（d）缓存服务器转发给客户端的响应内容（图5.5（a）④）

除了因经过缓存服务器中转而多了Via字段之外，其他内容与一般的响应消息相同。

```
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
Via: 1.1 proxy.lab.glasscom.com

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

【以下省略】
```

①

Via用于告知客户端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

图 5.6 （续）

接下来，缓存服务器会在响应消息中加上图 5.6（d）①这样的 Via 头部字段，它表示这个消息是经过缓存服务器中转的，然后缓存服务器会以 Web 服务器的身份向客户端发送响应消息（图 5.5（a）④）。同时，缓存服务器会将响应消息保存到缓存中，并记录保存的时间（图 5.5（a）④'）。

这种在客户端和 Web 服务器之间充当中间人的方式就是代理的基本原理。在中转消息的过程中，缓存服务器还会顺便将页面数据保存下来，随着缓存数据的积累，用户访问的数据命中缓存的几率也会提高。接下来我们来看一看命中缓存的情况（图 5.5（b））。

首先，接收客户端的请求消息并检查缓存的过程和刚才是一样的（图 5.5（b）①、图 5.6（a））。然后，如图 5.7（a），缓存服务器会添加一个 If-Modified-Since 头部字段并将请求转发给 Web 服务器，询问 Web 服务器用户请求的数据是否已经发生变化（图 5.5（b）②、图 5.7（a））。

然后，Web 服务器会根据 If-Modified-Since 的值与服务器上的页面数据的最后更新时间进行比较，如果在指定时间内数据没有变化，就会返回一个像图 5.7（b）一样的表示没有变化的响应消息（图 5.5（b）③）。这时，Web 服务器只要查询一下数据的最后更新时间就好了，比返回页面数据的

负担要小一些。而且返回的响应消息也比较短，能相应地减少负担。接下来，返回消息到达缓存服务器，然后缓存服务器就会知道 Web 服务器上的数据和本地缓存中的数据是一样的，于是就会将缓存的数据返回给客户端（图 5.5（b）④）。缓存服务器返回的响应消息的内容和没有命中缓存的情况是一样的（图 5.6（d））。

此外，当 Web 服务器上的数据有变化时，后面的过程和没有命中缓存的情况是一样的。Web 服务器会返回最新版本的数据（图 5.5（a）③、图 5.6（c）），然后缓存服务器加上 Via 字段发送给客户端，同时将数据保存在缓存中。

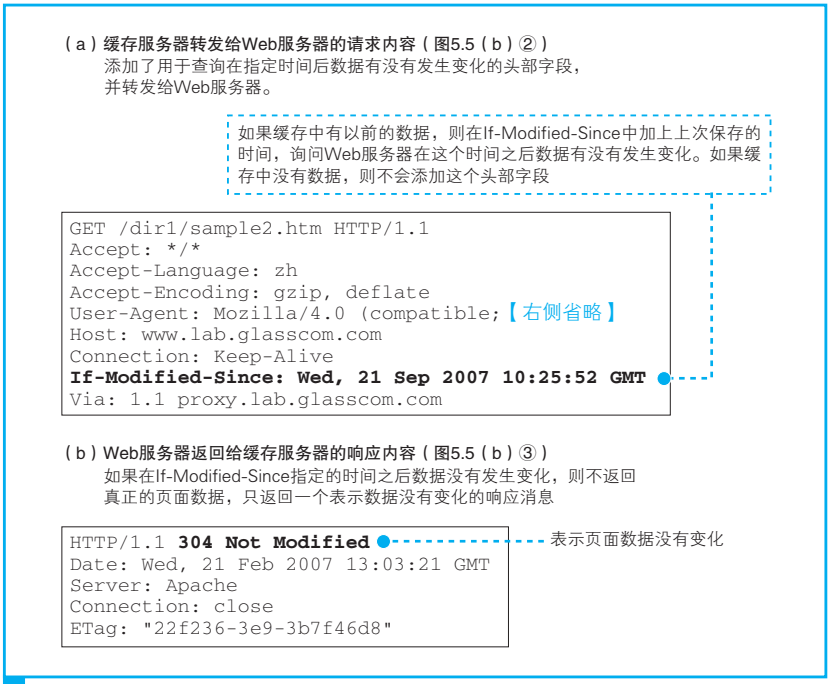


图 5.7　缓存中有数据的情况

5.4.3　最原始的代理——正向代理

刚才讲的是在 Web 服务器一端部署一个代理，然后利用其缓存功能来改善服务器的性能，还有一种方法是在客户端一侧部署缓存服务器。下面先稍微脱离一下主线，介绍一下客户端一侧的缓存服务器。

实际上，缓存服务器使用的代理机制最早就是放在客户端一侧的，这才是代理的原型，称为正向代理[①]（forward proxy）。

正向代理刚刚出现的时候，其目的之一就是缓存，这个目的和服务器端的缓存服务器相同。不过，当时的正向代理还有另外一个目的，那就是用来实现防火墙。

防火墙的目的是防止来自互联网的非法入侵，而要达到这个目的，最可靠的方法就是阻止互联网和公司内网之间的所有包。不过，这样一来，公司员工就无法上外网了，因此还必须想一个办法让必要的包能够通过，这个办法就是利用代理。简单来说，代理的原理如图 5.8 所示，它会先接收来自客户端的请求消息，然后再转发到互联网中[②]，这样就可以实现只允许通过必要的网络包了。这时，如果能够利用代理的缓存，那么效果就会更好，因为对于以前访问过的数据，可以直接从位于公司内网的代理服务器获得，这比通过低速线路访问互联网要快很多[③]。

---

① 其实正向代理并不是一开始就叫这个名字，最早说的“代理”指的就是我们现在说的正向代理，或者也叫“代理服务器”。这是因为最早只有这么一种代理，后来出现了各种其他方式的代理，为了相互区别才起了“××代理”这样的名字。此外，由于代理种类变多了，叫“××代理服务器”实在太长，一般都会省略“服务器”3 个字。

② 代理（Proxy）本来的意思并不是“转发”消息，而是先把消息收下来，然后“伪装”成原始客户端向 Web 服务器发出访问请求。

③ 代理出现于 ADSL、FTTH 等技术实用化之前，那个时候还没有廉价高速的接入网，因此必须想办法榨干低速接入网中的所有能力。代理的缓存功能正是有效利用低速接入网的一种方法。

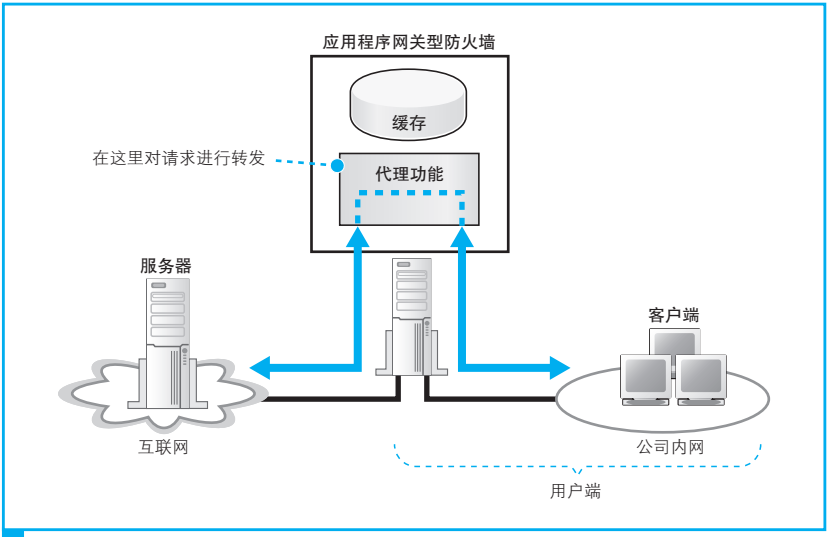


图 5.8 利用代理实现防火墙

此外，由于代理在转发过程中可以查看请求的内容，所以可以根据内容判断是否允许访问。也就是说，通过代理可以禁止员工访问危险的网站，或者是与工作内容无关的网站。包过滤方式的防火墙只能根据 IP 地址和端口号进行判断，因此无法实现这一目的。

在使用正向代理时，一般需要在浏览器的设置窗口中的“代理服务器”一栏中填写正向代理的 IP 地址，浏览器发送请求消息的过程也会发生相应的变化。在没有设置正向代理的情况下，浏览器会根据网址栏中输入的 http://... 字符串判断 Web 服务器的域名，并向其发送请求消息；当设置了正向代理时，浏览器会忽略网址栏的内容，直接将所有请求发送给正向代理。请求消息的内容也会有一些不同。没有正向代理时，浏览器会从网址中提取出 Web 服务器域名后面的文件名或目录名，然后将其作为请求的 URI 进行发送；而有正向代理时，浏览器会像图 5.9 这样，在请求的 URI 字段中填写完整的 http://... 网址。

正向代理转发消息的过程也和服务器端的缓存服务器有一些不同，不

同点在于对转发目标 Web 服务器的判断上。使用正向代理时，URI 部分为 http://... 这样的完整网址，因此可以根据这个网址来转发，不需要像服务器端的缓存服务器一样实现设置好转发目标 Web 服务器，而且可以发给任意 Web 服务器。而服务器端的缓存服务器只能向事先设置好的目标进行转发，这就是两者不同的地方。

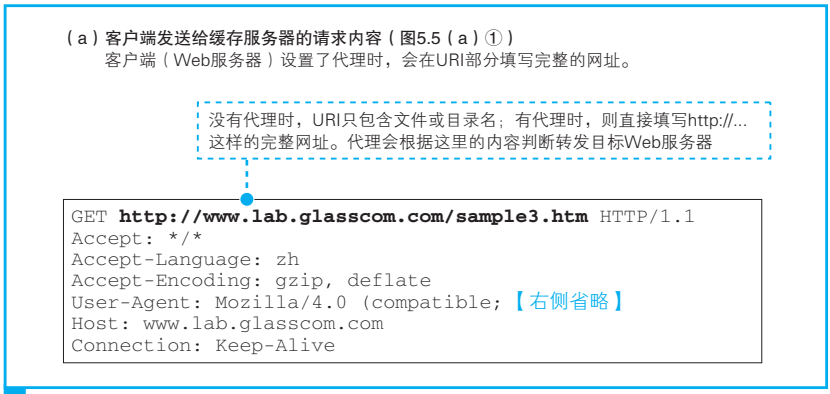


图 5.9　正向代理的 HTTP 消息

5.4.4　正向代理的改良版——反向代理

正如前面讲过的，使用正向代理需要在浏览器中进行设置，这可以说是识别正向代理的一个特征。但是，设置浏览器非常麻烦，如果设置错误还可能导致浏览器无法正常工作。

需要设置浏览器这一点除了麻烦、容易发生故障之外，还有其他一些限制。如果我们想把代理放在服务器端，那么服务器不知道谁会来访问，也没办法去设置客户端的浏览器，因此无法采用这种方法来实现。

于是，我们可以对这种方法进行改良，使得不需要在浏览器中设置代理也可以使用。也就是说，我们可以通过将请求消息中的 URI 中的目录名与 Web 服务器进行关联，使得代理能够转发一般的不包含完整网址的请求消息。我们前面介绍的服务器端的缓存服务器采用的正是这种方式，这种

方式称为反向代理（reverse proxy）。

5.4.5 透明代理

缓存服务器判断转发目标的方法还有一种，那就是查看请求消息的包头部。因为包的 IP 头部中包含接收方 IP 地址，只要知道了这个地址，就知道用户要访问哪台服务器了[①]。这种方法称为透明代理（transparent proxy）。

这种方法也可以转发一般的请求消息，因此不需要像正向代理一样设置浏览器参数，也不需要在缓存服务器上设置转发目标，可以将请求转发给任意 Web 服务器。

透明代理集合了正向代理和反向代理的优点，是一个非常方便的方式，但也需要注意一点，那就是如何才能让请求消息到达透明代理。由于透明代理不需要设置在浏览器中，那么浏览器还是照常向 Web 服务器发送请求消息。反向代理采用的是通过 DNS 服务器解析引导的方法，但透明代理是不能采用这种方法的，否则透明代理本身就变成了访问目标，也就无法通过接收方 IP 地址判断转发目标了，这就失去了透明代理的意义。总之，正常情况下，请求消息是从浏览器直接发送到 Web 服务器，并不会到达透明代理。

于是，我们必须将透明代理放在请求消息从浏览器传输到 Web 服务器的路径中，当消息经过时进行拦截。可能大家觉得这种方法太粗暴，但只有这样才能让消息到达透明代理，然后再转发给 Web 服务器。如果请求消息有多条路径可以到达 Web 服务器，那么就必须在这些路径上都放置透明代理，因此一般是将网络设计成只有一条路可以走的结构，然后在这一条路径上放置透明代理。连接互联网的接入网就是这样一个关口，因此可以在接入网的入口处放置反向代理[②]。使用透明代理时，用户不会察觉到代理

① HTTP 1.1 版本增加了一个用于表示访问目标 Web 服务器的 Host 字段，因此也可以通过 Host 字段来判断转发目标。

② 也可以采用在网络中的某些地方将 Web 访问包筛选出来并转发给透明代理的方法。

的存在，也不会注意到 HTTP 消息是如何被转发的，因此大家更倾向于将透明代理说成是缓存[①]。

5.5 内容分发服务

5.5.1 利用内容分发服务分担负载

缓存服务器部署在服务器端还是客户端，其效果是有差别的。如图 5.10（a）所示，当缓存服务器放在服务器端时，可以减轻 Web 服务器的负载，但无法减少互联网中的流量。这一点上，将缓存服务器放在客户端更有效（图 5.10（b））。互联网中会存在一些拥塞点，通过这些地方会比较花时间。如果在客户端部署缓存服务器，就可以不受或者少受这些拥塞点的影响，让网络流量更稳定，特别是当访问内容中含有大图片或视频时效果更明显。

不过，客户端的缓存服务器是归客户端网络运营管理者所有的，Web 服务器的运营者无法控制它。比如，某网站的运营者觉得最近网站上增加了很多大容量的内容，因此想要增加缓存服务器的容量。如果缓存放在服务器端，那么网站运营者可以自己通过增加磁盘空间等方式来进行扩容，但对于放在客户端的缓存就无能为力了。进一步说，客户端有没有缓存服务器还不一定呢。

因此，这两种部署缓存服务器的方式各有利弊，但也有一种方式能够集合两者的优点。那就是像图 5.10（c）这样，Web 服务器运营者和网络运营商签约，将可以自己控制的缓存服务器放在客户端的运营商处。

这样一来，我们可以把缓存服务器部署在距离用户很近的地方，同时 Web 服务器运营者还可以控制这些服务器，但这种方式也有问题。对于在互联网上公开的服务器来说，任何地方的人都可以来访问它，因此如果真的要实现这个方式，必须在所有的运营商 POP 中都部署缓存服务器才行，

① 最近有很多场合中已经将透明代理直接叫作“缓存”而不是“代理”了，不过无论叫什么名字，其内部结构是相同的。

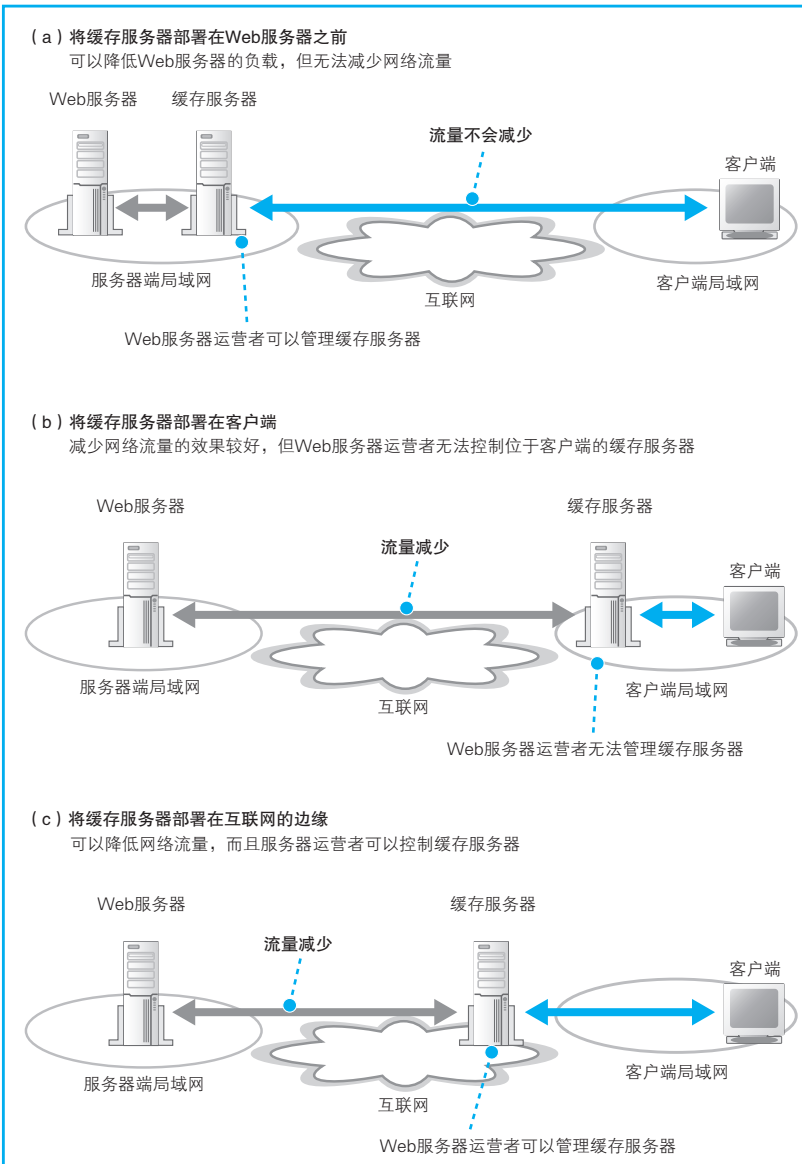


图 5.10 缓存服务器的 3 种部署方式

这个数量太大了，非常不现实。

要解决这个问题也有一些办法。首先，我们可以筛选出一些主要的运营商，这样可以减少缓存服务器的数量。尽管这样做可能会导致有些用户访问到缓存服务器还是要经过很长的距离，但总比直接访问 Web 服务器的路径要短多了，因此还是可以产生一定的效果。

接下来这个问题更现实，那就是即便减少了数量，作为一个 Web 服务器运营者，如果自己和这些运营商签约并部署缓存服务器，无论是费用还是精力都是吃不消的。为了解决这个问题，一些专门从事相关服务的厂商出现了，他们来部署缓存服务器，并租借给 Web 服务器运营者。这种服务称为内容分发服务[①]。下面我们来具体了解一下这种服务。

提供这种服务的厂商称为 CDSP[②]，他们会与主要的供应商签约，并部署很多台缓存服务器[③]。另一方面，CDSP 会与 Web 服务器运营者签约，使得 CDSP 的缓存服务器配合 Web 服务器工作。具体的方法我们后面会介绍，只要 Web 服务器与缓存服务器建立关联，那么当客户端访问 Web 服务器时，实际上就是在访问 CDSP 的缓存服务器了。

缓存服务器可以缓存多个网站的数据，因此 CDSP 的缓存服务器就可以提供给多个 Web 服务器的运营者共享。这样一来，每个网站运营者的平均成本就降低了，从而减少了网站运营者的负担。而且，和运营商之间的签约工作也由 CDSP 统一负责，网站运营者也节省了精力。

5.5.2　如何找到最近的缓存服务器

在使用内容分发服务时，如图 5.11 所示，互联网中有很多缓存服务器，如何才能从这些服务器中找到离客户端最近的一个，并让客户端去访问那台服务器呢？

---

① 内容分发服务也叫 CDS（Content Delivery Service）。（现在更常用的名称叫 CDN（Content Delivery Network 或 Content Distribution Network）。——译者注）

② CDSP：Content Delivery Service Provider，内容分发服务运营商。

③ 有些 CDSP 会在互联网中部署几百台缓存服务器。

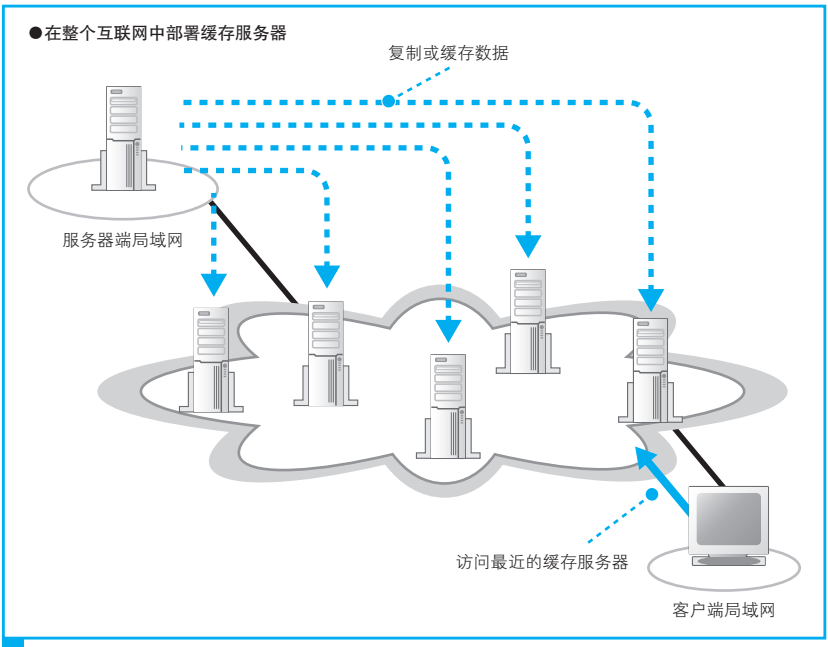


图 5.11 访问目标服务器的所在地

我们可以像正向代理一样在浏览器中进行设置，但用户那么多，也没办法帮所有人去设置浏览器。因此，我们需要一种机制，即便用户不进行任何设置，也能够将请求消息发送到缓存服务器。

这样的方法有几种，下面我们按顺序来介绍。第一个方法是像负载均衡一样用 DNS 服务器来分配访问。也就是说，我们可以在 DNS 服务器返回 Web 服务器 IP 地址时，对返回的内容进行一些加工，使其能够返回距离客户端最近的缓存服务器的 IP 地址。在解释这种方法的具体原理之前，我们先来复习一下 DNS 的基本工作方式。

互联网中有很多台 DNS 服务器，它们通过相互接力来处理 DNS 查询，这个过程从客户端发送查询消息开始，也就是说客户端会用要访问的 Web 服务器域名生成查询消息，并发送给自己局域网中的 DNS 服务器[①]（图

① 如果自己的本地局域网中没有 DNS 服务器，则将请求发送给运营商的 DNS 服务器。

5.12 ①)。然后，客户端 DNS 服务器会通过域名的层次结构找到负责管理该域名的 DNS 服务器，也就是 Web 服务器端的那个 DNS 服务器，并将查询消息发送给它(图 5.12 ②)。Web 服务器端的 DNS 服务器收到查询消息后，会查询并返回域名相对应的 IP 地址。在这台 DNS 中，有一张管理员维护的域名和 IP 地址的对应表，只要按照域名查表，就可以找到相应的 IP 地址(图 5.12 ③)。接下来，响应消息回到客户端的 DNS 服务器，然后再返回给客户端(图 5.12 ④)。

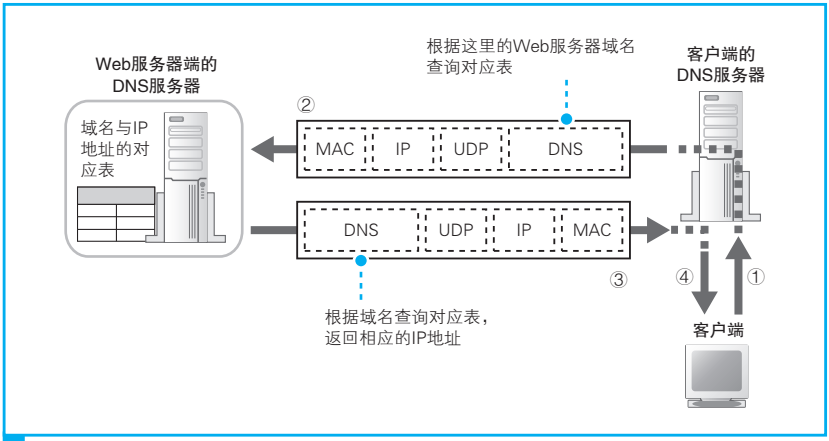


图 5.12　DNS 服务器的一般工作方式

上面讲的是 Web 服务器的域名只对应一个 IP 地址的情况，如果一个域名对应多个 IP 地址，则按照前面图 5.3 的轮询方式按顺序返回所有的 IP 地址。

如果按照 DNS 服务器的一般工作方式来看，它只能以轮询方式按顺序返回 IP 地址，完全不考虑客户端与缓存服务器的远近，因此可能会返回离客户端较远的缓存服务器 IP 地址。

如果要让用户访问最近的缓存服务器，则不应采用轮询方式，而是应该判断客户端与缓存服务器的距离，并返回距离客户端最近的缓存服务器 IP 地址。这里的关键点不言自明，那就是到底该怎样判断客户端与缓存服务器之间的距离呢?

方法是这样的。首先，作为准备，需要事先从缓存服务器部署地点的路由器收集路由信息（图 5.13）。例如，在图 5.13 的例子中，一共有 4 台缓存服务器，在这 4 台服务器的部署地点又分别有 4 台路由器，则我们需要分别获取这 4 台路由器的路由表，并将 4 张路由表集中到 DNS 服务器上。

接下来，DNS 服务器根据路由表查询从本机到 DNS 查询消息的发送方，也就是客户端 DNS 服务器的路由信息。例如，根据图 5.13 路由器 A 的路由表，可以查出路由器 A 到客户端 DNS 服务器的路由。通过互联网内部的路由表中的路由信息可以知道先通过运营商 X，然后通过运营商 Y，最后到达运营商 Z 这样的信息，通过这样的信息可以大致估算出距离。依次查询所有路由器的路由表之后，我们就可以通过比较找出哪一台路由器距离客户端 DNS 服务器最近。提供路由表的路由器位于缓存服务器的位置，而客户端 DNS 服务器也应该和客户端在同一位置，这样就等于估算出了缓存服务器与客户端之间的距离，从而能够判断出哪台缓存服务器距离客户端最近了。实际上，客户端 DNS 服务器不一定和客户端在同一位置，因此可能无法得出准确的距离，但依然可以达到相当的精度。

5.5.3 通过重定向服务器分配访问目标

还有另一个让客户端访问最近的缓存服务器的方法。HTTP 规格中定义了很多头部字段，其中有一个叫作 Location 的字段。当 Web 服务器数据转移到其他服务器时可以使用这个字段，它的意思是“您要访问的数据在另一台服务器上，请访问那台服务器吧。”这种将客户端访问引导到另一台 Web 服务器的操作称为重定向，通过这种方法也可以将访问目标分配到最近的缓存服务器。

当使用重定向告知客户端最近的缓存服务器时，首先需要将重定向服务器注册到 Web 服务器端的 DNS 服务器上。这样一来，客户端会将 HTTP 请求消息发送到重定向服务器上。重定向服务器和刚才一种方法中的 DNS 服务器一样，收集了来自各个路由器的路由信息，并根据这些信息找到最近的缓存服务器，然后将缓存服务器的地址放到 Location 字段中返回响应。

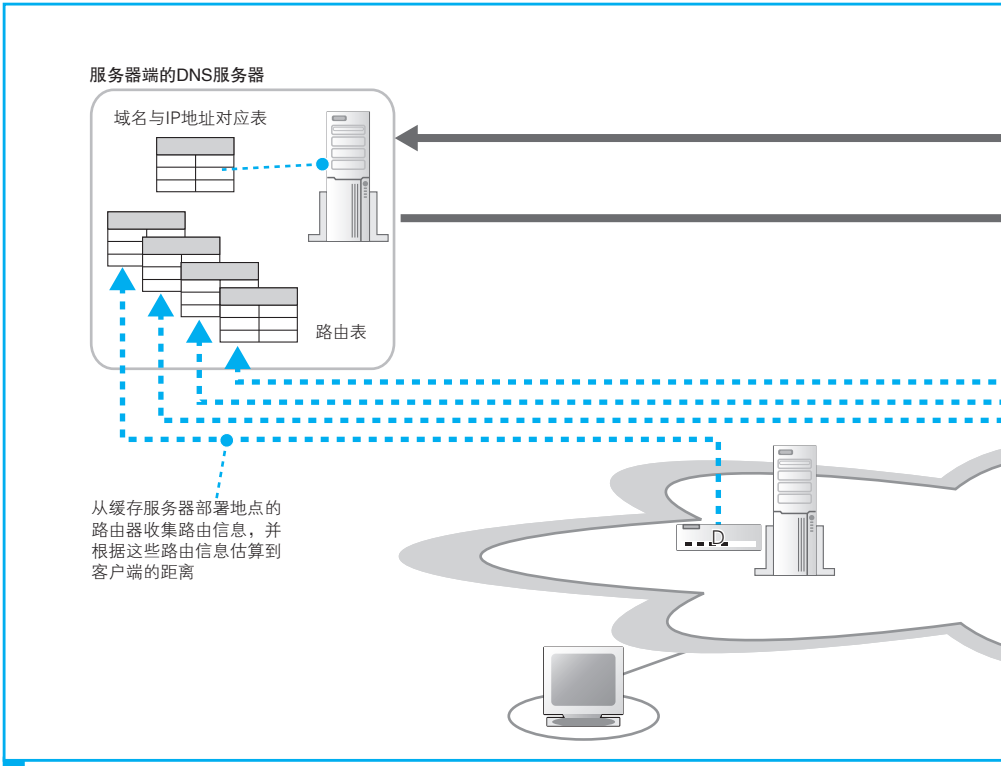
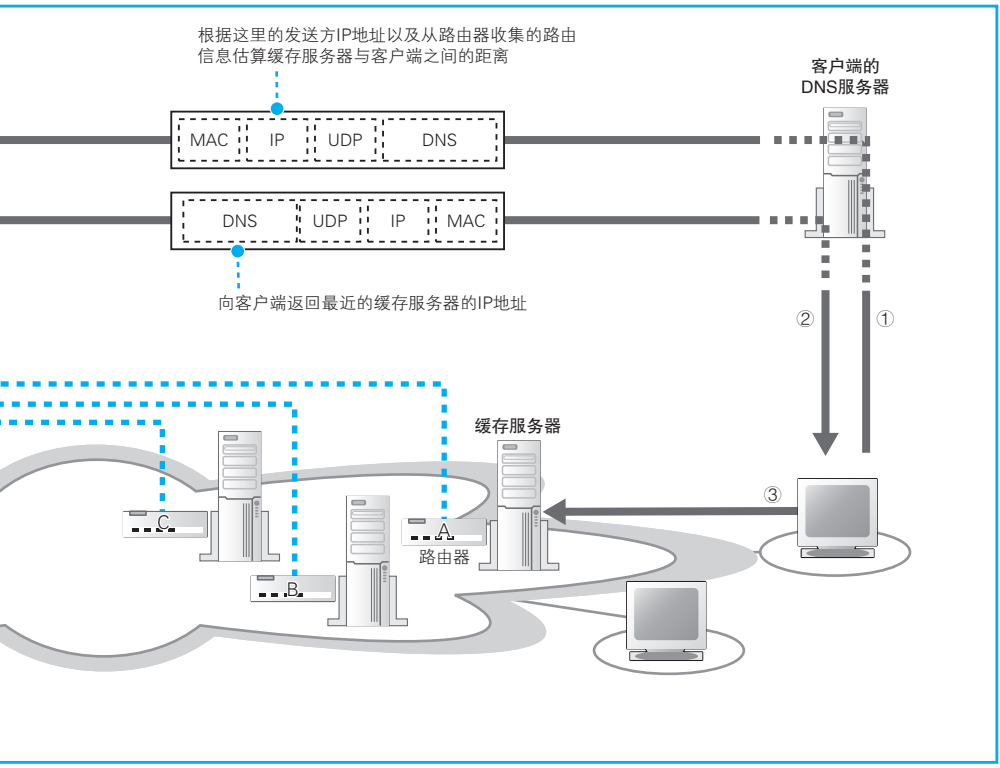


图 5.13　DNS 服务器参照路由信息时的工作方式

这样，客户端就会重新去访问指定的缓存服务器了（图 5.14、图 5.15）。

这种方法的缺点在于增加了 HTTP 消息的交互次数，相应的开销也比较大，但它也有优点。对 DNS 服务器进行扩展的方法是估算客户端 DNS 服务器到缓存服务器之间的距离，因此精度较差；相对而言，重定向的方法是根据客户端发送来的 HTTP 消息的发送方 IP 地址来估算距离的，因此精度较高。

此外，也可以使用除路由信息之外的其他一些信息来估算距离，进一步提高精度。重定向服务器不仅可以返回带有 Location 字段的 HTTP 消息，也可以返回一个通过网络包往返时间估算到缓存服务器的距离的脚本，通过在客户端运行脚本来找到最优的缓存服务器。这个脚本可以向不同的



缓存服务器发送测试包并计算往返时间，然后将请求发送到往返时间最短的一台缓存服务器，这样就可以判断出对于客户端最优的缓存服务器，并让客户端去访问该服务器。

5.5.4 缓存的更新方法会影响性能

还有一个因素会影响缓存服务器的效率，那就是缓存内容的更新方法。缓存本来的思路是像图 5.5 那样，将曾经访问过的数据保存下来，然后当再次访问时拿出来用，以提高访问操作的效率。不过，这种方法对于第一次访问是无效的，而且后面的每次访问都需要向原始服务器查询数据有没

有发生变化，如果遇到网络拥塞，就会使响应时间恶化。

要改善这一点，有一种方法是让 Web 服务器在原始数据发生更新时，立即通知缓存服务器，使得缓存服务器上的数据一直保持最新状态，这样就不需要每次确认原始数据是否有变化了，而且从第一次访问就可以发挥缓存的效果。内容分发服务采用的缓存服务器就具备这样的功能。

此外，除了事先编写好内容的静态页面之外，还有一些在收到请求后由 CGI 程序生成的动态页面，这种动态页面是不能保存在缓存服务器上

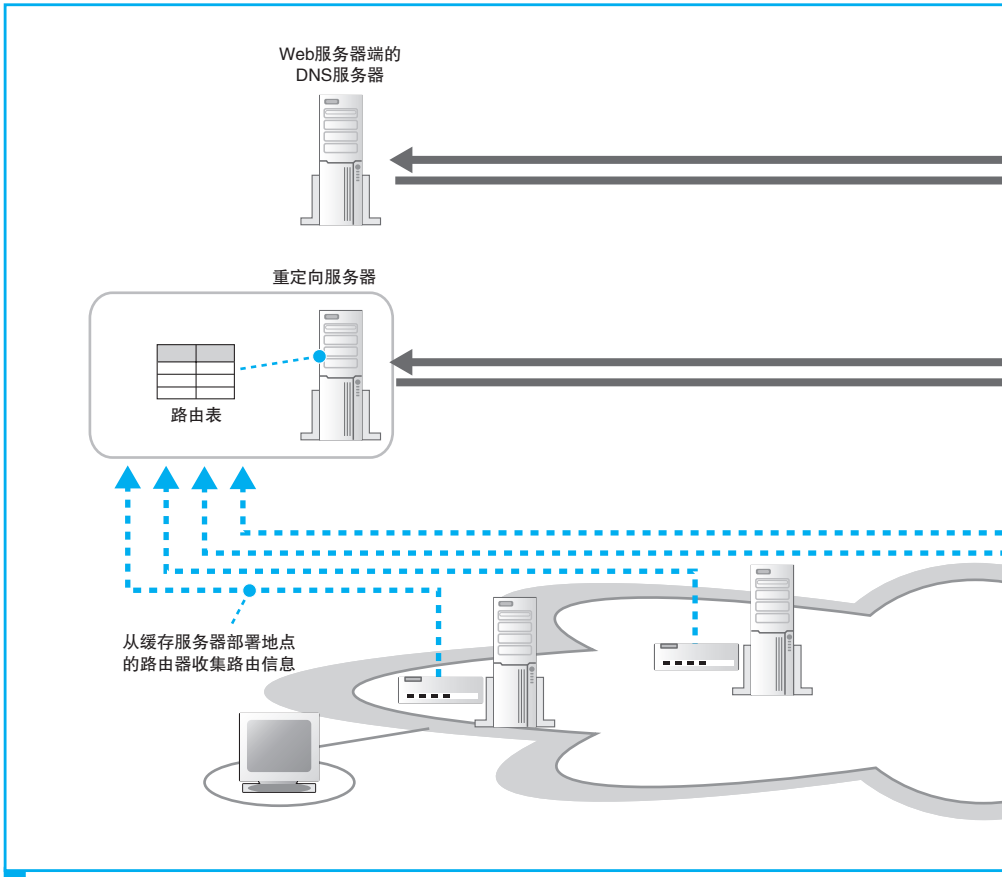


图 5.14　通过重定向让客户端访问最近的缓存服务器的机制