

部的协议栈<sup>①</sup>来执行。这是因为和浏览器一样，解析器本身也不具备使用网络收发数据的功能。解析器调用协议栈后，控制流程会再次转移，协议栈会执行发送消息的操作，然后通过网卡将消息发送给 DNS 服务器（图 1.12 ④⑤）。

当 DNS 服务器收到查询消息后，它会根据消息中的查询内容进行查询。这个查询的过程有点复杂，我们稍后会进行讲解，这里先不关心具体的方法。

总之，如果要访问的 Web 服务器已经在 DNS 服务器上注册，那么这条记录就能够被找到，然后其 IP 地址会被写入响应消息并返回给客户端（图 1.12 ⑥）。接下来，消息经过网络到达客户端，再经过协议栈被传递给解析器（图 1.12 ⑦⑧），然后解析器读取取出消息取出 IP 地址，并将 IP 地址传递给应用程序（图 1.12 ⑨）。实际上，解析器会将取出的 IP 地址写入应用程序指定的内存地址中，图 1.11 用“<内存地址>”来表示，在实际的程序代码中应该写的是代表这一内存地址的名称。

到这里，解析器的工作就完成了，控制流程重新回到应用程序（浏览器）。现在应用程序已经能够从内存中取出 IP 地址了，所以说 IP 地址是用这种方式传递给应用程序的。

计算机的内部结构就是这样一层一层的。也就是说，很多程序组成不同的层次，彼此之间分工协作。当接到上层委派的操作时，本层的程序并不会完成所有的工作，而是会完成一部分工作，再将剩下的部分委派到下层来完成。

顺带一提，向 DNS 服务器发送消息时，我们当然也需要知道 DNS 服务器的 IP 地址。只不过这个 IP 地址是作为 TCP/IP 的一个设置项目事先设置好的，不需要再去查询了。不同的操作系统中 TCP/IP 的设置方法也有差异，Windows 中的设置如图 1.13 所示，解析器会根据这里设置的 DNS 服务器 IP 地址来发送消息。

---

① 协议栈：操作系统内部的网络控制软件，也叫“协议驱动”“TCP/IP 驱动”等。

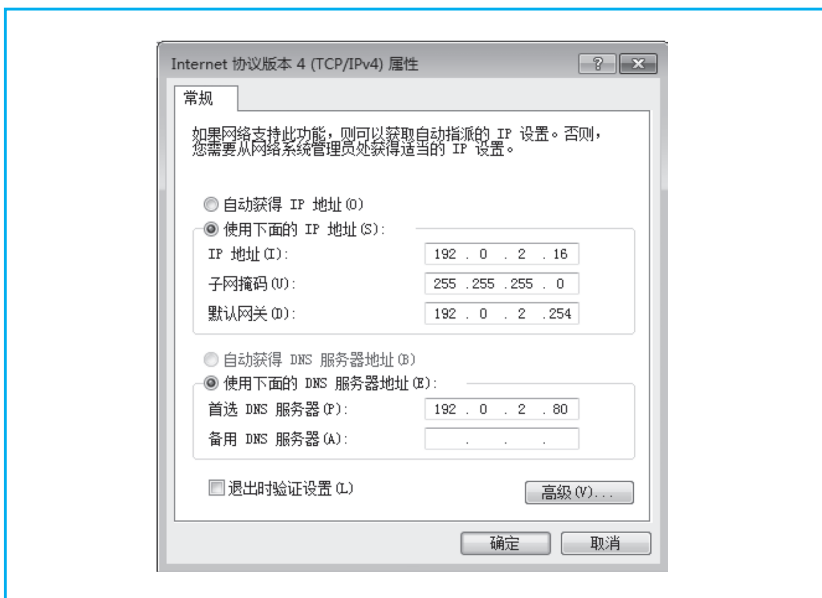


图 1.13 DNS 服务器地址的设置

## 1.3 全世界 DNS 服务器的大接力

### 1.3.1 DNS 服务器的基本工作

前文介绍了解析器与 DNS 服务器之间的交互过程，下面来了解一下 DNS 服务器的工作。DNS 服务器的基本工作就是接收来自客户端的查询消息，然后根据消息的内容返回响应。

其中，来自客户端的查询消息包含以下 3 种信息。

#### (a) 域名

服务器、邮件服务器（邮件地址中 @ 后面的部分）的名称

#### (b) Class

在最早设计 DNS 方案时，DNS 在互联网以外的其他网络中的应用

也被考虑到了，而 Class 就是用来识别网络的信息。不过，如今除了互联网并没有其他的网络了，因此 Class 的值永远是代表互联网的 IN

### (c) 记录类型

表示域名对应何种类型的记录。例如，当类型为 A 时，表示域名对应的是 IP 地址；当类型为 MX 时，表示域名对应的是邮件服务器。对于不同的记录类型，服务器向客户端返回的信息也会不同

DNS 服务器上事先保存有前面这 3 种信息对应的记录数据，如图 1.14 所示。DNS 服务器就是根据这些记录查找符合查询请求的内容并对客户端作出响应的。

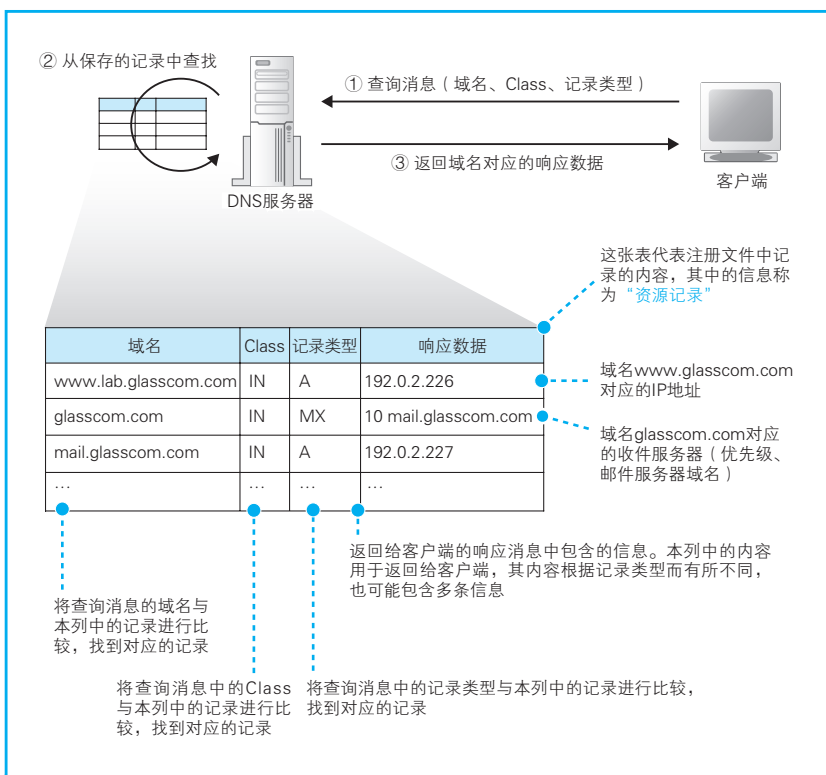


图 1.14 DNS 服务器的基本工作

例如，如果要查询 `www.lab.glasscom.com` 这个域名对应的 IP 地址，客户端会向 DNS 服务器发送包含以下信息的查询消息。

- (a) 域名 = `www.lab.glasscom.com`
- (b) Class = IN
- (c) 记录类型 = A

然后，DNS 服务器会从已有的记录中查找域名、Class 和记录类型全部匹配的记录。假如 DNS 服务器中的记录如图 1.14 所示，那么第一行记录与查询消息中的 3 个项目完全一致。于是，DNS 服务器会将记录中的 `192.0.2.226` 这个值返回给客户端。然而，Web 服务器的域名有很多都是像 `www.lab.glasscom.com` 这样以 `www` 开头的，但这并不是一定之规，只是因为最早设计 Web 的时候，很多 Web 服务器都采用了 `www` 这样的命名，后来就形成了一个惯例而已。因此，无论是 `WebServer1` 也好，`MySrv` 也好，只要是作为 A<sup>①</sup> 记录在 DNS 服务器上注册的，都可以作为 Web 服务器的域名<sup>②</sup>。

在查询 IP 地址时我们使用 A 这个记录类型，而查询邮件服务器时则使用 MX<sup>③</sup> 类型。这是因为在 DNS 服务器上，IP 地址是保存在 A 记录中的，而邮件服务器则是保存在 MX 记录中的。例如，对于一个邮件地址 `tone@glasscom.com`，当需要知道这个地址对应的邮件服务器时，我们需要提供 @ 后面的那一串名称。查询消息的内容如下。

- (a) 域名 = `glasscom.com`
- (b) Class = IN
- (c) 记录类型 = MX

---


① A 是 Address 的缩写。

② 不仅是 Web 服务器，像邮件服务器、数据库服务器等，无论任何服务器，只要注册了 A 类型的记录，都可以作为服务器的域名来使用。准确来说，A 类型的记录表示与 IP 地址所对应的域名，因此与其说是某个服务器的域名，不如说是被分配了某个 IP 地址的某台具体设备的域名。

③ MX: Mail eXchange，邮件交换。

DNS 服务器会返回 10 和 mail.glasscom.com 这两条信息。当记录类型为 MX 时，DNS 服务器会在记录中保存两种信息，分别是邮件服务器的域名和优先级<sup>①</sup>。此外，MX 记录的返回消息还包括邮件服务器 mail.glasscom.com 的 IP 地址。上表的第三行就是 mail.glasscom.com 的 IP 地址，因此只要用 mail.glasscom.com 的域名就可以找到这条记录。在这个例子中，我们得到的 IP 地址是 192.0.2.227。

综上所述，DNS 服务器的基本工作就是根据需要进行查询的域名和记录类型查找相关的记录，并向客户端返回响应消息。



DNS 服务器会从域名与 IP 地址的对照表中查找相应的记录，并返回 IP 地址。

前面只介绍了 A 和 MX 这两个记录类型，实际上还有很多其他的类型。例如根据 IP 地址反查域名的 PTR 类型，查询域名相关别名的 CNAME 类型，查询 DNS 服务器 IP 地址的 NS 类型，以及查询域名属性信息的 SOA 类型等。尽管 DNS 服务器的工作原理很简单，不过是根据查询消息中的域名和记录类型来进行查找并返回响应的信息而已，但通过组合使用不同的记录类型，就可以处理各种各样的信息。

此外，虽然图 1.14 展示的是表格形式，但实际上这些信息是保存在配置文件中的，表格中的一行信息被称为一条资源记录。

### 1.3.2 域名的层次结构

在前面的讲解中，我们假设要查询的信息已经保存在 DNS 服务器内部的记录中了。如果是在像公司内部网络这样 Web 和邮件服务器数量有限的环境中，所有的信息都可以保存在一台 DNS 服务器中，其工作方式也就完全符合我们前面讲解的内容。然而，互联网中存在着不计其数的服务器，

<sup>①</sup> 当一个邮件地址对应多个邮件服务器时，需要根据优先级来判断哪个邮件服务器是优先的。优先级数值较小的邮件服务器代表更优先。

将这些服务器的信息全部保存在一台 DNS 服务器中是不可能的，因此一定会出现在 DNS 服务器中找不到要查询的信息的情况。下面来看一看此时 DNS 服务器是如何工作的。

直接说答案的话很简单，就是将信息分布保存在多台 DNS 服务器中，这些 DNS 服务器相互接力配合，从而查找出要查询的信息。不过，这个机制其实有点复杂，因此我们先来看一看信息是如何在 DNS 服务器上注册并保存的。

首先，DNS 服务器中的所有信息都是按照域名以分层次的结构来保存的。层次结构这个词听起来可能有点不容易懂，其实就类似于公司中的事业集团、部门、科室这样的结构。层次结构能够帮助我们更好地管理大量的信息。

DNS 中的域名都是用句点来分隔的，比如 `www.lab.glasscom.com`，这里的句点代表了不同层次之间的界限，就相当于公司里面的组织结构不用部、科之类的名称来划分，只是用句点来分隔而已<sup>①</sup>。在域名中，越靠右的位置表示其层级越高，比如 `www.lab.glasscom.com` 这个域名如果按照公司里的组织结构来说，大概就是“com 事业集团 glasscom 部 lab 科的 www”这样。其中，相当于一个层级的部分称为域。因此，com 域的下一层是 glasscom 域，再下一层是 lab 域，再下面才是 www 这个名字。

这种具有层次结构的域名信息会注册到 DNS 服务器中，而每个域都是作为一个整体来处理的。换句话说就是，一个域的信息是作为一个整体存放在 DNS 服务器中的，不能将一个域拆开来存放在多台 DNS 服务器中。不过，DNS 服务器和域之间的关系也并不总是一对一的，一台 DNS 服务器中也可以存放多个域的信息。为了避免把事情搞得太复杂，这里先假设一台 DNS 服务器中只存放一个域的信息，后面的讲解也是基于这个前提来进行的。于是，DNS 服务器也具有了像域名一样的层次结构，每个域的信息都存放在相应层级的 DNS 服务器中。例如，这里有一个公司的域，那么

<sup>①</sup> 公司里面的部、科之类的名称会让层次变得固化，缺乏灵活性，而用句点来分隔则可以很容易地增加新的层次，从而提高了灵活性。

就相应地有一台 DNS 服务器，其中存放了公司中所有 Web 服务器和邮件服务器的信息<sup>①</sup>。

这里再补充一点。对于公司域来说，例如现在需要为每一个事业集团配备一台 DNS 服务器，分别管理各事业集团自己的信息，但我们之前也说过一个域是不可分割的，这该怎么办呢？没关系，我们可以在域的下面创建下级域<sup>②</sup>，然后再将它们分别分配给各个事业集团。比如，假设公司的域为 `example.co.jp`，我们可以在这个域的下面创建两个子域，即 `sub1.example.co.jp` 和 `sub2.example.co.jp`，然后就可以将这两个下级域分配给不同的事业集团来使用。如果公司下级的组织不是事业部而是子公司，对于域来说也是没有区别的。因为域并不代表“事业集团”这一特定组织，无论是子公司还是什么别的组织名称，都可以分配相应的域。实际上，互联网中的域也是一样，通过创建下级的域来分配给不同的国家、公司和组织使用。通过实际的域名可能更容易理解，比如 `www.nikkeibp.co.jp` 这个域名，最上层的 `jp` 代表分配给日本这个国家的域；下一层的 `co` 是日本国内进行分类的域，代表公司；再下层的 `nikkeibp` 就是分配给某个公司的域；最下层的 `www` 就是服务器的名称。

### 1.3.3 寻找相应的 DNS 服务器并获取 IP 地址

下面再来看一看如何找到 DNS 服务器中存放的信息。这里的关键在于如何找到我们要访问的 Web 服务器的信息归哪一台 DNS 服务器管。

互联网中有数万台 DNS 服务器，肯定不能一台一台挨个去找。我们可以采用下面的办法。首先，将负责管理下级域的 DNS 服务器的 IP 地址注册到它们的上级 DNS 服务器中，然后上级 DNS 服务器的 IP 地址再注册到更上一级的 DNS 服务器中，以此类推。也就是说，负责管理 `lab.glasscom`。

- 
- ① 实际上，由于一台 DNS 服务器可以存放多个域的信息，因此并不是每个域名都有一台与之相对应的 DNS 服务器。比如网络运营商的 DNS 服务器中就存放了很多个域的信息。
  - ② 下级的域称为“子域”。



com 这个域的 DNS 服务器的 IP 地址需要注册到 glasscom.com 域的 DNS 服务器中，而 glasscom.com 域的 DNS 服务器的 IP 地址又需要注册到 com 域的 DNS 服务器中。这样，我们就可以通过上级 DNS 服务器查询出下级 DNS 服务器的 IP 地址，也就可以向下级 DNS 服务器发送查询请求了。

在前面的讲解中，似乎 com、jp 这些域（称为顶级域）就是最顶层了，它们各自负责保存下级 DNS 服务器的信息，但实际上并非如此。在互联网中，com 和 jp 的上面还有一级域，称为根域。根域不像 com、jp 那样有自己的名字，因此在一般书写域名时经常被省略，如果要明确表示根域，应该像 www.lab.glasscom.com. 这样在域名的最后再加上一个句点，而这个最后的句点就代表根域。不过，一般都不写最后那个句点，因此根域的存在往往被忽略，但根域毕竟是真实存在的，根域的 DNS 服务器中保管着 com、jp 等的 DNS 服务器的信息。由于上级 DNS 服务器保管着所有下级 DNS 服务器的信息，所以我们可以从根域开始一路往下顺藤摸瓜找到任意一个域的 DNS 服务器。

除此之外还需要完成另一项工作，那就是将根域的 DNS 服务器信息保存在互联网中所有的 DNS 服务器中。这样一来，任何 DNS 服务器就都可以找到并访问根域 DNS 服务器了。因此，客户端只要能够找到任意一台 DNS 服务器，就可以通过它找到根域 DNS 服务器，然后再一路顺藤摸瓜找到位于下层的某台目标 DNS 服务器（图 1.15）。分配给根域 DNS 服务器的 IP 地址在全世界仅有 13 个<sup>①</sup>，而且这些地址几乎不发生变化，因此将这些地址保存在所有的 DNS 服务器中也并不是一件难事。实际上，根域 DNS 服务器的相关信息已经包含在 DNS 服务器程序的配置文件中了，因此只要安装了 DNS 服务器程序，这些信息也就被自动配置好了。

到这里所有的准备工作就都完成了。当我们配置一台 DNS 服务器时，必须要配置好上面这些信息，这样 DNS 服务器就能够从上万台 DNS 服务器中找到目标服务器。下面就来看一看这个过程是如何进行的。

<sup>①</sup> 根域 DNS 服务器在运营上使用多台服务器来对应一个 IP 地址，因此尽管 IP 地址只有 13 个，但其实服务器的数量是很多的。



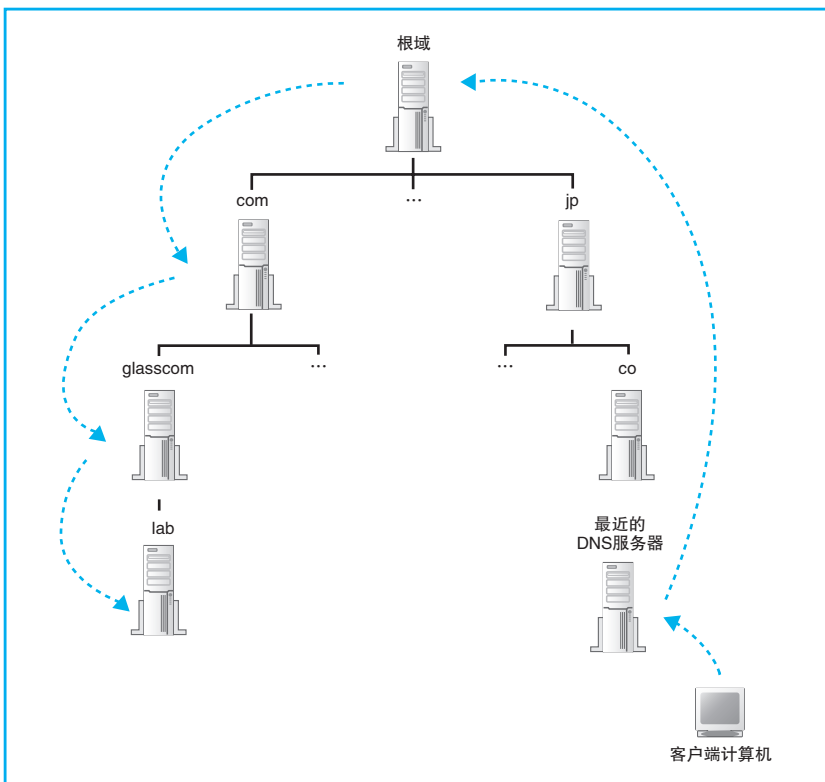


图 1.15 找到目标 DNS 服务器

如图 1.16 所示，客户端首先会访问最近的一台 DNS 服务器（也就是客户端的 TCP/IP 设置中填写的 DNS 服务器地址），假设我们要查询 `www.lab.glasscom.com` 这台 Web 服务器的相关信息（图 1.16 ①）。由于最近的 DNS 服务器中没有存放 `www.lab.glasscom.com` 这一域名对应的信息，所以我们需要从顶层开始向下查找。最近的 DNS 服务器中保存了根域 DNS 服务器的信息，因此它会将来自客户端的查询消息转发给根域 DNS 服务器（图 1.16 ②）。根域服务器中也没有 `www.lab.glasscom.com` 这个域名，但根据域名结构可以判断这个域名属于 `com` 域，因此根域 DNS 服务器会返回它所管理的 `com` 域中的 DNS 服务器的 IP 地址，意思是“虽然我不知道你要查的那个域名的地址，

但你可以去 com 域问问看”。接下来，最近的 DNS 服务器又会向 com 域的 DNS 服务器发送查询消息(图 1.16 ③)。com 域中也没有 www.lab.glasscom.com 这个域名的信息，和刚才一样，com 域服务器会返回它下面的 glasscom.com 域的 DNS 服务器的 IP 地址。以此类推，只要重复前面的步骤，就可以顺藤摸瓜找到目标 DNS 服务器(图 1.16 ⑤)，只要向目标 DNS 服务器发送查询消息，就能够得到我们需要的答案，也就是 www.lab.glasscom.com 的 IP 地址了。

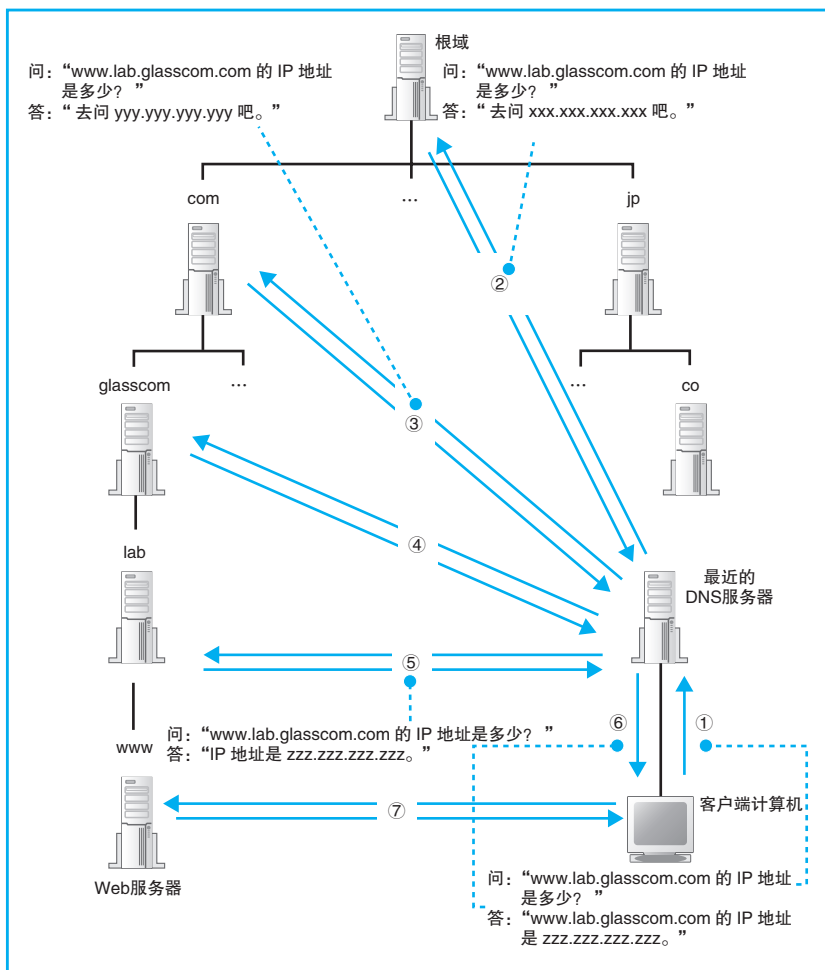


图 1.16 DNS 服务器之间的查询操作

收到客户端的查询消息之后，DNS 服务器会按照前面的方法来查询 IP 地址，并返回给客户端（图 1.16 ⑥）。这样，客户端就知道了 Web 服务器的 IP 地址，也就能够对其进行访问了（图 1.16 ⑦）。

搞清楚了 DNS 服务器的工作方式之后，我们将图 1.12 和图 1.16 连起来看看。图 1.16 中的①和⑥分别相当于图 1.12 中的⑤和⑥，将这部分重合起来，就可以将这两张图连起来了。不过，在图 1.12 和图 1.16 中，客户端和 DNS 服务器的上下位置关系是颠倒着的，因此需要将其中一张图倒过来看。这样，我们就可以看清楚浏览器调用 `gethostbyname` 查询 Web 服务器地址的全貌，这也就是向 DNS 服务器查询 IP 地址的实际过程。

### 1.3.4 通过缓存加快 DNS 服务器的响应

图 1.16 展示的是基本原理，与真实互联网中的工作方式还是有一些区别的。在真实的互联网中，一台 DNS 服务器可以管理多个域的信息，因此并不是像图 1.16 这样每个域都有一台自己的 DNS 服务器。图中，每一个域旁边都写着一台 DNS 服务器，但现实中上级域和下级域有可能共享同一台 DNS 服务器。在这种情况下，访问上级 DNS 服务器时就可以向下跳过一级 DNS 服务器，直接返回再下一级 DNS 服务器的相关信息。

此外，有时候并不需要从最上级的根域开始查找，因为 DNS 服务器有一个缓存<sup>①</sup>功能，可以记住之前查询过的域名。如果要查询的域名和相关信息已经在缓存中，那么就可以直接返回响应，接下来的查询可以从缓存的位置开始向下进行。相比每次都从根域找起来说，缓存可以减少查询所需的时间。

并且，当要查询的域名不存在时，“不存在”这一响应结果也会被缓存。这样，当下次查询这个不存在的域名时，也可以快速响应。

这个缓存机制中有一点需要注意，那就是信息被缓存后，原本的注册信息可能会发生改变，这时缓存中的信息就有可能是不正确的。因此，DNS 服

① 缓存：指的是将使用过的数据存放在离使用该数据的地方较近的高速存储装置中，以便提高后续访问速度的技术。这一技术有很多应用，如 CPU 和内存之间的缓存、磁盘和内存之间的缓存等，在网络中缓存也是一种用来提高访问速度的普遍性技术。

务器中保存的信息都设置有一个有效期，当缓存中的信息超过有效期后，数据就会从缓存中删除。而且，在对查询进行响应时，DNS 服务器也会告知客户端这一响应的结果是来自缓存中还是来自负责管理该域名的 DNS 服务器。

## 1.4 委托协议栈发送消息

### 1.4.1 数据收发操作概览

知道了 IP 地址之后，就可以委托操作系统内部的协议栈向这个目标 IP 地址，也就是我们要访问的 Web 服务器发送消息了。要发送给 Web 服务器的 HTTP 消息是一种数字信息 (digital data)，因此也可以说是委托协议栈来发送数字信息。收发数字信息这一操作不仅限于浏览器，对于各种使用网络的应用程序来说都是共通的。因此，这一操作的过程也不仅适用于 Web，而是适用于任何网络应用程序<sup>①</sup>。下面就来一起探索这一操作的过程。

和向 DNS 服务器查询 IP 地址的操作一样，这里也需要使用 Socket 库中的程序组件。不过，查询 IP 地址只需要调用一个程序组件就可以了，而这里需要按照指定的顺序调用多个程序组件，这个过程有点复杂。发送数据是一系列操作相结合来实现的，如果不能理解这个操作的全貌，就无法理解其中每个操作的意义。因此，我们先来介绍一下收发数据操作的整体思路。

向操作系统内部的协议栈发出委托时，需要按照指定的顺序来调用 Socket 库中的程序组件。

使用 Socket 库来收发数据的操作过程如图 1.17 所示<sup>②</sup>。简单来说，收发数据的两台计算机之间连接了一条数据通道，数据沿着这条通道流动，最

① 通过 DNS 服务器查询 IP 地址的操作也同样适用于所有网络应用程序。

② 图 1.17 中展示的是用 TCP 协议来收发数据的过程，还有另外一种名为 UDP (User Datagram Protocol, 用户数据报协议) 的协议，其收发数据的过程将在后面进行讲解。

终到达目的地。我们可以把数据通道想象成一条管道，将数据从一端送入管道，数据就会到达管道的另一端然后被取出。数据可以从任何一端被送入管道，数据的流动是双向的。不过，这并不是说现实中真的有这么一条管道，只是为了帮助大家理解数据收发操作的全貌。

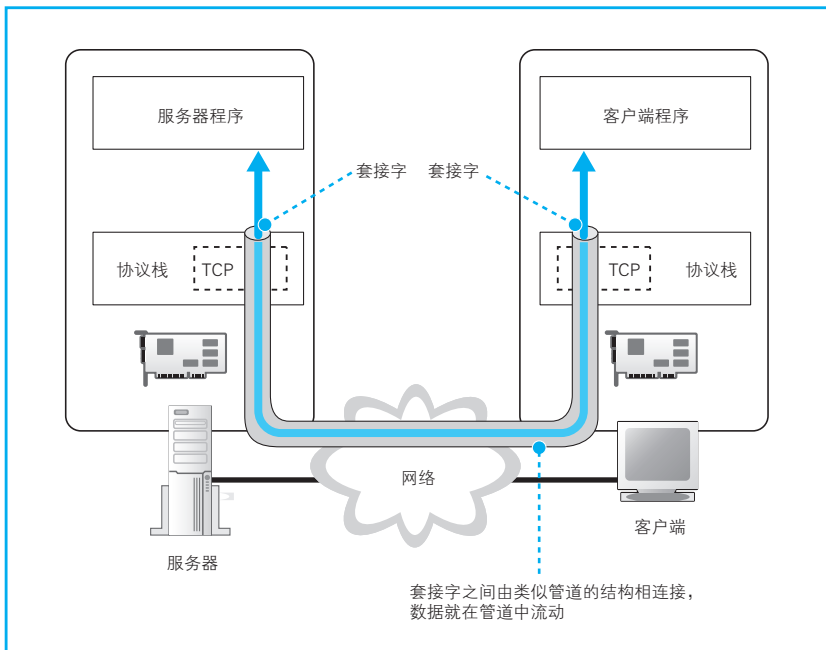


图 1.17 数据通过类似管道的结构来流动

收发数据的整体思路就是这样，但还有一点也非常重要。光从图上来看，这条管道好像一开始就有，实际上并不是这样，在进行收发数据操作之前，双方需要先建立起这条管道才行。建立管道的关键在于管道两端的数据出入口，这些出入口称为套接字。我们需要先创建套接字，然后再将套接字连接起来形成管道。实际的过程是下面这样的。首先，服务器一方先创建套接字，然后等待客户端向该套接字连接管道<sup>①</sup>。当服务器进入等待

<sup>①</sup> 服务器程序一般会在启动后就创建好套接字并等待客户端连接管道。

状态时，客户端就可以连接管道了。具体来说，客户端也会先创建一个套接字，然后从该套接字延伸出管道，最后管道连接到服务器端的套接字上。当双方的套接字连接起来之后，通信准备就完成了。接下来，就像我们刚刚讲过的一样，只要将数据送入套接字就可以收发数据了。

我们再来看看收发数据操作结束时的情形。当数据全部发送完毕之后，连接的管道将会被断开。管道在连接时是由客户端发起的，但在断开时可以由客户端或服务器任意一方发起<sup>①</sup>。其中一方断开后，另一方也会随之断开，当管道断开后，套接字也会被删除。到此为止，通信操作就结束了。

综上所述，收发数据的操作分为若干个阶段，可以大致总结为以下 4 个。

- (1) 创建套接字（创建套接字阶段）
- (2) 将管道连接到服务器端的套接字上（连接阶段）
- (3) 收发数据（通信阶段）
- (4) 断开管道并删除套接字（断开阶段）

在每个阶段，Socket 库中的程序组件都会被调用来执行相关的数据收发操作。不过，在探索其具体过程之前，我们来补充一点内容。前面这 4 个操作都是由操作系统中的协议栈来执行的，浏览器等应用程序并不会自己去做连接管道、放入数据这些工作，而是委托协议栈来代劳。本章将要介绍的只是这个“委托”的操作。关于协议栈收到委托之后具体是如何连接管道和放入数据的，我们将在第 2 章介绍。此外，这些委托的操作都是通过调用 Socket 库中的程序组件来执行的，但这些数据通信用的程序组件其实仅仅充当了一个桥梁的角色，并不执行任何实质性的操作，应用程序的委托内容最终会被原原本本地传递给协议栈。因此，我们无法形象地展示这些程序组件到底完成了怎样的工作，与其勉强强调 Socket 库的存在，还不如将 Socket 库和协议栈看成一个整体并讲解它们的整体行为让人

---

<sup>①</sup> 实际上，管道切断的顺序是根据应用程序的规则来决定的。在 Web 中，断开顺序根据 HTTP 版本的不同而不同，在 HTTP1.0 中，当服务器向客户端发送完所有 Web 数据之后，服务器一方会断开管道。1.4.5 一节会有详细介绍。

更容易理解。因此，后文将会采用这样的讲法。不过，请大家不要忘记 Socket 库这一桥梁的存在，正如图 1.12 中所示的一样。

### 1.4.2 创建套接字阶段

下面我们就来探索一下应用程序（浏览器）委托收发数据的过程。这个过程的关键点就是像对 DNS 服务器发送查询一样，调用 Socket 库中的特定程序组件。访问 DNS 服务器时我们调用的是一个叫作 `gethostbyname` 的程序组件（也就是解析器），而这一次则需要按照一定的顺序调用若干个程序组件，其过程如图 1.18 所示，请大家边看图边继续看下面的讲解。其中，调用 Socket 库中的程序组件的思路和图 1.11 旁边关于调用解析器的说明是一样的，请大家回忆一下。

首先是套接字创建阶段。客户端创建套接字的操作非常简单，只要调用 Socket 库中的 `socket` 程序组件<sup>①</sup>就可以了（图 1.18 ①）。和调用解析器一样，调用 `socket` 之后，控制流程会转移到 `socket` 内部并执行创建套接字的操作，完成之后控制流程又会被移交回应用程序。只不过，`socket` 的内部操作并不像解析器那样简单，因此我们将在第 2 章为大家详细讲解这部分内容<sup>②</sup>。现在大家只要知道调用 `socket` 后套接字就创建好了就可以了。

套接字创建完成后，协议栈会返回一个描述符，应用程序会将收到的描述符存放在内存中。描述符是用来识别不同的套接字的，大家可以作如下理解。我们现在只关注了浏览器访问 Web 服务器的过程，但实际上计算机中会同时进行多个数据的通信操作，比如可以打开两个浏览器窗口，同时访问两台 Web 服务器。这时，有两个数据收发操作在同时进行，也就需要创建两个不同的套接字。这个例子说明，同一台计算机上可能同时存在

---

① 书中出现了 Socket、socket、套接字（英文也是 socket）等看起来非常容易混淆的词，其中小写的 `socket` 表示程序组件的名称，大写字母开头的 `Socket` 表示库，而汉字的“套接字”则表示管道两端的接口。

② 后面将提到的 `connect`、`write`、`read`、`close` 等程序组件的内部操作也将在第 2 章讲解。