时使用 wrr 分配方式则会丢掉一台机器。目前这个问题还在进一步测试中。所以用 lvs 也得多多当心为妙。

二、nginx 和 lvs 作对比的结果：

1.nginx 工作在网络的第 7 层，所以它可以针对 http 应用本身来做分流策略，比如针对域名、目录结构等，相比之下 lvs 并不具备这样的功能，所以 nginx 单凭这点可以利用的场合就远多于 lvs 了；但 nginx 有用的这些功能使其可调整度要高于 lvs，所以经常要去触碰触碰，由 lvs 的第 2 条优点来看，触碰多了，人为出现问题的几率也就会大。

2.nginx 对网络的依赖较小，理论上只要 ping 得通，网页访问正常，nginx 就能连得通，nginx 同时还能区分内外网，如果是同时拥有内外网的节点，就相当于单机拥有了备份线路；lvs 就比较依赖于网络环境，目前来看服务器在同一网段内并且 lvs 使用 direct 方式分流，效果较能得到保证。另外注意，lvs 需要向托管商至少申请多于一个 ip 来做 visual ip，貌似是不能用本省的 ip 来做 VIP 的。要做好 lvs 管理员，确实得跟进学习很多有关网络通信方面的知识，就不再是一个 http 那么简单了。

3.nginx 安装和配置比较简单，测试起来也很方便，因为它基本能把错误用日志打印出来。lvs 的安装和配置、测试就要花比较长的时间，因为同上所述，lvs 对网络依赖性比较大，很多时候不能配置成功都是因为网络问题而不是配置问题，出了问题要解决也相应的会麻烦的多。

4.nginx 也同样能承受很高负载且稳定，但负载度很稳定度差 lvs 还有几个等级：nginx 处理所有流量所以受限于机器 IO 和配置；本身的 bug 也还是难以避免的；nginx 没有现成的双机热备方案，所以跑在单机上还是风险比较大，单机上的事情全都很难说。

5.nginx 可以检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点。目前 lvs 中 ldirectd 也能支持针对服务器内部的情况来监控，但 lvs 的原理使其不能重发请求。重发请求这点，比如用户正在上传一个文件，而处理该上传的节点刚好在上传过程中出现故障，nginx 会把上传切到另一台服务器重

新处理，而 lvs 就直接断掉了，如果是上传一个很大的文件或者很重要的文件的话，用户可能会因此而恼火。

6.nginx 对请求的异步处理可以帮助节点服务器减轻负载，键入使用 Apache 直接对外服务，那么出现很多的窄带链接时 Apache 服务器将会占用大量内存而不能释放，使用多于一个 nginx 做 Apache 代理的话，这些窄带链接会被 nginx 挡住，Apache 上就不会堆积过多的请求，这样就减少了相当多的内存占用。这点使用 squid 也有相同的作用，即使 squid 本身配置为不缓存，对 Apache 还是有很大帮助你的。lvs 没有这些功能，也就无法能比较。

7.nginx 能支持 http 和 Email（Email 的功能估计比较少人用），lvs 所支持的应用在这点上会比 nginx 更过。

在使用上，一般最前端所采取的的策略应是 lvs，也就是 dns 的指向应为 lvs 均衡器，lvs 的优点另它非常适合做这个任务。

重要的 ip 地址，最好交由 lvs 托管，比如数据库的 ip、webservice 服务器的 ip 等等，这些 ip 地址随着时间推移，使用面会越来越大，如果更换 ip 则故障会接踵而来。所以将这些重要 ip 交给 lvs 托管式最为稳妥的，这样做的唯一缺点是需要 VIP 数量会比较多。

nginx 可以作为 lvs 节点机器使用，一是可以利用 nginx 的功能，二是可以利用 nginx 的性能。当然这一层面也可以直接使用 squid，squid 的功能方面就比 nginx 弱不少，性能上也有所逊色于 nginx。

nginx 也可以作为中层代理使用，这一层面 nginx 基本上无对手，唯一可以撼动 nginx 的就只有 lighttpd 了，不过 lighttpd 目前还没有能做到 nginx 完全的功能，配置也不那么清晰易读。另外，中层代理的 ip 也是重要的，所以中层代理业拥有一个 VIP 和 lvs 是最完美的方案了。

nginx 也可以作为网页静态服务器。

具体的应用还得具体分析，如果是比较小的网站（日 pv<1000 万），用 nginx 就完全可以了，如果机器也不少，可以用 dns 轮询，lvs 所耗费的机器还是比较多的；大型网站或者重要的服务，机器不发愁的时候要多多考虑利用 lvs。

Why nginx?

1.高并发连接：官方测试能够支撑 5 万并发连接，在实际生产环境中跑到 2——3 万并发连接数。

2.内存消耗少：在 3 万并发连接数下，开启的 10 个 nginx 进程才消耗 150M 内存（150*10=150M）。

3.配置文件非常简单：风格跟程序一样通俗易懂。

4.成本低廉：nginx 为开源软件，可以免费使用。而购买 F5 big-ip、netscaler 等硬件负载均衡交换机则需要十多万至几十万人民币。

5.支持 rewrite 重写规则：能够根据域名、url 的不同，将 http 请求分到不同的后端服务器群组。

6.内置的健康检查功能：如果 nginx proxy 后端的某台 web 服务器宕机了，不会影响前端访问。

7.节省带宽：支持 gzip 压缩，可以添加浏览器本地缓存的 header 头。