

图 6.4 分配接收到的包

- 服务器 IP 地址
- 服务器端口号

服务器上可能存在多个端口号相同的套接字，但客户端的套接字都是对应不同端口号的，因此我们可以通过客户端的端口号来确定服务器上的某个套接字。不过，使用不同端口号的规则仅限一台客户端的内部，当有多个客户端进行连接时，它们之间的端口号是可以重复的。因此，我们还必须加上客户端的 IP 地址才能进行判断。例如，IP 地址为 198.18.203.154 的客户端的 1025 端口，就和 IP 地址为 198.18.142.86 的客户端的 1025 端口对应不同的套接字。如果能够理解上面这些内容，那么关于套接字和端

口号的知识就已经掌握得差不多了。

说句题外话，既然通过客户端 IP 地址、客户端端口号、服务器 IP 地址、服务器端口号这 4 种信息可以确定某个套接字，那么要指代某个套接字时用这 4 种信息就好了，为什么还要使用描述符呢？这个问题很好，不过我们无法用上面 4 种信息来代替描述符。原因是，在套接字刚刚创建好，还没有建立连接的状态下，这 4 种信息是不全的。此外，为了指代一个套接字，使用一种信息（描述符）比使用 4 种信息要简单。出于上面两个原因，应用程序和协议栈之间是使用描述符来指代套接字的。



使用描述符来指代套接字的原因如下。

- (1) 等待连接的套接字中没有客户端 IP 地址和端口号
- (2) 使用描述符这一种信息比较简单

## 6.2

## 服务器的接收操作

### 6.2.1

### 网卡将接收到的信号转换成数字信息

了解了服务器的整体结构之后，下面我们重新回到探索之旅。现在，客户端发送的网络包已经到达了服务器。

到达服务器的网络包其本质是电信号或者光信号，接收信号的过程和客户端是一样的。关于这个过程我们在第 2 章介绍客户端包收发操作时已经讲过了<sup>①</sup>，不过这里还是简单复习一下，顺便从整体上看一看接收操作的全过程。

接收操作的第一步是网卡接收到信号，然后将其还原成数字信息<sup>②</sup>。局域网中传输的网络包信号是由 1 和 0 组成的数字信息与用来同步的时钟信

<sup>①</sup> 2.5.8 节有相关介绍。

<sup>②</sup> 关于网卡的结构，请参见 2.5.7 节。

号叠加而成的，因此只要从中分离出时钟信号，然后根据时钟信号进行同步，就可以读取并还原出 1 和 0 的数字信息了。

信号的格式随传输速率的不同而不同，因此某些操作过程可能存在细微差异，例如 10BASE-T 的工作方式如图 6.5 所示。首先从报头部分提取出时钟信号(图 6.5 ①)，报头的信号是按一定频率变化的，只要测定这个变化的频率就可以和时钟信号同步了。接下来，按照相同的周期延长时钟信号(图 6.5 ②)，并在每个时钟周期位置检测信号的变化方向(图 6.5 ③)。图中用向上和向下的箭头表示变化方向，实际的信号则是正或负电压，这里需要检测电压是从正变为负，还是从负变为正，这两种变化方向分别对应 0 和 1 (图 6.5 ④)。在图中，向上的箭头为 1，向下的箭头为 0，实际上是从负到正变化为 1，从正到负变化为 0。这样，信号就被还原成数字信息了(图 6.6)。

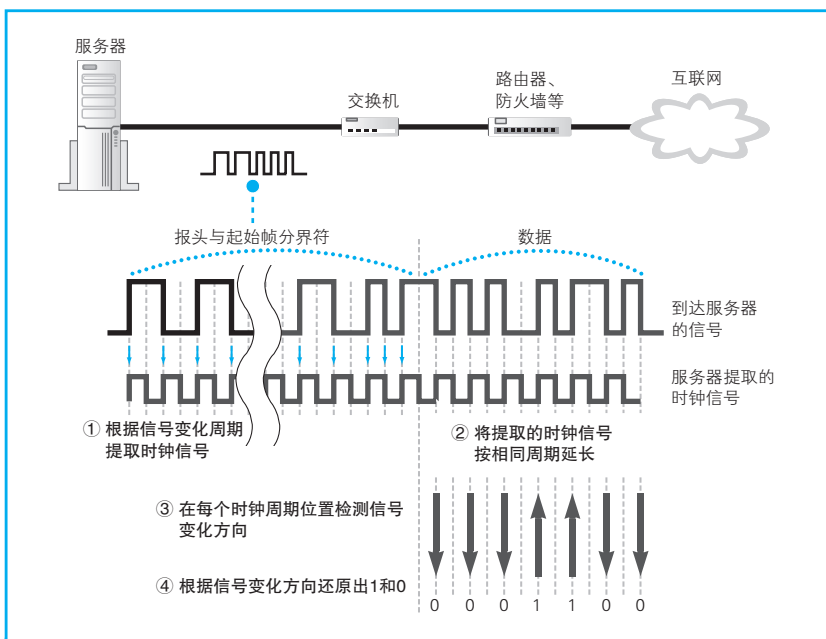


图 6.5 服务器将接收到的电信号还原为数字信息

服务器接收电信号的过程和客户端发送的过程相反，是从模拟信息转换为数字信息。

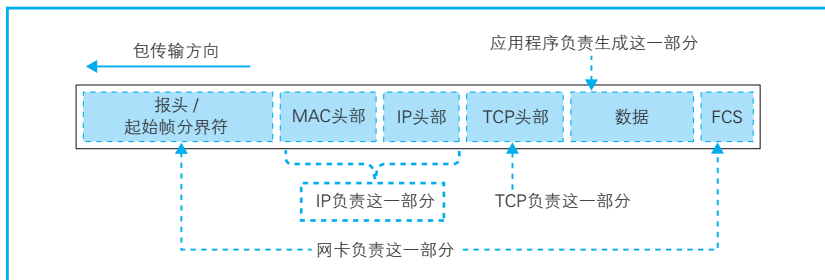


图 6.6 根据信号还原的数字信息

接下来需要根据包末尾的帧校验序列 (FCS) 来校验错误, 即根据校验公式<sup>①</sup> 计算刚刚接收到的数字信息, 然后与包末尾的 FCS 值进行比较。FCS 值是在发送时根据转换成电信号之前的数字信息进行计算得到的, 因此如果根据信号还原出的数字信息与发送前的信息一致, 则计算出的 FCS 也应该与包末尾的 FCS 一致。如果两者不一致, 则可能是因为噪声等影响导致信号失真, 数据产生了错误, 这时接收的包是无效的, 因此需要丢弃<sup>②</sup>。


当 FCS 一致, 即确认数据没有错误时, 接下来需要检查 MAC 头部中的接收方 MAC 地址, 看看这个包是不是发给自己的。以太网的基本工作方式是将数据广播到整个网络上, 只有指定的接收者才接收数据, 因此网络中还有很多发给其他设备的数据在传输, 如果包的接收者不是自己, 那么就需要丢弃这个包。

到这里, 接收信号并还原成数字信息的操作就完成了, 还原后的数字信息被保存在网卡内部的缓冲区中。上面这些操作都是由网卡的 MAC 模块<sup>③</sup> 来完成的。

① 以太网中使用 CRC-32 方式来计算。

② 包的丢失会由 TCP 检测出来并重传, 因此错误的包可以直接丢弃。


③ 关于 MAC 模块请参见 2.5.7 节。



网卡的 MAC 模块将网络包从信号还原为数字信息，校验 FCS 并存入缓冲区。

在这个过程中，服务器的 CPU 并不是一直在监控网络包的到达，而是在执行其他的任务，因此 CPU 并不知道此时网络包已经到达了。但接下来的接收操作需要 CPU 来参与，因此网卡需要通过中断将网络包到达的事件通知给 CPU。

接下来，CPU 就会暂停当前的工作，并切换到网卡的任务。然后，网卡驱动会开始运行，从网卡缓冲区中将接收到的包读取出来，根据 MAC 头部的以太类型字段判断协议的种类，并调用负责处理该协议的软件。这里，以太类型的值应该是表示 IP 协议，因此会调用 TCP/IP 协议栈，并将包转交给它<sup>①</sup>。



网卡驱动会根据 MAC 头部判断协议类型，并将包交给相应的协议栈。



### 6.2.2 IP 模块的接收操作

当网络包转交到协议栈时，IP 模块会首先开始工作，检查 IP 头部。IP 模块首先会检查 IP 头部的格式是否符合规范，然后检查接收方 IP 地址，看包是不是发给自己的。当服务器启用类似路由器的包转发功能时<sup>②</sup>，对于不是发给自己的包，会像路由器一样根据路由表对包进行转发<sup>③</sup>。

- ① 实际的工作过程因操作系统的不同而不同，大多数情况下，网卡驱动并不会直接调用协议栈，而是先切换回操作系统，然后再由操作系统去调用协议栈，由协议栈继续执行接收操作。
- ② 服务器操作系统中内置了可实现路由器功能的软件，只要启用这一功能，服务器就可以像路由器一样工作。
- ③ 服务器也可以启用类似防火墙的包过滤功能，这时，在包转发的过程中还会对包进行检查，并丢弃不符合规则的包。

确认包是发给自己的之后,接下来需要检查包有没有被分片<sup>①</sup>。检查 IP 头部的内容就可以知道是否分片<sup>②</sup>,如果是分片的包,则将包暂时存放在内存中,等所有分片全部到达之后将分片组装起来还原成原始包;如果没有分片,则直接保留接收时的样子,不需要进行重组。到这里,我们就完成了包的接收。

接下来需要检查 IP 头部的协议号字段,并将包转交给相应的模块。例如,如果协议号为 06 (十六进制),则将包转交给 TCP 模块;如果是 11 (十六进制),则转交给 UDP 模块。这里我们假设这个包被交给 TCP 模块处理,然后继续往下看。



协议栈的 IP 模块会检查 IP 头部,(1)判断是不是发给自己的;(2)判断网络包是否经过分片;(3)将包转交给 TCP 模块或 UDP 模块。

### 6.2.3 TCP 模块如何处理连接包

前面的步骤对于任何包都是一样的,但后面的 TCP 模块的操作则根据包的内容有所区别。首先,我们来看一下发起连接的包是如何处理的。

当 TCP 头部中的控制位 SYN 为 1 时,表示这是一个发起连接的包(图 6.7 ①)。这时,TCP 模块会执行接受连接的操作,不过在此之前,需要先检查包的接收方端口号,并确认在该端口上有没有与接收方端口号相同且正处于等待连接状态的套接字。如果指定端口号没有等待连接的套接字,则向客户端返回错误通知的包<sup>③</sup>。

如果存在等待连接的套接字,则为这个套接字复制一个新的副本,并将发送方 IP 地址、端口号、序号初始值、窗口大小等必要的参数写入这个

① 关于分片请参见 3.3.7 节。

② 参见 2.5.3 节的表 2.2 的 IP 头部标志。

③ 向客户端返回一个表示接收方端口不存在等待连接的套接字的 ICMP 消息。

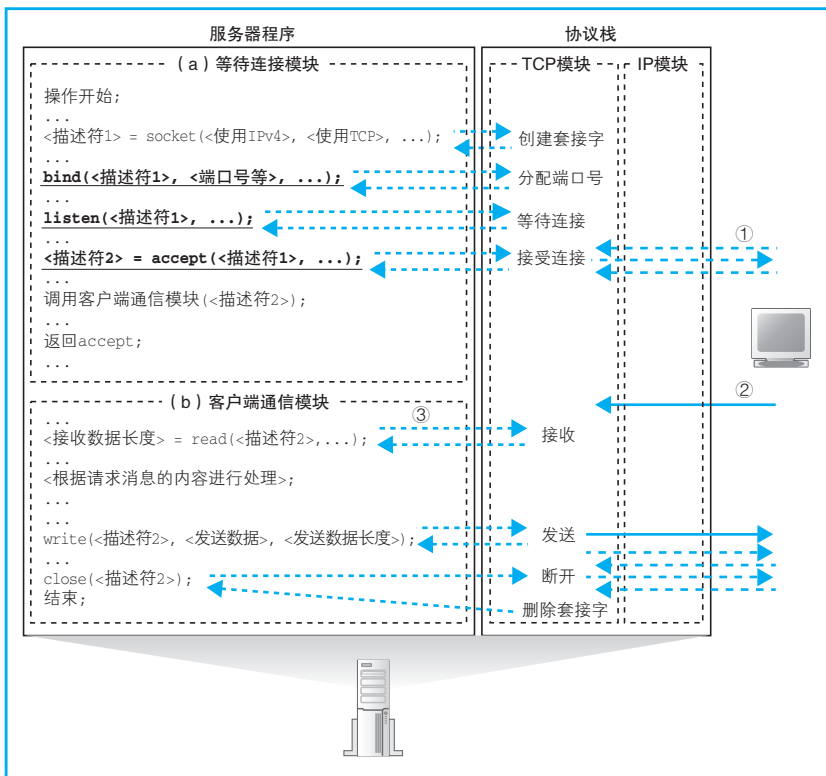


图 6.7 服务器的工作过程

套接字中，同时分配用于发送缓冲区和接收缓冲区的内存空间。然后生成代表接收确认的 ACK 号，用于从服务器向客户端发送数据的序号初始值，表示接收缓冲区剩余容量的窗口大小，并用这些信息生成 TCP 头部，委托 IP 模块发送给客户端<sup>①</sup>。

这个包到达客户端之后，客户端会返回表示接收确认的 ACK 号，当这个 ACK 号返回服务器后，连接操作就完成了。

这时，服务器端的程序应该进入调用 `accept` 的暂停状态，当将新套接字的描述符转交给服务器程序之后，服务器程序就会恢复运行。

<sup>①</sup> 这个包只有 TCP 头部，没有数据。



如果收到的是发起连接的包，则 TCP 模块会 (1) 确认 TCP 头部的控制位 SYN; (2) 检查接收方端口号; (3) 为相应的等待连接套接字复制一个新的副本; (4) 记录发送方 IP 地址和端口号等信息。



### 6.2.4 TCP 模块如何处理数据包

接下来我们来看看进入数据收发阶段之后，当数据包<sup>①</sup>到达时 TCP 模块是如何处理的 (图 6.7 ②)。

首先，TCP 模块会检查收到的包对应哪一个套接字。在服务器端，可能有多个已连接的套接字对应同一个端口号，因此仅根据接收方端口号无法找到特定的套接字。这时我们需要根据 IP 头部中的发送方 IP 地址和接收方 IP 地址，以及 TCP 头部中的接收方端口号和发送方端口号共 4 种信息，找到上述 4 种信息全部匹配的套接字<sup>②</sup>。

找到 4 种信息全部匹配的套接字之后，TCP 模块会对比该套接字中保存的数据收发状态和收到的包的 TCP 头部中的信息是否匹配，以确定数据收发操作是否正常。具体来说，就是根据套接字中保存的上一个序号和数据长度计算下一个序号，并检查与收到的包的 TCP 头部中的序号是否一致<sup>③</sup>。如果两者一致，就说明包正常到达了服务器，没有丢失。这时，TCP 模块会从包中提出数据，并存放接收缓冲区中，与上次收到的数据块连接起来。这样一来，数据就被还原成分包之前的状态了<sup>④</sup>。

当收到的数据进入接收缓冲区后，TCP 模块就会生成确认应答的 TCP

① 假设包中的数据为 HTTP 请求消息。

② 6.1.3 节有关于服务器端口号的介绍，请大家回忆一下。

③ 关于序号请参见 2.3.3 节。

④ 拼合数据块的操作在每次收到数据包时都会进行，而不是等所有数据全部接受完毕之后再统一拼合的。



头部，并根据接收包的序号和数据长度计算出 ACK 号，然后委托 IP 模块发送给客户端<sup>①</sup>。

收到的数据块进入接收缓冲区，意味着数据包接收的操作告一段落了。接下来，应用程序会调用 Socket 库的 read (图 6.7 ③) 来获取收到的数据，这时数据会被转交给应用程序。如果应用程序不来获取数据，则数据会被一直保存在缓冲区中，但一般来说，应用程序会在数据到达之前调用 read 等待数据到达，在这种情况下，TCP 模块在完成接收操作的同时，就会执行将数据转交给应用程序的操作。

然后，控制流程会转移到服务器程序，对收到的数据进行处理，也就是检查 HTTP 请求消息的内容，并根据请求的内容向浏览器返回相应的数据。这一部分已经超出了 TCP 模块的范围，我们将在稍后探索服务器程序内部时进行介绍。



收到数据包时，TCP 模块会 (1) 根据收到的包的发送方 IP 地址、发送方端口号、接收方 IP 地址、接收方端口号找到相对应的套接字；(2) 将数据块拼合起来并保存在接收缓冲区中；(3) 向客户端返回 ACK。

### 6.2.5 TCP 模块的断开操作

当数据收发完成后，便开始执行断开操作。这个过程和客户端是一样的，让我们简单复习一下。

在 TCP 协议的规则中，断开操作可以由客户端或服务器任何一方发起，具体的顺序是由应用层协议决定的。Web 中，这一顺序随 HTTP 协议版本不同而不同，在 HTTP1.0 中，是服务器先发起断开操作。

这时，服务器程序会调用 Socket 库的 close，TCP 模块会生成一个控制位 FIN 为 1 的 TCP 头部，并委托 IP 模块发送给客户端。当客户端收到

<sup>①</sup> 在返回 ACK 号之前，会先等待一段时间，看看能不能和后续的应答包合并。

这个包之后，会返回一个 ACK 号。接下来客户端调用 close，生成一个 FIN 为 1 的 TCP 头部发给服务器，服务器再返回 ACK 号，这时断开操作就完成了。HTTP1.1 中，是客户端先发起断开操作，这种情况下只要将客户端和服务器的操作颠倒一下就可以了。

无论哪种情况，当断开操作完成后，套接字会在经过一段时间后被删除。

## 6.3 Web 服务器程序解释请求消息并作出响应

### 6.3.1 将请求的 URI 转换为实际的文件名

图 6.7 展示了服务器程序的工作过程，这个过程不仅限于 Web 服务器，对于各种服务器程序都是共通的，收发数据的过程也是大同小异的。各种服务器程序的不同点在于图中 (b) 客户端通信部分的第一行调用 read 后面的如下部分。

[ 处理请求消息内容 ];

图 6.7 中只写了一行，但实际上这里应该是一组处理各种工作的程序<sup>①</sup>，或者说这里才是服务器程序的核心部分。接下来让我们来对这一部分进行探索。

Web 服务器中，图 6.7 的 read 获取的数据内容就是 HTTP 请求消息。服务器程序会根据收到的请求消息中的内容进行相应的处理，并生成响应消息，再通过 write 返回给客户端。请求消息包括一个称为“方法”的命令，以及表示数据源的 URI (文件路径名)，服务器程序会根据这些内容向客户端返回数据，但对于不同的方法和 URI，服务器内部的工作过程会有所不同。下面我们从简单的开始依次进行介绍。

最简单的一种情况如图 6.8 中的例子所示，请求方法为 GET，URI 为一个 HTML 文件名。这种情况只要从文件中读出 HTML 文档，然后将其

<sup>①</sup> 这部分可能会有几万行代码。

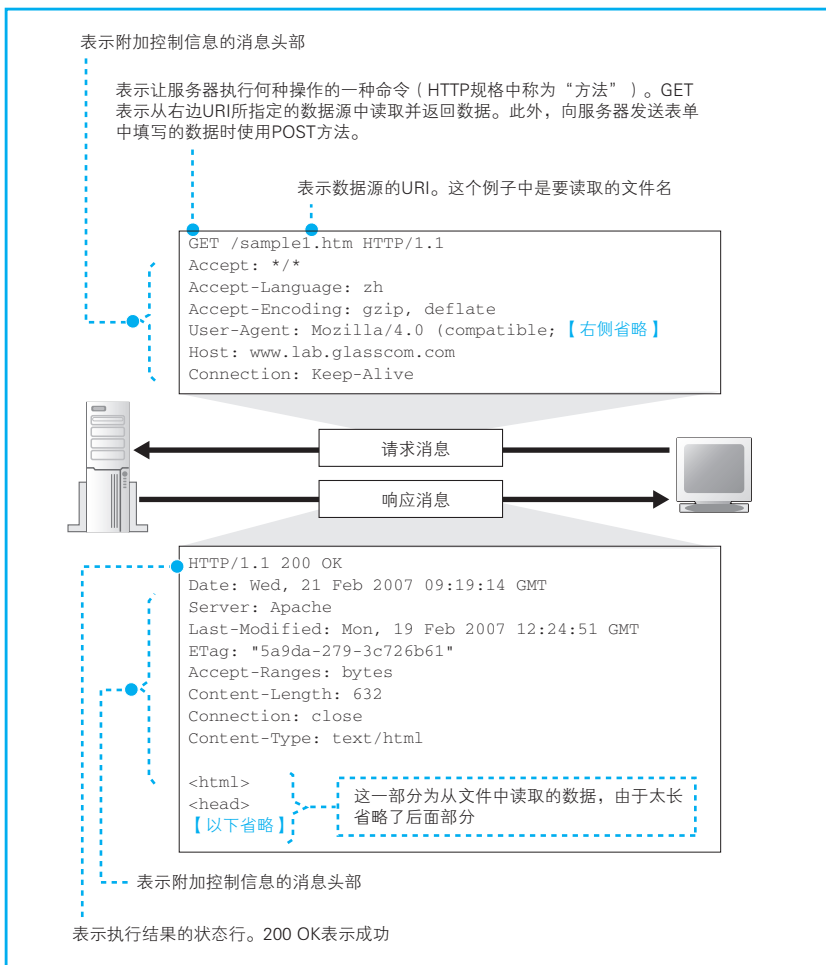


图 6.8 Web 的基本工作方式

作为响应消息返回就可以了。不过，按照 URI 从磁盘上读取文件并没有这么简单。如果完全按照 URI 中的路径和文件名读取<sup>①</sup>，那就意味着磁盘上所

① 对于 UNIX 操作系统的服务器来说，URI 的路径名和磁盘文件的路径名格式是相同的，对于 Windows 也只要将“/”替换成“\”就可以了，因此我们可以将 URI 当作是磁盘文件的路径名。

有的文件都可以访问，Web 服务器的磁盘内容就全部暴露了，这很危险。因此，这里需要一些特殊的机制。

Web 服务器公开的目录其实并不是磁盘上的实际目录，而是如图 6.9 这样的虚拟目录，而 URI 中写的就是在这个虚拟目录结构下的路径名。因此，当读取文件时，需要先查询虚拟目录与实际目录的对应关系，并将 URI 转换成实际的文件名后，才能读取文件并返回数据。举个例子，假设我们的虚拟目录结构如图 6.9 所示，如果请求消息中的 URI 如下页 (1) 所示，那么因为 `/~user2/...` 对应的实际目录为 `/home/user2/...`，所以将 URI 转换成实际文件名后应该是如下页 (2)。

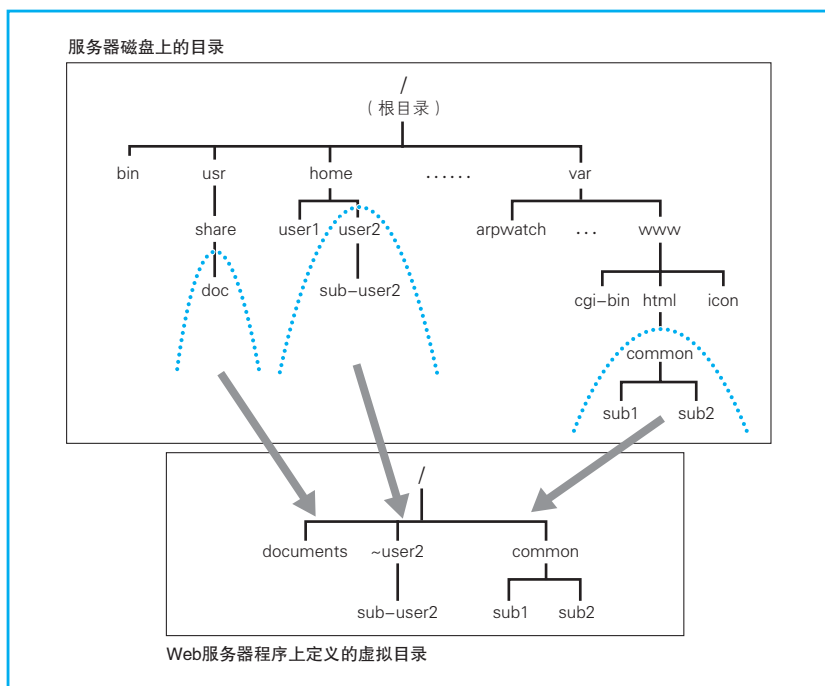


图 6.9 客户端看到的目录结构和实际目录结构是不同的

客户端看到的 Web 服务器目录是虚拟的，和实际的目录结构不同。Web 服务器内部会将实际的目录名和供外部访问的虚拟目录名进行关联。

```
/~user2/sub-user2/sample.html (1)
```

```
/home/user2/sub-user2/sample.html (2)
```

于是，服务器就会根据上述路径从磁盘中读取相应的文件，然后将数据返回给客户端。

文件名转换是有特例的，比如 URI 中的路径省略了文件名的情况，这时服务器会读取事先设置好的默认文件名。例如在浏览器中输入如下网址。

```
http://www.glasscom.com/tone/
```

上面这个网址省略了文件名，服务器会在末尾添加默认文件名，如下。

```
http://www.glasscom.com/tone/index.html
```

在这个例子中，index.html 这个文件名是在服务器中设置好的<sup>①</sup>，服务器会将它添加在目录名的后面。

有些 Web 服务器程序还具有文件名改写功能，只要设置好改写的规则，当 URI 中的路径符合改写规则时，就可以将 URI 中的文件名改写成其他的文件名进行访问<sup>②</sup>。当出于某些原因 Web 服务器的目录和文件名发生变化，但又希望用户通过原来的网址进行访问的时候，这个功能非常有用。

### 6.3.2 运行 CGI 程序

如果 URI 指定的文件内容为 HTML 文档或图片，那么只要直接将文件内容作为响应消息返回客户端就可以了。但 URI 指定的文件内容不仅限于 HTML 文档，也有可能是一个程序。在这个情况下，服务器不会直接返回文件内容，而是会运行这个程序，然后将程序输出的数据返回给客户端。

① 这个文件名是在 Web 服务器配置文件中设置的。尽管这个文件名可以任意设置，但一般来说会设置成类似 index.html、index.cgi、default.htm 等这样的文件名。

② 例如 Web 服务器程序 Apache 就具有这样的功能。

Web 服务器可以启动的程序有几种类型，每种类型的具体工作方式有所区别，下面我们来看看 CGI 程序是如何工作的。

当需要 Web 服务器运行程序时，浏览器发送的 HTTP 请求消息内容和访问 HTML 文档时不太一样，我们先从这里开始讲。Web 服务器运行程序时，一般来说浏览器会将需要程序处理的数据放在 HTTP 请求消息中发送给服务器。这些数据有很多种类，例如购物网站订单表中的品名、数量、发货地址等，搜索引擎中输入的关键字也是一个常见的例子。

总之，浏览器需要在发送给 Web 服务器的请求消息中加入一些数据。我们在第 1 章曾经介绍过有两种加入数据的方法。一种是在 HTML 文档的表单中加上 `method="GET"`，通过 HTTP 的 GET 方法，将输入的数据作为参数添加在 URI 后面发送给服务器。另一种方法是在 HTML 文档的表单中加上 `method="POST"`，将数据放在 HTTP 请求消息的消息体<sup>①</sup>中发送给服务器（图 6.10）。

收到请求消息之后，Web 服务器会进行下面的工作。首先，Web 服务器会检查 URI 指定的文件名，看一看这个文件是不是一个程序。这里的判断方法是在 Web 服务器中事先设置好的，一般是通过文件的扩展名来进行判断，例如将 `.cgi`、`.php` 等扩展名的文件设置为程序，当遇到这些文件时，Web 服务器就会将它们作为程序来对待。也可以设置一个存放程序的目录，将这个目录下的所有文件都作为程序来对待。此外，还可以根据文件的属性来进行判断。

如果判断要访问的文件为程序文件，Web 服务器会委托操作系统运行这个程序，然后从请求消息中取出数据并交给运行的程序<sup>②</sup>。如果方法为 GET，则将 URI 后面的参数传递给程序；如果方法为 POST，则将消息体中的数据传递给程序（图 6.11）。

接下来，运行的程序收到数据后会进行一系列处理，并将输出的数据返回给 Web 服务器。程序可以返回各种内容，如表示订单已接受的说明，

① 即头部字段之后的部分，参见 1.1.5 节。

② 除了数据，还可以将请求消息的头部字段传递给程序。

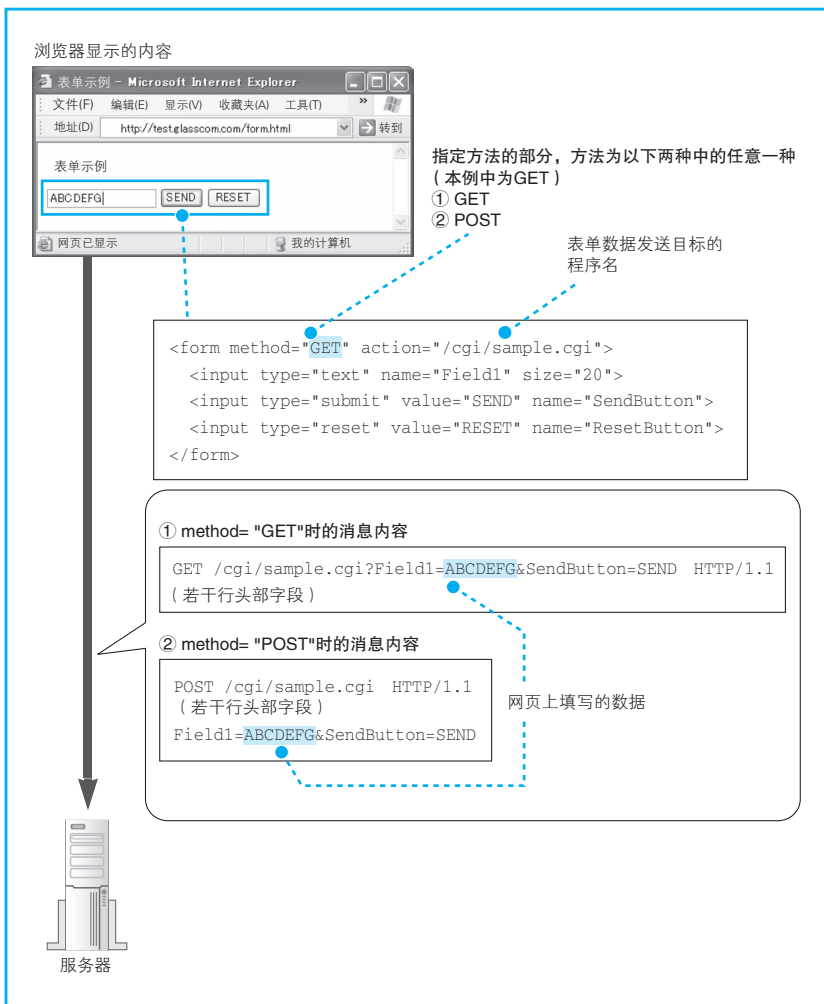


图 6.10 用 HTTP 请求发送表单中输入的内容

或者按照关键字从数据库中搜索出的结果等。无论如何，为了将数据处理的结果返回给客户端，首先需要将它返回给 Web 服务器。这些输出的数据一般来说会嵌入到 HTML 文档中，因此 Web 服务器可以直接将其作为响应消息返回给客户端。输出数据的内容是由运行的程序生成的，Web 服务