

终点（接收方 IP 地址）为 Web 服务器 IP 地址的包通过（图 5.2 中表的第 1 行）。如果可以确定发送方 IP 地址，也可以将其加入规则，但这个例子中起点是不确定的，因此可以不将发送方 IP 地址设为判断条件。

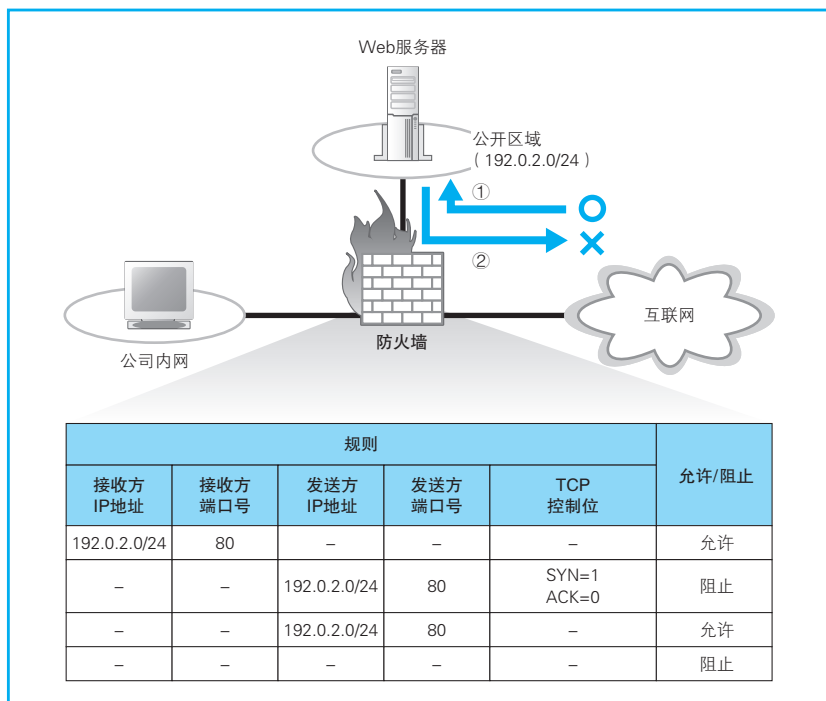


图 5.2 包过滤的典型示例

这样一来，从互联网发往 Web 服务器的包就可以通过防火墙了，但光这样还无法完成访问。因为收到包之后，Web 服务器需要通过确认应答机制<sup>①</sup>通知发送方数据已经正常收到，这需要 Web 服务器向互联网发送包。在 Web 服务器发往互联网的包中，我们可以将起点（发送方 IP 地址）为 Web 服务器地址的包设置为允许通过（图 5.2 中表的第 3 行）。像这样，我们可以先根据接收方和发送方地址判断包的流向，并设置是允许还是阻止。

<sup>①</sup> 详见第 2 章的内容。

### 5.2.3 通过端口号限定应用程序

不过，按照前面的设置，相当于允许了互联网和 Web 服务器之间所有的包通过，这个状态很危险。假如服务器上还有一个文件服务器程序在工作，那么这些文件就可能会被非法访问而造成信息泄露。有风险的还不仅是文件服务器，现在每天都会发布若干安全漏洞，可以说随处都隐藏着风险。因此，我们最好是阻止除了必需服务（也就是本例中的 Web 服务）以外的所有应用程序的包。

当我们要限定某个应用程序时，可以在判断条件中加上 TCP 头部或者 UDP 头部中的端口号。Web 服务器的端口号为 80<sup>①</sup>，因此我们在刚才的接收方 IP 地址和发送方 IP 地址的基础上再加上 80 端口作为条件就可以了。也就是说，当包的接收方 IP 地址为 Web 服务器地址，且接收方端口号为 80 时，允许这些包通过（图 5.2 中表的第 1 行）；或者当包的发送方 IP 地址为 Web 服务器地址，且发送方端口号为 80 时，允许这些包通过（图 5.2 中的表的第 3 行）。如果要允许访问除 Web 之外的其他应用程序，则只要将该应用程序的端口号设置到防火墙中并允许通过就可以了。

### 5.2.4 通过控制位判断连接方向

现在我们已经可以指定某个具体的应用程序了，但是条件还没达到，因为还没有办法阻止 Web 服务器访问互联网。Web 使用的 TCP 协议是双向收发网络包的，因此如果单纯地阻止从 Web 服务器发往互联网的包，则从互联网访问 Web 服务器的操作也会受到影响而无法进行。光判断包的流向还不够，我们必须要根据访问的方向来进行判断。这里就需要用到 TCP 头部中的控制位。TCP 在执行连接操作时需要收发 3 个包<sup>②</sup>，其中第一个包

① 也可以不使用 80 端口而使用其他端口，但这种情况一定是在 Web 服务器程序中特别设置过的，因此只要按照服务器的设置来调整防火墙设置就可以了。

② 第 2 章有具体介绍。

的 TCP 控制位中 SYN 为 1，而 ACK 为 0。其他的包中这些值都不同，因此只要按照这个规则就能够过滤到 TCP 连接的第一个包。

如果这第一个包是从 Web 服务器发往互联网的，那么我们就阻止它（图 5.2 表中的第 2 行）。这样设置之后，当然也不会收到对方返回的第二个响应包，TCP 连接操作就失败了。也就是说，只要以 Web 服务器为起点访问互联网，其连接操作必然会失败，这样一来，我们就阻止了 Web 服务器对互联网的访问。

那么，从互联网访问 Web 服务器会不会受影响呢？从互联网访问 Web 服务器时，第一个包是接收方为 Web 服务器，符合图 5.2 表中的第 1 行，因此允许通过。第二个包的发送方是 Web 服务器，但 TCP 控制位的规则与第二行不匹配<sup>①</sup>，因此符合第三行的规则，允许通过。随后的所有包要么符合第一行，要么符合第三行，因此从互联网访问 Web 服务器的所有包都会被允许通过。

通过接收方 IP 地址、发送方 IP 地址、接收方端口号、发送方端口号、TCP 控制位这些条件，我们可以判断出通信的起点和终点、应用程序种类，以及访问的方向。当然，如表 5.1 列出的那样，还有很多其他的字段可以用作判断条件。通过对这些条件进行组合，我们就可以对包进行筛选。这里也可以添加多个规则，直到能够将允许的访问和不允许的访问完全区分开为止。这样，我们就可以通过设置规则，让允许访问的包通过防火墙，其他的包则不能通过防火墙<sup>②</sup>。

不过，实际上也存在无法将希望允许和阻止的访问完全区分开的情况，其中一个代表性的例子就是对 DNS 服务器的访问。DNS 查询使用的是 UDP 协议，而 UDP 与 TCP 不同，它没有连接操作，因此无法像 TCP 一样根据控制位来判断访问方向。所以，我们无法设置一个规则，只允许公司

① 第二个包中，SYN 为 1，ACK 也为 1，因此不符合刚刚设定的规则。

② 还有一种思路是只阻止有风险的访问，允许其他所有的访问，这种方法有可能漏掉未知的有风险的网路包。因此，为了避免未知的风险，一般来说都是只允许必要的包通过，其他的包全部阻止。

内部访问互联网上的 DNS 服务器，而阻止从互联网访问公司内部 DNS 服务器。这一性质不仅适用于 DNS，对于所有使用 UDP 协议的应用程序都是共通的。在这种情况下，只能二者择其一——要么冒一定的风险允许该应用程序的所有包通过，要么牺牲一定的便利性阻止该应用程序的所有包通过<sup>①</sup>。

### 5.2.5 从公司内网访问公开区域的规则

图 5.2 这样的网络结构中，我们不仅要设置互联网和公开区域之间的包过滤规则，还需要设置公司内网和互联网之间，或者公司内网与公开区域之间的包过滤规则。这时，需要注意的是不要让这些规则互相干扰。例如，为了让公司内网与公开区域之间的网络包自由流动，我们可以将接收方 IP 地址为公开区域的包设置成全部允许通过。但是，如果在这条规则里没有限定发送方 IP 地址，那么连来自互联网的包也都会被无条件允许进入公开区域了，这会导致公开区域中的服务器全部暴露在危险状态中。因此，我们必须谨慎地设置规则，防止出现这样的情况。

### 5.2.6 从外部无法访问公司内网

包过滤方式的防火墙不仅可以允许或者阻止网络包的通过，还具备地址转换功能<sup>②</sup>，因此还需要进行相关的设置。也就是说，互联网和公司内网之间的包需要进行地址转换才能传输，因此必须要进行相关的设置<sup>③</sup>。具体

- 
- ① 如果是使用包过滤之外的其他方式的防火墙，有时候是可以判断 UDP 应用程序的访问方向的。
  - ② 关于地址转换请参见第 3 章。
  - ③ 互联网路由器的路由表中没有私有地址的路由信息，因此凡是接收方为私有地址的包，在经过互联网中的路由器时都会被丢弃，这就是为什么必须使用地址转换的原因。相对地，防火墙内置的路由功能可以由用户自行设置，因此可以在路由表中配置私有地址相关的路由，使得公司内网到公开区域的访问可以以私有地址的形式来进行，这意味着公司内网和公开区域之间传输的包不需要地址转换。

来说, 就是和包过滤一样, 以起点和终点作为条件, 根据需要设置是否需要地址转换。私有地址和公有地址之间的对应关系, 以及端口号的对应关系都是自动管理的, 因此只需要设置是否允许地址转换就可以了。

请大家回忆一下地址转换的工作原理, 当使用地址转换时, 默认状态下是无法从互联网访问公司内网的, 因此我们不需要再设置一条包过滤规则来阻止从互联网访问公司内网。

### 5.2.7 通过防火墙

像这样, 我们可以在防火墙中设置各种规则, 当包到达防火墙时, 会根据这些规则判断是允许通过还是阻止通过。


如果判断结果为阻止, 那么这个包会被丢弃并被记录下来<sup>①</sup>。这是因为这些被丢弃的包中通常含有非法入侵的痕迹, 通过分析这些包能够搞清楚入侵者使用的手法, 从而帮助我们更好地防范非法入侵。

如果包被判断为允许通过, 则该包会被转发出去, 这个转发的过程和路由器是相同的。如果我们只关注判断是否允许包通过这一点, 可能会觉得防火墙是一种特殊机制, 而且市面上销售的防火墙大多是专用的硬件设备或者软件, 这也加深了大家的这种印象。

实际上, 在防火墙允许包通过之后, 就没有什么特别的机制了, 因此包过滤并不是防火墙专用的一种特殊机制, 而是应该看作在路由器的包转发功能基础上附加的一种功能。只不过当判断规则比较复杂时, 通过路由器的命令难以维护这些规则, 而且对阻止的包进行记录对于路由器来说负担也比较大, 因此才出现了专用的硬件和软件。如果规则不复杂, 也不需要记录日志, 那么用内置包过滤功能的普通路由器来充当防火墙也是可以的。

---

① 如果将内置包过滤功能的路由器用作防火墙, 则在丢弃包时基本上不会留下记录, 这是因为路由器的内存容量小, 没有足够的空间用来记录日志。



包过滤方式的防火墙可根据接收方 IP 地址、发送方 IP 地址、接收方端口号、发送方端口号、控制位等信息来判断是否允许某个包通过。

### 5.2.8 防火墙无法抵御的攻击

防火墙可以根据包的起点和终点来判断是否允许其通过，但仅凭起点和终点并不能筛选出所有有风险的包。比如，假设 Web 服务器在收到含有特定数据的包时会引起宕机。但是防火墙只关心包的起点和终点，因此即便包中含有特定数据，防火墙也无法发现，于是包就被放行了。然后，当包到达 Web 服务器时，就会引发服务器宕机。通过这个例子大家可以看出，只有检查包的内容才能识别这种风险，因此防火墙对这种情况无能为力。

要应对这种情况有两种方法。这个问题的根源在于 Web 服务器程序的 Bug，因此修复 Bug 防止宕机就是其中一种方法。这类 Bug 中，危险性较高的会作为安全漏洞公布出来，开发者会很快发布修复了 Bug 的新版本，因此持续关注安全漏洞信息并更新软件的版本是非常重要的。

另一种方法就是在防火墙之外部署用来检查包的内容并阻止有害包的设备或软件<sup>①</sup>。当然，即便是采用这种方法也并不是完美无缺的，因为包的内容是否有风险，是由 Web 服务器有没有 Bug 决定的，因此当服务器程序中有潜在的 Bug 并且尚未被发现时，我们也无法判断包中的风险，也无法阻止这样的包。也就是说，我们无法抵御未知的风险。从这一点来看，这种方法和直接修复 Bug 的方法是基本等效的，但如果服务器数量较多，更新软件版本需要花费一定的时间，或者容易忘记更新软件，这时对包的内容进行检查就会比较有效。

---

① 有时会作为防火墙的附件提供。

## 5.3

## 通过将请求平均分配给多台服务器来平衡负载

## 5.3.1 性能不足时需要负载均衡

当服务器的访问量上升时，增加服务器线路的带宽是有效的，但并不是网络变快了就可以解决所有的问题。高速线路会传输大量的网络包，这会导致服务器的性能跟不上<sup>①</sup>。尤其是通过 CGI 等应用程序动态生成数据的情况下，对服务器 CPU 的负担更重，服务器性能的问题也会表现得越明显。

要解决这个问题，大家可能首先想到的是换一台性能更好的服务器，但当很多用户同时访问时，无论服务器的性能再好，仅靠一台服务器还是难以胜任的。在这种情况下，使用多台服务器来分担负载的方法更有效。这种架构统称为分布式架构，其中对于负载的分担有几种方法，最简单的一种方法就是采用多台 Web 服务器，减少每台服务器的访问量。假设现在我们有 3 台服务器，那么每台服务器的访问量会减少到三分之一，负载也就减轻了。要采用这样的方法，必须有一个机制将客户端发送的请求分配到每台服务器上。具体的做法有很多种，最简单的一种是通过 DNS 服务器来分配。当访问服务器时，客户端需要先向 DNS 服务器查询服务器的 IP 地址，如果在 DNS 服务器中填写多个名称相同的记录，则每次查询时 DNS 服务器都会按顺序返回不同的 IP 地址。例如，对于域名 `www.lab.glasscom.com`，如果我们给它分配如下 3 个 IP 地址。

192.0.2.60

192.0.2.70

192.0.2.80

当第 1 次查询这个域名时，服务器会返回如下内容。

192.0.2.60 192.0.2.70 192.0.2.80

① 无论服务器部署在公司里还是数据中心里，这个问题都是共通的。



当第 2 次查询时，服务器会返回如下内容。

192.0.2.70 192.0.2.80 192.0.2.60

当第 3 次查询时，服务器会返回如下内容。

192.0.2.80 192.0.2.60 192.0.2.70

当第 4 次查询时就又回到第 1 次查询的结果(图 5.3)。这种方式称为轮询(round-robin)，通过这种方式可以将访问平均分配给所有的服务器。

但这种方式是有缺点的。假如多台 Web 服务器中有一台出现了故障，这时我们希望在返回 IP 地址时能够跳过故障的 Web 服务器，然而普通的 DNS 服务器并不能确认 Web 服务器是否正常工作，因此即便 Web 服务器宕机了，它依然可能会返回这台服务器的 IP 地址<sup>①</sup>。

此外，轮询分配还可能会引发一些问题。在通过 CGI 等方式动态生成网页的情况下，有些操作是要跨多个页面的，如果这期间访问的服务器发生了变化，这个操作就可能无法继续。例如在购物网站中，可能会在第一个页面中输入地址和姓名，在第二个页面中输入信用卡号，这就属于刚才说的那种情况。

### 5.3.2 使用负载均衡器分配访问

为了避免出现前面的问题，可以使用一种叫作负载均衡器的设备。使用负载均衡器时，首先要用负载均衡器的 IP 地址代替 Web 服务器的实际地址注册到 DNS 服务器上。假设有一个域名 `www.lab.glasscom.com`，我们将这个域名对应的 IP 地址设置为负载均衡器的 IP 地址并注册到 DNS 服务器上。于是，客户端会认为负载均衡器就是一台 Web 服务器，并向其发送

---

① 如果浏览器在访问 DNS 服务器返回的第一个 IP 地址失败时，能够继续尝试第二个 IP 地址，就可以回避这个问题了，最近的浏览器有很多都已经具备了这样的功能。



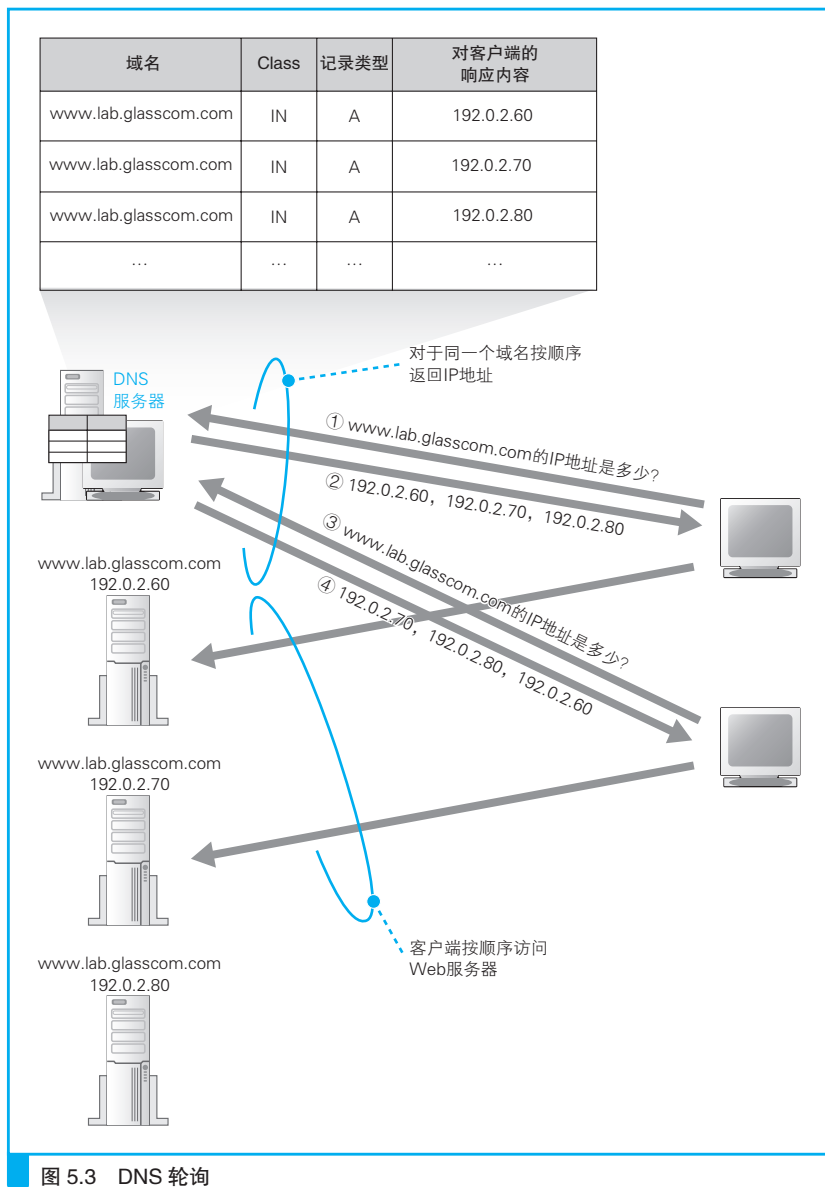


图 5.3 DNS 轮询

请求，然后由负载均衡器来判断将请求转发给哪台 Web 服务器（图 5.4）<sup>①</sup>。这里的关键点不言而喻，那就是如何判断将请求转发给哪台 Web 服务器。

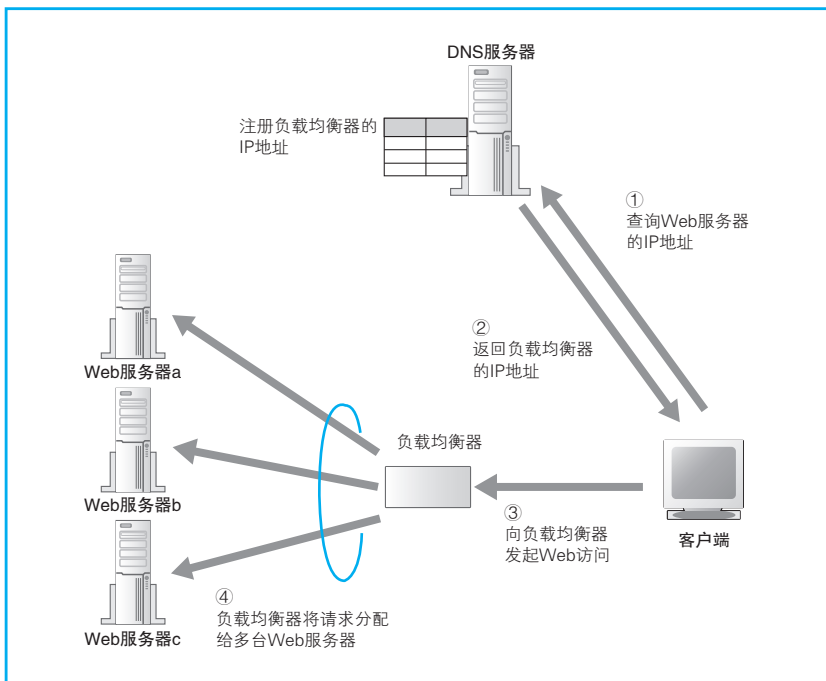


图 5.4 用于对多台 Web 服务器分配访问的负载均衡器

判断条件有很多种，根据操作是否跨多个页面，判断条件也会有所不同。如果操作没有跨多个页面，则可以根据 Web 服务器的负载状况来进行判断。负载均衡器可以定期采集 Web 服务器的 CPU、内存使用率，并根据这些数据判断服务器的负载状况，也可以向 Web 服务器发送测试包，根据响应所需的时间来判断负载状况。当然，Web 服务器的负载可能会在短时间内上下波动，因此无法非常准确地把握负载状况，反过来说，如果过于

① 转发请求消息使用的是后面要讲到的“代理”机制，缓存服务器也使用这种机制。此外，有些负载均衡器中也内置缓存功能。负载均衡器和缓存服务器很相似，或者说它是由缓存服务器进一步发展而来的。

密集地去查询服务器的负载，这个查询操作本身就会增加 Web 服务器的负载。因此也有一种方案是不去查询服务器的负载，而是根据事先设置的服务器性能指数，按比例来分配请求。无论如何，这些方法都能够避免负载集中在某一台服务器上。

当操作跨多个页面时，则不考虑 Web 服务器的负载，而是必须将请求发送到同一台 Web 服务器上。要实现这一点，关键在于我们必须判断一个操作是否跨了多个页面。HTTP 的基本工作方式是在发送请求消息之前先建立 TCP 连接，当服务器发送响应消息后断开连接，下次访问 Web 服务器的时候，再重新建立 TCP 连接<sup>①</sup>。因此，在 Web 服务器看来，每一次 HTTP 访问都是相互独立的，无法判断是否和之前的请求相关。

之所以会这样，是因为 Web 中使用的 HTTP 协议原本就是这样设计的。如果要判断请求之间的相关性，就必须在 Web 服务器一端保存相应的信息，这会增加服务器的负担。此外，Web 服务器最早并不是用来运行 CGI 程序的，而是主要用来提供静态文件的，而静态文件不需要判断请求之间的相关性，因此最早设计 HTTP 规格的时候，就有意省略了请求之间相关性的判断。

那么在不知道请求之间的相关性时，能不能根据一系列请求的发送方 IP 地址相同这一点来判断呢？也不行。如果使用了我们后面要讲的代理机制<sup>②</sup>，所有请求的发送方 IP 地址都会变成代理服务器的 IP 地址，无法判断实际发送请求的客户端是哪个。此外，如果使用了地址转换，发送方 IP 地址则会变成地址转换设备的 IP 地址，也无法判断具体是哪个客户端。

于是，人们想出了一些方案来判断请求之间的相关性。例如，可以在发送表单数据时在里面加上用来表示关联的信息，或者是对 HTTP 规格进

- 
- ① 现在越来越多的服务器在发送响应消息之后会等待一段时间再断开连接，这个等待时间大约只有几秒钟，像购物网站这种跨多页面填写信息的场景已经超过了这个等待时间，因此还是会断开连接。
- ② 代理：一种介于客户端与 Web 服务器之间，对访问操作进行中转的机制。这部分内容稍后会在 5.4 节进行介绍。

行扩展，在 HTTP 头部字段中加上用来判断相关性的信息<sup>①</sup>。这样，负载均衡器就可以通过这些信息来作出判断，将一系列相关的请求发送到同一台 Web 服务器，对于不相关的请求则发送到负载较低的服务器了。

## 5.4

## 使用缓存服务器分担负载

### 5.4.1

### 如何使用缓存服务器

除了使用多台功能相同的 Web 服务器分担负载之外，还有另外一种方法，就是将整个系统按功能分成不同的服务器<sup>②</sup>，如 Web 服务器、数据库服务器。缓存服务器就是一种按功能来分担负载的方法。

缓存服务器是一台通过代理机制对数据进行缓存的服务器。代理介于 Web 服务器和客户端之间，具有对 Web 服务器访问进行中转的功能。当进行中转时，它可以将 Web 服务器返回的数据保存在磁盘中，并可以代替 Web 服务器将磁盘中的数据返回给客户端。这种保存的数据称为缓存，缓存服务器指的也就是这样的功能。

Web 服务器需要执行检查网址和访问权限，以及在页面上填充数据等内部操作过程，因此将页面数据返回客户端所需的时间较长。相对地，缓存服务器只要将保存在磁盘上的数据读取出来发送给客户端就可以了，因此可以比 Web 服务器更快地返回数据。

不过，如果在缓存了数据之后，Web 服务器更新了数据，那么缓存的数据就不能用了，因此缓存并不是永久可用的。此外，CGI 程序等产生的页面数据每次都不同，这些数据也无法缓存。无论如何，在来自客户端的访问中，总有一部分访问可以无需经过 Web 服务器，而由缓存服务器直接处理。即便只有这一部分操作通过缓存服务器提高了速度，整体性能也可

① 这种信息俗称 Cookie。

② 也可以将“使用多台功能相同的服务器”和“使用多台功能不同的服务器”这两种方法结合起来使用。

以得到改善。此外，通过让缓存服务器处理一部分请求，也可以减轻 Web 服务器的负担，从而缩短 Web 服务器的处理时间。



### 5.4.2 缓存服务器通过更新时间管理内容

下面来看一看缓存服务器的工作过程<sup>①</sup>。缓存服务器和负载均衡器一样，需要代替 Web 服务器被注册到 DNS 服务器中。然后客户端会向缓存服务器发送 HTTP 请求消息（图 5.5 (a) ①、图 5.6 (a)）。这时，缓存服务器会接收请求消息，这个接收操作和 Web 服务器相同。Web 服务器的接收操作我们会在第 6 章的 6.2 节进行介绍<sup>②</sup>，简单来说就是创建用来等待连接的套接字，当客户端进行连接时执行连接操作，然后接收客户端发送的请求消息。从客户端来看，缓存服务器就相当于 Web 服务器。接下来，缓存服务器会检查请求消息的内容，看看请求的数据是否已经保存在缓存中。根据是否存在缓存数据，后面的操作会有所不同，现在我们假设不存在缓存数据。这时，缓存服务器会像图 5.6 (b) ②这样，在 HTTP 头部字段中添加一个 Via 字段，表示这个消息经过缓存服务器转发，然后将消息转发给 Web 服务器（图 5.5 (a) ②）。

在这个过程中，我们需要判断应该将请求消息转发给哪台 Web 服务器。如果只有一台 Web 服务器，那么情况比较简单，只要将 Web 服务器的域名和 IP 地址配置在缓存服务器上，让它无条件转发给这台服务器就可以了。不过，如果一台缓存服务器对应多台 Web 服务器就没那么简单了，需要根据请求消息的内容来判断应该转发给哪台 Web 服务器。要实现这个目的有几种方法，其中比较有代表性的是根据请求消息的 URI（图 5.6 (b) ①）中的目录名来进行判断。使用这种方法时，我们首先需要在缓存服务器上进行如下设置。

① 要理解缓存服务器的工作过程，需要先理解 Web 服务器和 HTTP 协议，这些内容在第 1 章进行了介绍。

② 数据收发操作的基本知识在第 2 章也有相关介绍。

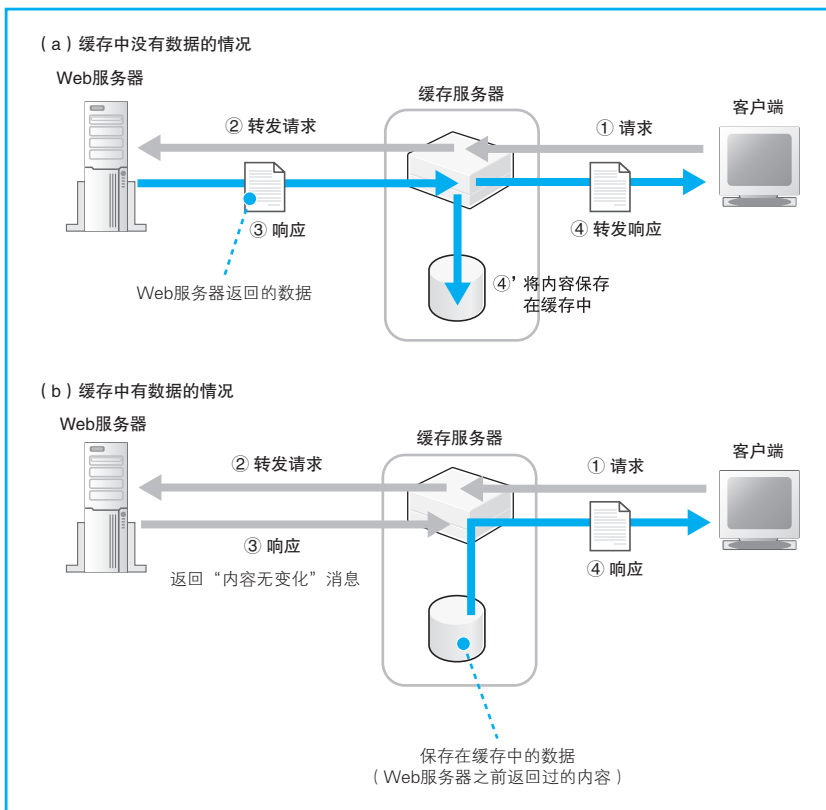


图 5.5 临时保存内容并代替 Web 服务器返回内容的缓存服务器

- 当 URI 为 /dir1/ 这个目录时，转发给 `www1.lab.glasscom.com`
- 当 URI 为 /dir2/ 这个目录时，转发给 `www2.lab.glasscom.com`

缓存服务器会根据上述规则来转发请求消息，在这个过程中，缓存服务器会以客户端的身份向目标 Web 服务器发送请求消息。也就是说，它会先创建套接字，然后连接到 Web 服务器的套接字，并发送请求消息。从 Web 服务器来看，缓存服务器就相当于客户端。于是，缓存服务器会收到来自 Web 服务器的响应消息（图 5.5 (a) ③、图 5.6 (c)），接收消息的过程也是以客户端的身份来完成的。

## (a) 客户端发送给缓存服务器的请求内容 (图5.5 (a) ①)

客户端发送给缓存服务器的请求和一般的请求相同。

```
GET /dir1/sample1.html HTTP/1.1
Accept: */*
Accept-Language: zh
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; 【右侧省略】
Host: www.lab.glasscom.com
Connection: Keep-Alive
```

## (b) 缓存服务器转发给Web服务器的请求内容 (图5.5 (a) ②)

添加了表示经过缓存服务器中转的头部字段, 通过URI判断转发目标。

```
GET /dir1/sample1.html HTTP/1.1
Accept: */*
Accept-Language: zh
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; 【右侧省略】
Host: www.lab.glasscom.com
Connection: Keep-Alive
Via: 1.1 proxy.lab.glasscom.com
```

②

Via用于告知Web服务器这个消息是经过缓存服务器中转的。这个信息并不是非常重要, 因此根据缓存服务器的配置, 有时不会添加这个字段

①

根据URI中的目录判断转发目标Web服务器

## (c) Web服务器返回给缓存服务器的响应内容 (图5.5 (a) ③)

当没有If-Modified-Since字段时 (即缓存中没有数据时), 或者Web服务器端数据发生变化时, 直接返回页面数据, 内容和通常情况下相同

```
HTTP/1.1 200 OK
Date: Wed, 21 Feb 2007 12:20:40 GMT
Server: Apache
Last-Modified: Mon, 19 Feb 2007 12:24:51 GMT
ETag: "5a9da-279-3c726b61"
Accept-Ranges: bytes
Content-Length: 632
Connection: close
Content-Type: text/html

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; 【右侧省略】
【以下省略】
```

图 5.6 缓存中没有数据的情况