

第 6 章

Nginx HTTP 负载均衡和反向代理的配置与优化

6.1 什么是负载均衡和反向代理

随着网站访问量的快速增长，单台服务器已经无法承担大量用户的并发访问，必须采用多台服务器协同工作，以提高计算机系统的处理能力和计算强度，满足当前业务量的需求。而如何在完成同样功能的多个网络设备之间实现合理的业务量分配，使之不会出现一台设备过忙、而其他的设备却没有充分使用的情况。要解决这一问题，可以采用负载均衡的方法。

6.1.1 负载均衡

负载均衡是由多台服务器以对称的方式组成一个服务器集合，每台服务器都具有等价的地位，都可以单独对外提供服务而无须其他服务器的辅助。通过某种负载分担技术，将外部发送来的请求均匀分配到对称结构中的某一台服务器上，而接收到请求的服务器独立地回应客户的请求。均衡负载能够平均分配客户请求到服务器阵列，藉此快速获取重要数据，解决大量并发访问服务问题。这种群集技术可以用最少的投资获得接近于大型主机的性能。

6.1.2 反向代理

反向代理（Reverse Proxy）是指以代理服务器来接受 Internet 上的连接请求，然后将请求转

发给内部网络上的服务器，并将从服务器上得到的结果返回给 Internet 上请求连接的客户端，此时代理服务器对外就表现为一个服务器。

通常的代理服务器，只用于代理内部网络对 Internet 的连接请求，客户机必须指定代理服务器，并将本来要直接发送到 Web 服务器上的 http 请求发送到代理服务器中。由于外部网络上的主机并不会配置并使用这个代理服务器，普通代理服务器也被设计为在 Internet 上搜寻多个不确定的服务器，而不是针对 Internet 上多个客户机的请求访问某一个固定的服务器，因此普通的 Web 代理服务器不支持外部对内部网络的访问请求。当一个代理服务器能够代理外部网络上的主机访问内部网络时，这种代理服务的方式称为反向代理服务。此时代理服务器对外就表现为一个 Web 服务器，外部网络就可以简单把它当作一个标准的 Web 服务器而不需要特定的配置。不同之处在于，这个服务器没有保存任何网页的真实数据，所有的静态网页或 CGI 程序，都保存在内部的 Web 服务器上。因此对反向代理服务器的攻击并不会使网页信息遭到破坏，这样就增强了 Web 服务器的安全性。

反向代理方式和包过滤方式或普通代理方式并无冲突，因此可以在防火墙设备中同时使用这两种方式，其中反向代理用于外部网络访问内部网络时使用，正向代理或包过滤方式用于拒绝其他外部访问方式并提供内部网络对外部网络的访问能力。因此可以结合这些方式提供最佳的安全访问方式。

6.2 常见的 Web 负载均衡方法

Web 负载均衡的方法有很多，下面介绍几种常见的负载均衡方式。

6.2.1 用户手动选择方式

这是一种较为古老的方式，通过在主站首页入口提供不同线路、不同服务器链接的方式，来实现负载均衡。这种方式在一些提供下载业务的网站中比较常见，例如：华军软件园，如图 6-1 所示。



图 6-1 华军软件园的负载均衡方式

6.2.2 DNS 轮询方式

大多域名注册商都支持对同一主机名添加多条 A 记录，这就是 DNS 轮询，DNS 服务器将解析请求按照 A 记录的顺序，随机分配到不同的 IP 上，这样就完成了简单的负载均衡。

DNS 轮询的成本非常低，在一些不重要的服务上，被经常使用。

图 6-2 为全球第二大域名注册商——enom 的域名 Web 管理界面，这里我们为 api.bz 域名添加一个二级域名 ntp.api.bz（主机名为 ntp），并为该域名添加多条 A 记录，让其 DNS 轮询 114.80.81.72、218.75.4.130、61.129.66.79、61.153.197.226、114.80.81.1、114.80.81.69 7 个 IP，用 7 台服务器来做负载均衡。

The screenshot shows a web management interface for a domain. It features a table with columns for a checkbox, a text input for the subdomain, a dropdown menu for the record type (set to 'A (Address)'), a text input for the IP address, and a dropdown for options (set to 'No Options'). There are seven rows of this table, each with a different IP address: 114.80.81.69, 114.80.81.72, 218.75.4.130, 61.129.66.79, 114.80.81.1, 61.153.197.226, and a URL redirect for 'mail' pointing to 'http://mail.google.com/a/api.bz'. Below the table are buttons for 'new row', 'delete checked', 'add SRV or SPF record', and 'park domain'. At the bottom, there is an 'Edit MX Records' link, a 'reset' button, and a 'save' button.

图 6-2 DNS 服务器的 Web 管理界面

添加完成后，我们用 Linux 下的 dig 命令查看 ntp.api.bz 域名的域名解析情况，如代码 6-1 所示：

代码 6-1

```
[root@localhost ~]# dig ntp.api.bz

;<<>> DiG 9.3.4-P1 <<>> ntp.api.bz
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50025
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 5, ADDITIONAL: 4

;; QUESTION SECTION:
;ntp.api.bz.                IN      A

;; ANSWER SECTION:
ntp.api.bz.                 1792    IN      A       114.80.81.72
ntp.api.bz.                 1792    IN      A       218.75.4.130
ntp.api.bz.                 1792    IN      A       61.129.66.79
```

```
ntp.api.bz.      1792    IN      A       61.153.197.226
ntp.api.bz.      1792    IN      A       114.80.81.1
ntp.api.bz.      1792    IN      A       114.80.81.69

;; AUTHORITY SECTION:
api.bz.          3592    IN      NS      dns4.name-services.com.
api.bz.          3592    IN      NS      dns1.name-services.com.
api.bz.          3592    IN      NS      dns2.name-services.com.
api.bz.          3592    IN      NS      dns3.name-services.com.
api.bz.          3592    IN      NS      dns5.name-services.com.

;; ADDITIONAL SECTION:
dns1.name-services.com. 45929   IN      A       98.124.192.1
dns2.name-services.com. 4665    IN      A       98.124.197.1
dns3.name-services.com. 38315   IN      A       98.124.193.1
dns5.name-services.com. 45929   IN      A       98.124.196.1

;; Query time: 1 msec
;; SERVER: 219.141.136.10#53(219.141.136.10)
;; WHEN: Sun Aug 16 17:34:43 2009
;; MSG SIZE rcvd: 300
```

我们可以发现，域名解析正常，A 记录有七个 IP。每次访问 ntp.api.nz 域名会被随机解析到其中一个 IP 上，从而实现负载均衡的目的。

虽然 DNS 轮询成本低廉，但是，DNS 负载均衡存在两个明显的缺点。

1. 可靠性低

假设一个域名 DNS 轮询多台服务器，如果其中的一台服务器发生故障，那么所有的访问该服务器的请求将不会有所回应，这是任何人都不愿意看到的。即使从 DNS 中去掉该服务器的 IP，但在 Internet 上，各地区电信、网通等宽带接入商将众多的 DNS 存放在缓存中，以节省访问时间，DNS 记录全部生效需要几个小时，甚至更久。所以，尽管 DNS 轮流在一定程度上解决了负载均衡问题，但是却存在可靠性不高的缺点。

2. 负载分配不均衡

DNS 负载均衡采用的是简单的轮询负载算法，不能区分服务器的差异，不能反映服务器的当前运行状态，不能做到为性能较好的服务器多分配请求，甚至会出现客户请求集中在某一台服务器上的情况。

DNS 服务器是按照一定的层次结构组织的，本地 DNS 服务器会缓冲已解析的域名到 IP 地址的映射，这会导致使用该 DNS 服务器的用户在一段时间内访问的是同一台 Web 服务器，导致 Web 服务器间的负载不均衡。

此外，用户本地计算机也会缓存已解析的域名到 IP 地址的映射。当多个用户计算机都缓存

了某域名到 IP 地址的映射时，而这些用户又继续访问该域名下的网页，这时也会导致不同 Web 服务器间的负载分配不均衡。

负载不均衡可能导致的后果有：某几台服务器负荷很低，而另几台服务器负荷很高、处理缓慢；配置高的服务器分配到的请求少，而配置低的服务器分配到的请求多。

因此，DNS 轮询方式仅适用于一些可靠性要求不高的服务器集群，例如：图片服务器集群、纯静态网页服务器集群等。

6.2.3 四/七层负载均衡设备

由于 DNS 轮询的缺点，一些对可靠性要求较高的服务器集群，则通过采用四/七层负载均衡设备来实现服务器的负载均衡。

世界上第一个网络体系结构由 IBM 公司提出（1974 年，名为 SNA），以后其他公司也相继提出自己的网络体系结构如：Digital 公司的 DNA，美国国防部的 TCP/IP 等，多种网络体系结构并存，其结果是若采用 IBM 的结构，只能选用 IBM 的产品，只能与同种结构的网络互联。为了促进计算机网络的发展，国际标准化组织 ISO 于 1977 年成立了一个委员会，在现有网络的基础上，提出了不基于具体机型、操作系统或公司的网络体系结构，称为开放系统互联模型（OSI，open system interconnection）。

这个模型把网络通信的工作分为七层（可参见图 6-3）。一至四层被认为是低层，这些层与数据移动密切相关。五至七层是高层，包含应用程序级的数据。每一层负责一项具体的工作，然后把数据传送到下一层。由低到高具体分为：物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

OSI 模型的最低层或第一层：物理层

物理层包括物理连网媒介，实际上就是布线、光纤、网卡和其他用来把两台网络通信设备连接在一起的设施。它规定了激活、维持、关闭通信端点之间的机械特性、电气特性、功能特性及过程特性。虽然物理层不提供纠错服务，但它能够设定数据传输速率并监测数据出错率。

OSI 模型的第二层：数据链路层

数据链路层的主要作用是控制网络层与物理层之间的通信。它保证了数据在不可靠的物理线路上进行可靠的传递。它把从网络层接收到的数据分割成特定的可被物理层传输的帧，保证了传输的可靠性。它的主要作用包括：物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。它是独立于网络层和物理层的，工作时无须关心计算机是否正在运行软件还是其他操作。

数据链路层协议的主要内容包括：SDLC、HDLC、PPP、STP、帧中继等。

OSI 模型的第三层：网络层

很多用户经常混淆第二层和第三层的界限，简单来说，如果你在谈论一个与 IP 地址、路由协议或地址解析协议（ARP）相关的问题，那么这就是第三层的问题。

网络层负责对子网间的数据包进行路由选择，它通过综合考虑发送优先权、网络拥塞程度、服务质量及可选路由的花费来决定一个网络中两个节点的最佳路径。另外，它还可以实现拥塞控制、网际互联等功能。

网络层协议的主要内容包括：IP、IPX、RIP、OSPF 等。

OSI 模型的第四层：传输层

传输层是 OSI 模型中最重要的一层，它是两台计算机经过网络进行数据通信时，第一个端到端的层次，起到缓冲作用。当网络层的服务质量不能满足要求时，它将提高服务，以满足高层的要求；而当网络层服务质量较好时，它只须进行很少的工作。另外，它还要处理端到端的差错控制和流量控制等问题，最终为会话提供可靠的、无误的数据传输。

传输层协议的主要内容包括：TCP、UDP、SPX 等。

在 IP 协议栈中第四层是 TCP（传输控制协议）和 UDP（用户数据报协议）所在的协议层。TCP 和 UDP 包含端口号，它可以唯一区分每个数据包包含哪些应用协议（例如 HTTP、FTP、telnet 等）。TCP/UDP 端口号提供的附加信息可以为网络交换机所利用，四层交换机利用这种信息来区分包中的数据，这是第四层交换的基础。

OSI 模型的第五层：会话层

会话层负责在网络中的两节点之间建立和维持通信，并保持会话同步，它还决定通信是否中断，以及通信中断时决定从何处重新发送。

OSI 模型的第六层：表示层

表示层的作用是管理数据的解密与加密，如常见的系统口令处理，当你的账户数据在发送前被加密，在网络的另一端，表示层将对接收到的数据解密。另外，表示层还要对图片和文件格式信息进行解码和编码。

OSI 模型的第七层：应用层

简单来说，应用层就是为操作系统或网络应用程序提供访问网络服务的接口，包括文件传输、文件管理及电子邮件等的信息处理。

应用层协议的代表包括：Telnet、FTP、HTTP、SNMP 等。

OSI 网络模型如图 6-3 所示。

现代负载均衡技术通常操作于 OSI 网络模型的第四层或第七层。第四层负载均衡将一个 Internet 上合法注册的 IP 地址映射为多个内部服务器的 IP 地址，对每次 TCP 连接请求动态使用其中一个内部 IP 地址，达到负载均衡的目的。在第四层交换机中，此种均衡技术得到广泛的应用，一个目标地址是服务器群 VIP（虚拟 IP，Virtual IP address）连接请求的数据包流经交换机，交换机根据源端和目的 IP 地址、TCP 或 UDP 端口号和一定的负载均衡策略，在服务器 IP 和 VIP 间进行映射，选取服务器群中最好的服务器来处理连接请求。

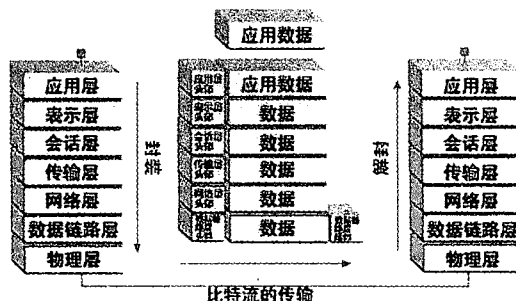


图 6-3 OSI 网络模型

第七层负载均衡控制应用层服务的内容，提供了一种对访问流量的高层控制方式，适合对 HTTP 服务器群的应用。第七层负载均衡技术通过检查流经的 HTTP 报头，根据报头内的信息来执行负载均衡任务。

常见的四/七层负载均衡设备：

1. 硬件四/七层负载均衡交换机

硬件四/七层负载均衡交换机的代表有：F5 BIG-IP、Citrix NetScaler、Radware、Cisco CSS、Foundry 等产品，这些产品价格不菲，高达几十万元人民币。在中国大陆，采用 F5 Network 公司的 BIG-IP 负载均衡交换机的网站（有些网站为部分频道采用）最多，包括：新浪网、雅虎、百度、搜狐、凤凰网、央视国际、中华英才网、猫扑、慧聪网等。

图 6-4 是一张 F5 BIG-IP 实现动、静态网页分离的负载均衡架构图，很好地描述了 F5 BIG-IP 是如何实现四/七层负载均衡的。

(1) 如图，假设域名 `blog.s135.com` 被解析到 F5 的外网/公网虚拟 IP：61.1.1.3 (vs_squid)，该虚拟 IP 下有一个服务器池 (pool_squid)，该服务器池下包含两台真实的 Squid 服务器 (192.168.1.11 和 192.168.1.12)。

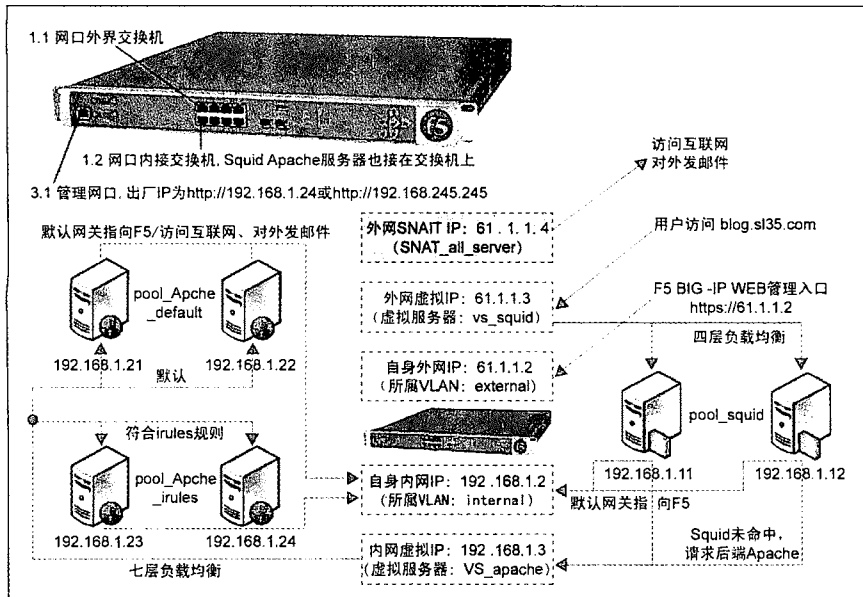


图 6-4 F5 BIG-IP 实现动、静态网页分离的负载均衡架构图

(2) 如果 Squid 缓存未命中, 则会请求 F5 的内网虚拟 IP: 192.168.1.3 (vs_apache), 该虚拟 IP 下有一个默认服务器池 (pool_apache_default), 该服务器池下包含两台真实的 Apache 服务器 (192.168.1.21 和 192.168.1.22), 当该虚拟 IP 匹配 iRules 规则时, 则会访问另外一个服务器池 (pool_apache_irules), 该服务器池下同样包含两台真实的 Apache 服务器 (192.168.1.23 和 192.168.1.24)。

(3) 另外, 所有真实服务器的默认网关指向 F5 的自身内网 IP, 即 192.168.1.2。

(4) 所有的真实服务器通过 SNAT IP 地址 61.1.1.4 访问互联网。

2. 软件四层负载均衡

软件四层负载均衡的代表作品为 LVS (Linux Virtual Server), 作者为曾经在国家并行与分布式处理重点实验室工作, 现在已加盟淘宝网的章文嵩博士。LVS 是一个开源的软件, 可以实现 LINUX 平台下的简单负载均衡。LVS 集群采用 IP 负载均衡技术和基于内容请求分发技术。调度器具有很好的吞吐率, 将请求均衡地转移到不同的服务器上执行, 且调度器自动屏蔽掉服务器的故障, 从而将一组服务器构成一个高性能的、高可用的虚拟服务器。整个服务器集群的结构对客户是透明的, 而且无须修改客户端和服务器端的程序。为此, 在设计时要考虑系统的透明性、可伸缩性、高可用性和易管理性。

LVS 负载均衡的结构如图 6-5 所示。

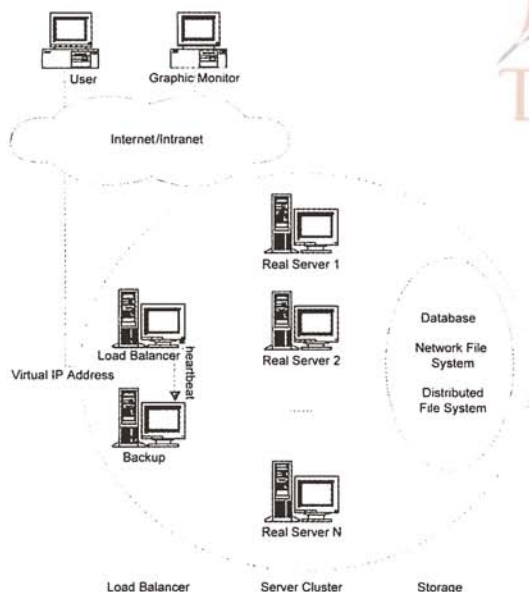


图 6-5 LVS 负载均衡架构图

3. 软件七层负载均衡

软件的七层负载均衡大多基于 HTTP 反向代理方式，代表产品有 Nginx、L7SW (Layer7 switching)、HAProxy 等。Nginx 的反向代理负载均衡能够很好地支持虚拟主机，可配置性很强，可以按轮询、IP 哈希、URL 哈希、权重等多种方式对后端服务器做负载均衡，同时还支持后端服务器的健康检查。

6.2.4 多线多地区智能 DNS 解析与混合负载均衡方式

以新浪首页 (www.sina.com.cn) 为例，负载均衡同时用到了“多线多地区智能 DNS 解析、DNS 轮询、四/七层负载均衡交换机”等技术。智能 DNS 解析能够根据用户本地设置的 DNS 服务器线路和地区，将对同一个域名请求解析到不同的 IP 上。

例如：当北京电信用户访问 www.sina.com.cn 时，会被新浪的 DNS 服务器解析到北京电信机房的 IP 上；当北京网通用户访问 www.sina.com.cn 时，会被解析到北京网通机房的 IP 上；当教育网的用户访问 www.sina.com.cn 时，会被解析到教育网机房的 IP 上；当广东电信的用户访问 www.sina.com.cn 时，会被解析到广州电信机房的 IP 上；当湖南、湖北的电信用户访问 www.sina.com.cn 时，会被解析到武汉电信机房的 IP 上，等等。

将 DNS 地址设为北京电信的 DNS 服务器 219.141.136.10，通过 Linux 下的 dig 命令可以发现，

访问 `www.sina.com.cn` 被解析到了北京电信的多台服务器的 IP 上, 这属于智能 DNS 解析+DNS 轮询解决负载均衡, 如代码 6-2 所示。

代码 6-2

```
[root@localhost ~]# dig www.sina.com.cn

; <<>> DiG 9.3.4-P1 <<>> www.sina.com.cn
;; global options: printcmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 46279
;; flags: qr rd ra; QUERY: 1, ANSWER: 18, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www.sina.com.cn.                IN      A

;; ANSWER SECTION:
www.sina.com.cn.                60      IN      CNAME   jupiter.sina.com.cn.
jupiter.sina.com.cn.           600     IN      CNAME   hydra.sina.com.cn.
hydra.sina.com.cn.             20      IN      A        218.30.108.191
hydra.sina.com.cn.             20      IN      A        218.30.108.192
hydra.sina.com.cn.             20      IN      A        218.30.108.63
hydra.sina.com.cn.             20      IN      A        218.30.108.64
hydra.sina.com.cn.             20      IN      A        218.30.108.65
hydra.sina.com.cn.             20      IN      A        218.30.108.66
hydra.sina.com.cn.             20      IN      A        218.30.108.67
hydra.sina.com.cn.             20      IN      A        218.30.108.180
hydra.sina.com.cn.             20      IN      A        218.30.108.181
hydra.sina.com.cn.             20      IN      A        218.30.108.182
hydra.sina.com.cn.             20      IN      A        218.30.108.183
hydra.sina.com.cn.             20      IN      A        218.30.108.184
hydra.sina.com.cn.             20      IN      A        218.30.108.185
hydra.sina.com.cn.             20      IN      A        218.30.108.186
hydra.sina.com.cn.             20      IN      A        218.30.108.187
hydra.sina.com.cn.             20      IN      A        218.30.108.189

;; AUTHORITY SECTION:
sina.com.cn.                    72560   IN      NS       ns1.sina.com.cn.
sina.com.cn.                    72560   IN      NS       ns2.sina.com.cn.
sina.com.cn.                    72560   IN      NS       ns3.sina.com.cn.

;; ADDITIONAL SECTION:
ns1.sina.com.cn.                54070   IN      A        202.106.184.166
ns2.sina.com.cn.                73689   IN      A        61.172.201.254
ns3.sina.com.cn.                57207   IN      A        202.108.44.55

;; Query time: 1 msec
;; SERVER: 219.141.136.10#53(219.141.136.10)
;; WHEN: Sun Aug 16 23:54:43 2009
;; MSG SIZE rcvd: 433
```

将 DNS 地址设为北京网通的 DNS 服务器 202.106.0.20，通过 Linux 下的 dig 命令可以发现，访问 www.sina.com.cn 最终被解析到了北京网通的一个 IP 地址 202.108.33.32 上，这属于用智能 DNS 解析+四/七层负载均衡交换机解决负载均衡，该 IP 地址 202.108.33.32 是四/七层负载均衡交换机的虚拟 IP，如代码 6-3 所示。

代码 6-3

```
[root@localhost ~]# dig www.sina.com.cn

;<<>> DiG 9.3.4-P1 <<>> www.sina.com.cn
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65261
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.sina.com.cn.                IN      A

;; ANSWER SECTION:
www.sina.com.cn.                24      IN      CNAME   jupiter.sina.com.cn.
jupiter.sina.com.cn.           320     IN      A       202.108.33.32

;; Query time: 2 msec
;; SERVER: 202.106.0.20#53(202.106.0.20)
;; WHEN: Sun Aug 16 23:58:10 2009
;; MSG SIZE rcvd: 71
```

6.3 Nginx 负载均衡与反向代理的配置实例

6.3.1 完整的 Nginx 反向代理示例

完整的 Nginx 反向代理示例如代码 6-4 所示。

代码 6-4

```
user www www;

worker_processes 10;

error_log /data1/logs/nginx_error.log crit;

pid /usr/local/webserver/nginx/nginx.pid;

worker_rlimit_nofile 51200;

events
```

```
{
    use epoll;
    worker_connections 51200;
}

http
{
    include      mime.types;
    default_type application/octet-stream;

    #charset utf-8;

    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;

    sendfile on;
    #tcp_nopush    on;

    keepalive_timeout 65;

    tcp_nodelay on;

    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;

    gzip on;
    gzip_min_length 1k;
    gzip_buffers    4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 2;
    gzip_types      text/plain application/x-javascript text/css application/xml;
    gzip_vary on;

    #limit_zone crawler $binary_remote_addr 10m;

    #允许客户端请求的最大单个文件字节数
    client_max_body_size    300m;

    #缓冲区代理缓冲用户端请求的最大字节数，可以理解为先保存到本地再传给用户
    client_body_buffer_size 128k;

    #跟后端服务器连接的超时时间_发起握手等候响应超时时间
    proxy_connect_timeout    600;

    #连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理
    proxy_read_timeout        600;
```

```

#后端服务器数据回传时间_就是在规定时间内后端服务器必须传完所有的数据
proxy_send_timeout      600;

#代理请求缓存区_这个缓存区间会保存用户的头信息以供Nginx进行规则处理_一般只要能保存下头信息即可
proxy_buffer_size        16k;

#同上 告诉Nginx保存单个用的几个Buffer 最大用多大空间
proxy_buffers            4 32k;

#如果系统很忙的时候可以申请更大的proxy_buffers 官方推荐*2
proxy_busy_buffers_size  64k;

#proxy 缓存临时文件的大小
proxy_temp_file_write_size 64k;

upstream php_server_pool {
    server 192.168.1.10:80 weight=4 max_fails=2 fail_timeout=30s;
    server 192.168.1.11:80 weight=4 max_fails=2 fail_timeout=30s;
    server 192.168.1.12:80 weight=2 max_fails=2 fail_timeout=30s;
}

upstream message_server_pool {
    server 192.168.1.13:3245;
    server 192.168.1.14:3245 down;
}

upstream bbs_server_pool {
    server 192.168.1.15:80 weight=1 max_fails=2 fail_timeout=30s;
    server 192.168.1.16:80 weight=1 max_fails=2 fail_timeout=30s;
    server 192.168.1.17:80 weight=1 max_fails=2 fail_timeout=30s;
    server 192.168.1.18:80 weight=1 max_fails=2 fail_timeout=30s;
}

#第一个虚拟主机, 反向代理 php_server_pool 这组服务器
server
{
    listen      80;
    server_name www.yourdomain.com;

    location /
    {
        #如果后端的服务器返回 502、504、执行超时等错误, 自动将请求转发到 upstream 负载均衡池中的
        另一台服务器, 实现故障转移。
        proxy_next_upstream http_502 http_504 error timeout invalid_header;
        proxy_pass http://php_server_pool;
        proxy_set_header Host www.yourdomain.com;
        proxy_set_header X-Forwarded-For $remote_addr;
    }

    access_log /data1/logs/www.yourdomain.com_access.log;
}

```

```

#第二个虚拟主机
server
{
    listen      80;
    server_name www1.yourdomain.com;

    #访问 http://www1.yourdomain.com/message/**地址, 反向代理 message_server_pool 这组服务器
    location /message/
    {
        proxy_pass http://message_server_pool;
        proxy_set_header Host $host;
    }

    #访问除了/message/之外的 http://www1.yourdomain.com/**地址, 反向代理 php_server_pool 这组服务器
    location /
    {
        proxy_pass http://php_server_pool;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }

    access_log /data1/logs/message.yourdomain.com_access.log;
}

#第三个虚拟主机
server{
    listen      80;
    server_name bbs.yourdomain.com *.bbs.yourdomain.com;

    location /
    {
        proxy_pass http://bbs_server_pool;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }

    access_log off;
}
}

```

通过上述示例, 我们已经看到 Nginx 对于多个域名的负载均衡是如何配置的。Upstream 指令用于设置一组可以在 proxy_pass 和 fastcgi_pass 指令中使用的代理服务器, 默认的负载均衡方式为轮询。Upstream 模块中的 Server 指令用于指定后端服务器的名称和参数, 服务器的名称可以是一个域名、一个 IP 地址、端口号或 UNIX Socket。

而在 server{...} 虚拟主机内, 可以通过 proxy_pass 和 fastcgi_pass 指令设置进行反向代理的 upstream 服务器集群。

`proxy_set_header` 指令用于在向反向代理的后端 Web 服务器发起请求时添加指定的 Header 头信息。

当后端 Web 服务器上有多个基于域名的虚拟主机时,要通过添加 Header 头信息 Host,用于指定请求的域名,这样后端 Web 服务器才能识别该反向代理访问请求由哪一个虚拟主机来处理。

使用反向代理之后,后端 Web 服务器(以 PHP 为例)就不能直接通过 `$_SERVER["REMOTE_ADDR"]` 变量来获取用户的真实 IP 了,通过 `$_SERVER["REMOTE_ADDR"]` 获取的将是 Nginx 负载均衡服务器的 IP。这时,就要通过在 Nginx 反向代理时添加 Header 头信息 X-Forwarded-For,让后端 Web 服务器(以 PHP 为例)能够通过 `$_SERVER["HTTP_X_FORWARDED_FOR"]` 获取到用户的真实 IP。

6.3.2 Nginx 负载均衡与反向代理实现动、静态网页分离

动、静态网页分离,就是让动态 PHP 等程序网页去访问 PHP Web 服务器,让缓存页、图片、JavaScript、CSS、Flash 去访问 Squid 等缓存服务器。

在配置 Nginx 负载均衡与反向代理之前,我们首先来看看 NetScaler 等硬件四/七层负载均衡交换机是如何实现负载均衡的,如图 6-6 所示。

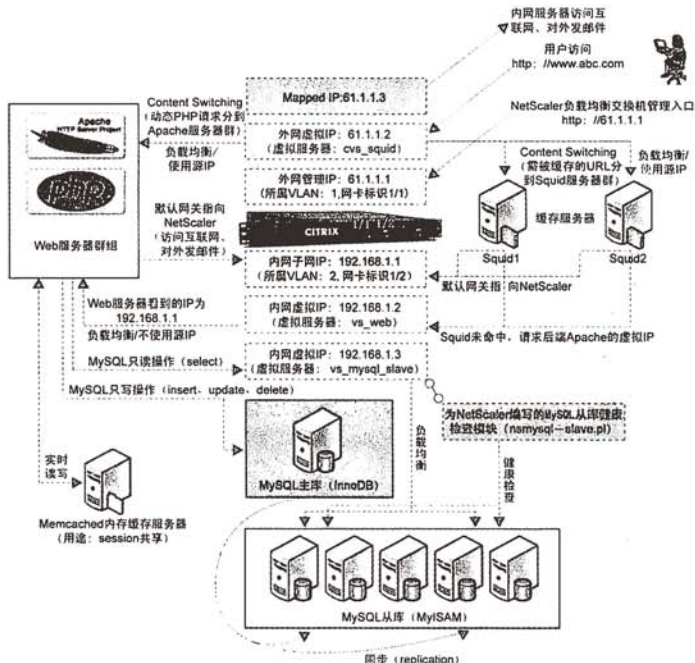


图 6-6 NetScaler 负载均衡交换机动静分离系统架构图

NetScaler 的七层负载均衡是基于 TCP 的，可以用于 Web Server、MySQL 数据库、邮件服务器等大多数基于 TCP 服务器的负载均衡。Nginx 的七层负载均衡仅支持 HTTP、邮件协议，能够实现 Web Server、邮件服务器的负载均衡。在 Web 服务器集群应用中，可以通过 Nginx 负载均衡，来实现并代替 NetScaler 七层负载均衡的部分功能。对于以上的 NetScaler 负载均衡架构，我们可以用以下的 Nginx 负载均衡架构来代替其中的 Web 服务器部分，如图 6-7 所示。

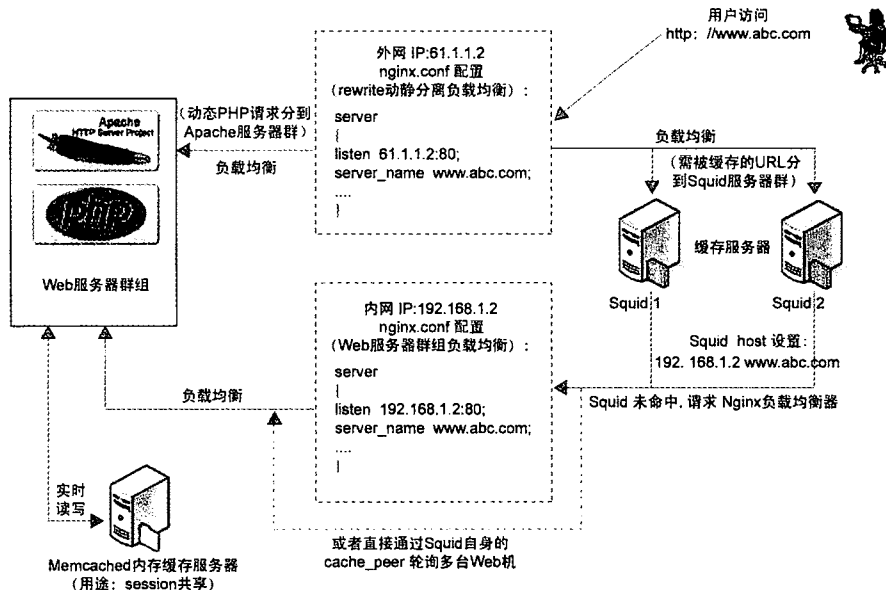


图 6-7 Nginx 反向代理负载均衡动静分离系统架构图

6.4 Nginx 负载均衡的 HTTP Upstream 模块

Upstream 模块是 Nginx 负载均衡的主要模块，它提供了一个简单方法来实现轮询和客户端 IP 之间的后端服务器负载均衡，并可以对后端服务器进行健康检查。

代码示例如 6-5 所示：

代码 6-5

```
upstream backend {  
    server backend1.example.com weight=5;  
    server backend2.example.com:8080;  
    server unix:/tmp/backend3;  
}
```

```
server {
```



```
location / {  
    proxy_pass http://backend;  
}
```

6.4.1 ip_hash 指令

ip_hash

语法: ip_hash

默认值: none

使用环境: upstream

当对后端的多台动态应用服务器做负载均衡时, ip_hash 指令能够将某个客户端 IP 的请求通过哈希算法定位到同一台后端服务器上。这样, 当来自某个 IP 的用户在后端 Web 服务器 A 上登录后, 再访问该站点的其他 URL, 能保证其访问的还是后端 Web 服务器 A。如果不采用 ip_hash 指令, 假设来自某个 IP 的用户在后端 Web 服务器 A 上登录后, 再访问该站点的其他 URL, 有可能被定向到后端 Web 服务器 B, C, ... 上, 由于用户登录后 SESSION 信息是记录在服务器 A 上的, B, C, ... 上没有, 这时就会提示用户未登录。

使用 ip_hash 指令无法保证后端服务器的负载均衡, 可能有些后端服务器接收到的请求多, 有些后端服务器接收到的请求少, 而且设置后端服务器权重等方法将不起作用。所以, 如果后端的动态应用服务器能够做到 SESSION 共享, 还是建议采用后端服务器的 SESSION 共享方式来代替 Nginx 的 ip_hash 方式。

如果后端服务器有时要从 Nginx 负载均衡 (已使用 ip_hash) 中摘除一段时间, 你必须将其标记为 “down”, 而不是直接从配置文件中删掉或注释掉该后端服务器的信息。代码示例如 6-6:

代码 6-6

```
upstream backend {  
    ip_hash;  
    server backend1.example.com;  
    server backend2.example.com;  
    server backend3.example.com down;  
    server backend4.example.com;  
}
```

这样, 当原来为 4 台后端服务器时, 摘除 backend3.example.com (标记为 “down”) 后, Nginx 仍然会按 4 台服务器进行哈希。如果直接注释掉 “server backend3.example.com” 这行, Nginx 就会按照 3 台服务器进行重新哈希, 原来被哈希到 backend1.example.com 的客户端 IP 有可能被哈希到 backend2.example.com 服务器上, 原有的 SESSION 就会失效。

6.4.2 server 指令

server

语法: server name [parameters]

默认值: none

使用环境: upstream

该指令用于指定后端服务器的名称和参数。服务器的名称可以是一个域名、一个 IP 地址、端口号或 UNIX Socket。

在后端服务器名称之后，可以跟以下参数：

weight = NUMBER——设置服务器的权重，权重数值越高，被分配到的客户端请求数越多。如果没有设置权重，则为默认权重 1。

max_fails = NUMBER——在参数 **fail_timeout** 指定的时间内对后端服务器请求失败的次数，如果检测到后端服务器无法连接及发生服务器错误（404 错误除外），则标记为失败。如果没有设置，则为默认值 1。设为数值 0 将关闭这项检查。

fail_timeout = TIME——在经历参数 **max_fails** 设置的失败次数后，暂停的时间。

down——标记服务器为永久离线状态，用于 **ip_hash** 指令。

backup——仅仅在非 **backup** 服务器全部宕机或繁忙的时候才启用。

示例如下：

```
upstream backend {  
    server backend1.example.com weight=5;  
    server 127.0.0.1:8080 max_fails=3 fail_timeout=30s;  
    server unix:/tmp/backend3;  
}
```

6.4.3 upstream 指令

upstream

语法: upstream name { ... }

默认值: none

使用环境: http

该指令用于设置一组可以在 **proxy_pass** 和 **fastcgi_pass** 指令中使用的代理服务器，默认的负

载均衡方式为轮询。示例如下：

```
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080      max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;
}
```

6.4.4 upstream 相关变量

从 Nginx 0.5.18 版本开始，可以支持用 `log_format` 指令设置日志格式，日志格式中可以使用变量，例如：

```
log_format timing '$remote_addr - $remote_user [$time_local] $request '
    'upstream_response_time $upstream_response_time '
    'msec $msec request_time $request_time';
```

```
log_format up_head '$remote_addr - $remote_user [$time_local] $request '
    'upstream_http_content_type $upstream_http_content_type';
```

upstream 模块拥有以下变量：

`$upstream_addr`

处理请求的 upstream 服务器地址。

`$upstream_status`

Upstream 服务器的应答状态。

`$upstream_response_time`

Upstream 服务器响应时间（毫秒），多个响应以逗号和冒号分割。

`$upstream_http_$HEADER`

任意的 HTTP 协议头信息，例如：

```
$upstream_http_host
```

6.5 Nginx 负载均衡服务器的双机高可用

如果将 Web 服务器集群当作一个城池，那么负载均衡服务器则相当于城门，重要性不言而喻。如果“城门”关闭了，与外界通道也就掐断了。如果只有一台 Nginx 负载均衡服务器，当该服务器发生故障时，则会导致整个网站无法访问。因此，我们需要两台以上的 Nginx 负载均衡服务器，实现故障转移与高可用。

双机高可用一般是通过虚拟 IP（也称漂移 IP）方式来实现的，基于 Linux/Unix 的 IP 别名技术。双机高可用方式目前可分为两种：第一种方式为一台主服务器加一台热备服务器，正常情况下主服务器绑定一个公网虚拟 IP，提供负载均衡服务，热备服务器处于空闲状态，当主服务器发生故障时，热备服务器接管主服务器的虚拟 IP，提供负载均衡服务；第二种方式为两台负载均衡服务器都处于活动状态，各自绑定一个公网虚拟 IP，提供负载均衡服务，当其中一台服务器发生故障时，另一台服务器接管发生故障服务器的虚拟 IP。第一种方式较为常见，但始终有一台服务器处于空闲状态，浪费了一台服务器的负载均衡处理能力。第二种方式需要多用一个公网 IP，笔者已经在金山游戏官方网站——逍遥网（xoyo.com）线上环境成功使用，能够在正常情况下将两台服务器都用于实际的负载均衡处理。

第一种方式：

(1) www.yourdomain.com 域名解析到虚拟 IP 61.1.1.2 上。

(2) 正常情况下，主机 61.1.1.4 绑定虚拟 IP 61.1.1.2。

```
/sbin/ifconfig eth0:1 61.1.1.2 broadcast 61.1.1.255 netmask 255.255.255.0 up
/sbin/route add -host 61.1.1.2 dev eth0:1
/sbin/arping -I eth0 -c 3 -s 61.1.1.2 61.1.1.1
```

(3) 用户访问 www.yourdomain.com（虚拟 IP 61.1.1.2）实际访问的是主机 61.1.1.4，而备机 61.1.1.5 则处于空闲状态。

(4) 如果主机 61.1.1.4 发生故障，备机 61.1.1.5 将在几秒钟内接管虚拟 IP 61.1.1.2，与自己绑定，并发送 ARPing 包给 IDC 的公网网关刷新 MAC 地址。

```
/sbin/ifconfig eth0:1 61.1.1.2 broadcast 61.1.1.255 netmask 255.255.255.0 up
/sbin/route add -host 61.1.1.2 dev eth0:1
/sbin/arping -I eth0 -c 3 -s 61.1.1.2 61.1.1.1
```

(5) 这时，用户访问 www.yourdomain.com（虚拟 IP 61.1.1.2）实际上访问的是备机 61.1.1.5，从而实现故障转移与高可用，避免了单点故障。转移过程如图 6-8 所示。

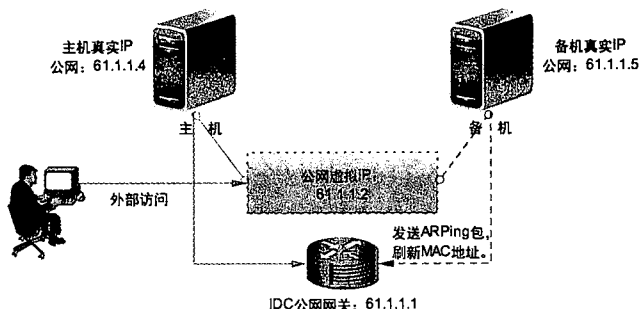


图 6-8 一台主机配一台备机的可用服务方式

另外，第一种方式还可以利用基于 VRRP 路由协议的 Keepalived 软件来实现。

第二种方式：

- (1) www.yourdomain.com 域名通过 DNS 轮询解析到虚拟 IP 61.1.1.2 和 61.1.1.3 上。
- (2) 正常情况下，服务器①61.1.1.4 绑定虚拟 IP 61.1.1.2，服务器②61.1.1.5 绑定虚拟 IP 61.1.1.3。其联结与运行方式如图 6-9 所示。

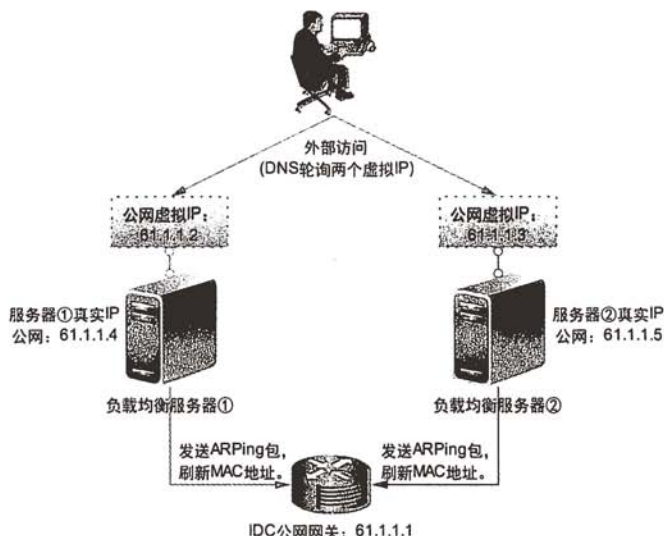


图 6-9 两台负载均衡服务器的高可用服务方式（正常状态）

在服务器①61.1.1.4 上执行以下命令：

```
/sbin/ifconfig eth0:1 61.1.1.2 broadcast 61.1.1.255 netmask 255.255.255.0 up
/sbin/route add -host 61.1.1.2 dev eth0:1
/sbin/arping -I eth0 -c 3 -s 61.1.1.2 61.1.1.1
```

在服务器②61.1.1.5 上执行以下命令：

```
/sbin/ifconfig eth0:1 61.1.1.3 broadcast 61.1.1.255 netmask 255.255.255.0 up
/sbin/route add -host 61.1.1.3 dev eth0:1
/sbin/arping -I eth0 -c 3 -s 61.1.1.3 61.1.1.1
```

(3) 用户访问 www.yourdomain.com（虚拟 IP 61.1.1.2 和 61.1.1.3）实际上是根据 DNS 轮询访问两台负载均衡服务器，两台服务器均处于活动状态。

(4) 如果服务器①发生故障，服务器②将在几秒钟内接管服务器①的虚拟 IP 61.1.1.2，与自己绑定，并发送 ARPing 包给 IDC 的公网网关刷新 MAC 地址。这时，服务器②同时绑定 61.1.1.2 和 61.1.1.3 两个虚拟 IP，其联结与运行方式如图 6-10 所示。

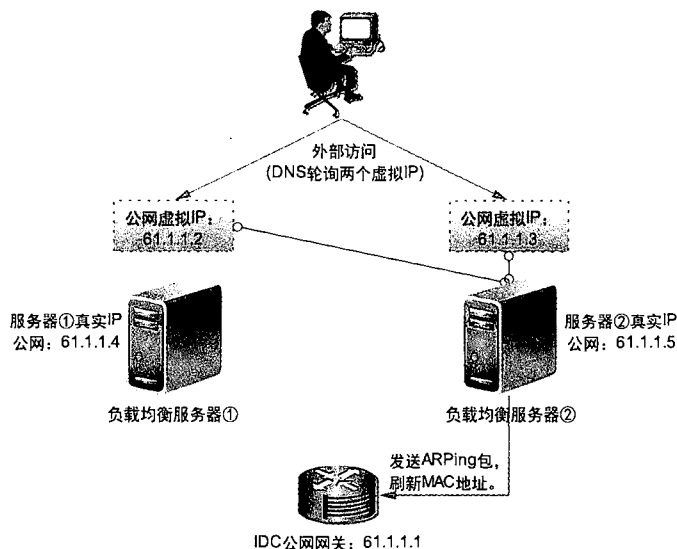


图 6-10 其中一台负载均衡服务器发生故障的运行方式

在服务器②61.1.1.5 上执行以下命令：

```
/sbin/ifconfig eth0:1 61.1.1.2 broadcast 61.1.1.255 netmask 255.255.255.0 up
/sbin/route add -host 61.1.1.2 dev eth0:1
/sbin/arping -I eth0 -c 3 -s 61.1.1.2 61.1.1.1
```

我们可以写两个 shell 脚本，来实现第二种方式的自动故障转移。

以下代码 6-7 为脚本 1 (nginx_ha1.sh)，部署在 Nginx 负载均衡服务器①：

代码 6-7

```
#!/bin/sh
LANG=C
date=$(date -d "today" +"%Y-%m-%d %H:%M:%S")

function_bind_vip1()
{
    /sbin/ifconfig eth0:ha1 61.1.1.2 broadcast 219.232.254.255 netmask
    255.255.255.192 up
    /sbin/route add -host 61.1.1.2 dev eth0:ha1
}

function_bind_vip2()
{
    /sbin/ifconfig eth0:ha2 61.1.1.3 broadcast 219.232.254.255 netmask
    255.255.255.192 up
    /sbin/route add -host 61.1.1.3 dev eth0:ha2
}
```

```

function_restart_nginx()
{
    kill -USR1 `cat /usr/local/webserver/nginx/nginx.pid`
}

function_remove_vip1()
{
    /sbin/ifconfig eth0:ha1 61.1.1.2 broadcast 219.232.254.255 netmask
    255.255.255.192 down
}

function_remove_vip2()
{
    /sbin/ifconfig eth0:ha2 61.1.1.3 broadcast 219.232.254.255 netmask
    255.255.255.192 down
}

function_vip_arping1()
{
    /sbin/arping -I eth0 -c 3 -s 61.1.1.2 61.1.1.1 > /dev/null 2>&1
}

function_vip_arping2()
{
    /sbin/arping -I eth0 -c 3 -s 61.1.1.3 61.1.1.1 > /dev/null 2>&1
}

bind_time_vip1="N";
bind_time_vip2="N";

while true
do
    httpcode_rip1=`/usr/bin/curl -o /dev/null -s -w %{http_code} http://61.1.1.4`
    httpcode_rip2=`/usr/bin/curl -o /dev/null -s -w %{http_code} http://61.1.1.5`

    if [ x$httpcode_rip1 == "x200" ];
    then
        if [ $bind_time_vip1 == "N" ];
        then
            function_bind_vip1
            function_vip_arping1
            function_restart_nginx
            bind_time_vip1="Y"
        fi
        function_vip_arping1
    else
        if [ $bind_time_vip1 == "Y" ];
        then
            function_remove_vip1
            bind_time_vip1="N"
        fi
    fi
fi

```

```

if [ x$httpcode_rip2 == "x200" ];
then
    if [ $bind_time_vip2 == "Y" ];
    then
        function_remove_vip2
        bind_time_vip2="N"
    fi
else
    if [ $bind_time_vip2 == "N" ];
    then
        function_bind_vip2
        function_vip_arping2
        function_restart_nginx
        bind_time_vip2="Y"
    fi
    function_vip_arping2
fi

sleep 5
done

```

在 Nginx 负载均衡服务器①将脚本驻留后台运行:

```
nohup /bin/sh ./nginx_ha1.sh 2>&1 > /dev/null &
```

以下代码 6-8 为脚本 2 (server2.sh)，部署在 Nginx 负载均衡服务器②:

代码 6-8

```

#!/bin/sh
LANG=C
date=$(date -d "today" +"%Y-%m-%d %H:%M:%S")

function_bind_vip1()
{
    /sbin/ifconfig eth0:ha1 61.1.1.3 broadcast 219.232.254.255 netmask
    255.255.255.192 up
    /sbin/route add -host 61.1.1.3 dev eth0:ha1
}

function_bind_vip2()
{
    /sbin/ifconfig eth0:ha2 61.1.1.2 broadcast 219.232.254.255 netmask
    255.255.255.192 up
    /sbin/route add -host 61.1.1.2 dev eth0:ha2
}

function_restart_nginx()
{
    kill -USR1 `cat /usr/local/webserver/nginx/nginx.pid`
}

```



```

function_remove_vip1()
{
    /sbin/ifconfig eth0:ha1 61.1.1.3 broadcast 219.232.254.255 netmask
    255.255.255.192 down
}

function_remove_vip2()
{
    /sbin/ifconfig eth0:ha2 61.1.1.2 broadcast 219.232.254.255 netmask
    255.255.255.192 down
}

function_vip_arping1()
{
    /sbin/arping -I eth0 -c 3 -s 61.1.1.3 61.1.1.1 > /dev/null 2>&1
}

function_vip_arping2()
{
    /sbin/arping -I eth0 -c 3 -s 61.1.1.2 61.1.1.1 > /dev/null 2>&1
}

bind_time_vip1="N";
bind_time_vip2="N";

while true
do
    httpcode_rip1=`/usr/bin/curl -o /dev/null -s -w %{http_code} http://61.1.1.5`
    httpcode_rip2=`/usr/bin/curl -o /dev/null -s -w %{http_code} http://61.1.1.4`

    if [ x$httpcode_rip1 == "x200" ];
    then
        if [ $bind_time_vip1 == "N" ];
        then
            function_bind_vip1
            function_vip_arping1
            function_restart_nginx
            bind_time_vip1="Y"
        fi
        function_vip_arping1
    else
        if [ $bind_time_vip1 == "Y" ];
        then
            function_remove_vip1
            bind_time_vip1="N"
        fi
    fi

    if [ x$httpcode_rip2 == "x200" ];
    then
        if [ $bind_time_vip2 == "Y" ];
        then

```

```
        function_remove_vip2
        bind_time_vip2="N"
    fi
else
    if [ $bind_time_vip2 == "N" ];
    then
        function_bind_vip2
        function_vip_arping2
        function_restart_nginx
        bind_time_vip2="Y"
    fi
    function_vip_arping2
fi

sleep 5
done
```

在 Nginx 负载均衡服务器②将脚本驻留后台运行：

```
nohup /bin/sh ./nginx_ha2.sh 2>&1 > /dev/null &
```