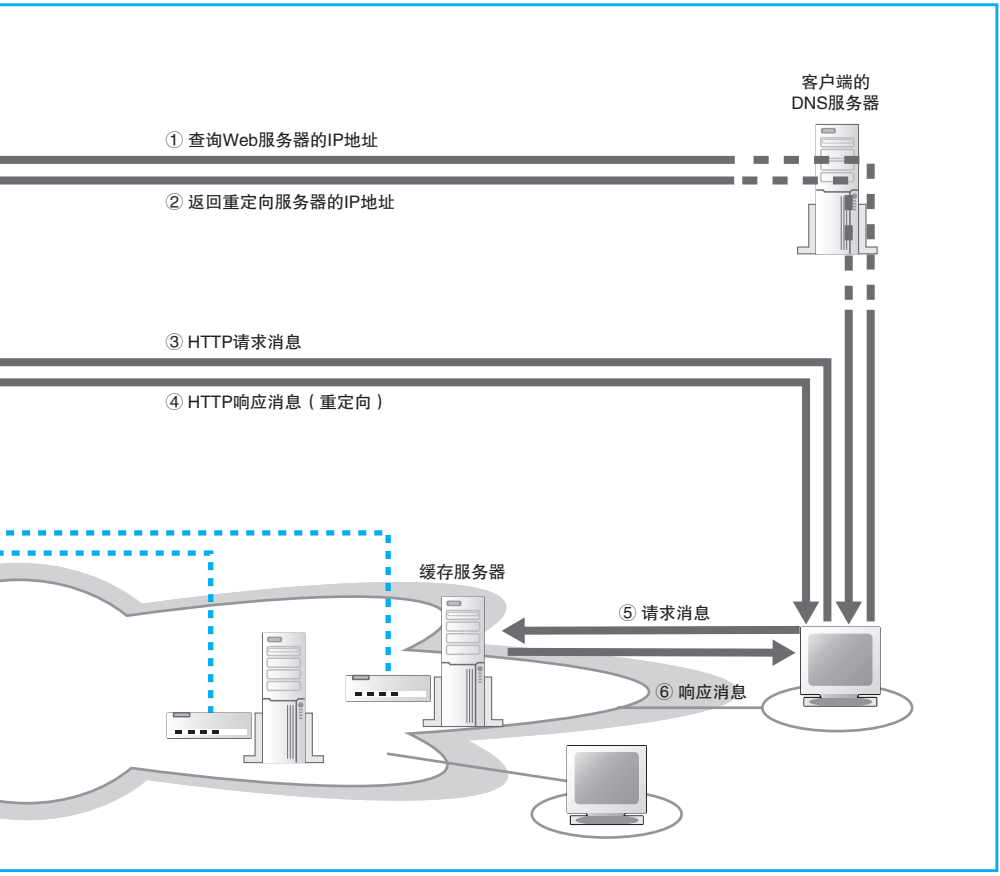
的。这种情况下，我们可以不保存整个页面，而是将应用程序生成的部分，也就是每次内容都会发生变化的动态部分，与内容不会发生变化的静态部分分开，只将静态部分保存在缓存中。

Web 服务器前面存在着各种各样的服务器，如防火墙、代理服务器、缓存服务器等。请求消息最终会通过这些服务器，到达 Web 服务器。Web 服务器接收请求之后，会查询其中的内容，并根据请求生成并返回响应消息。关于这一部分，我们将在下一章进行介绍。



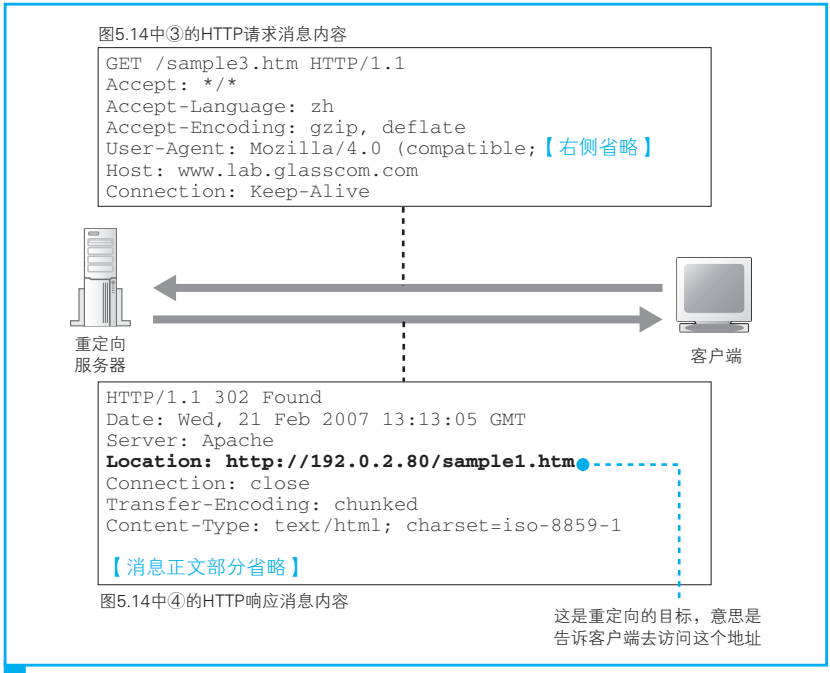



图 5.15　使用重定向时 HTTP 消息的内容

## 小测验

本章的旅程告一段落，我们为大家准备了一些小测验题目，确认一下自己的成果吧。

### 问题

1. 现在主流的防火墙方式叫什么？
2. 当防火墙需要确定应用程序种类时要检查什么信息？
3. 用于分担 Web 服务器负载，将访问分配到多台服务器上的设备叫什么？
4. 部署在服务器端的代理是正向代理还是反向代理？
5. 在互联网中部署多台缓存服务器，并将其租借给 Web 服务器运营者的服务叫什么？

# Column
# 网络术语其实很简单

## 当通信线路变成局域网

**探索队员：**局域网和通信线路有什么不同啊？

**探索队长：**其实局域网这个词本来的意思呢……

**队员：**我知道我知道，我已经查过字典了。不过我的字典里面没有这个词啊，是不是我上学时买的字典太老了？

**队长：**语言日新月异，你该去买本新的啦。

**队员：**说得轻巧，还有一个礼拜才发工资呢……

**队长：**没办法，好吧，我借你一本。

**队员：**哦，查到了，局域网英文全称叫 Local Area Network，这个 local 是中央和地方的那个地方的意思吗？

**队长：**这里的 local 指的是一个小的区域，是“本地”的意思，比如本地线、本地局。局域网是在一幢楼里面使用的，一幢楼就是一小块地方，所以才叫 local。

**队员：**这个我知道的，我问的是它和通信线路有什么不同？

**队长：**通信线路是通信运营商，也就是电话公司部署的遍布全球的线路，所以在世界各地都可以使用，这跟局域网完全是两码事吧？

**队员：**这个我也知道啊，我想问的不是这个，我想问的是，局域网是不是比通信线路更快更便宜？

**队长：**算是吧。

**队员：**那么通信线路为什么不能像局域网一样又快又便宜呢？

**队长：**原来如此，我很理解你的想法，也很赞同，不过事情可没那么简单。

**队员：**为什么呢？

**队长：**回想一下，我们探索 ADSL 的时候曾经说过，离电话局越远，速度就越慢，对不对？

**队员：**对。

**队长：**在一幢楼里，距离近的时候，局域网可以做到又快又便宜，但局域网可不能直接扩展到全世界。

**队员：**噢……

**队长：**看来还是没懂啊？

**队员：**ADSL 确实是越远越慢，但 FTTH 什么的不是跟局域网一样快吗？

**队长：**唔，你还真是不撞南墙不回头。

**队员：**不，我只是想知道真相而已，队长阁下！

**队长：**FTTH 之所以又快又便宜，是因为它能够使用局域网的技术。

**队员：**可是为什么局域网不能用在距离远的地方呢？

**队长：**我就知道你会这么问，所以我才不想回答这种问题啊。刚才我说局域网只能用在近的地方，那是光纤普及之前的事了。

**队员：**什么嘛，原来是老黄历了。（马上又开始倚老卖老了……）

**队长：**你说什么？

**队员：**没……没什么。

**队长：**使用现在的光纤技术，的确可以将线路延伸到几十公里，因此用局域网技术也可以将距离很远的地方连接起来。

**队员：**原来如此。

**队长：**而且，随着网络的普及，现在光纤可以大批量生产了，还很廉价。

**队员：**那我们把通信线路全都换成局域网不就好了吗？

**队长：**也许将来会有这一天吧……

**队长：**哎，真的吗？

**队长：**谁知道呢……

**小测验答案**

1. 包过滤方式（参见【5.2.1】）
2. 端口号（参见【5.2.3】）
3. 负载均衡器（参见【5.3.2】）
4. 反向代理（参见【5.3.7】）
5. 内容分发服务（CDS 或 CDN）（参见【5.3.8】）

# 第6章

# 请求到达 Web 服务器，响应返回浏览器

## ——短短几秒的“漫长旅程”迎来终点

### 热身问答

在开始探索之旅之前，我们准备了一些和本章内容有关的小题目，请大家先试试看。

这些题目是否答得出来并不影响接下来的探索之旅，因此请大家放轻松。


下列说法是正确的（√）还是错误的（×）？

1. 服务器向客户端返回的响应消息不一定和客户端向服务器发送的请求消息通过相同的路由传输。
2. 客户端计算机也可以当作服务器来使用。
3. 一台服务器可以同时用作 Web 服务器和邮件服务器。

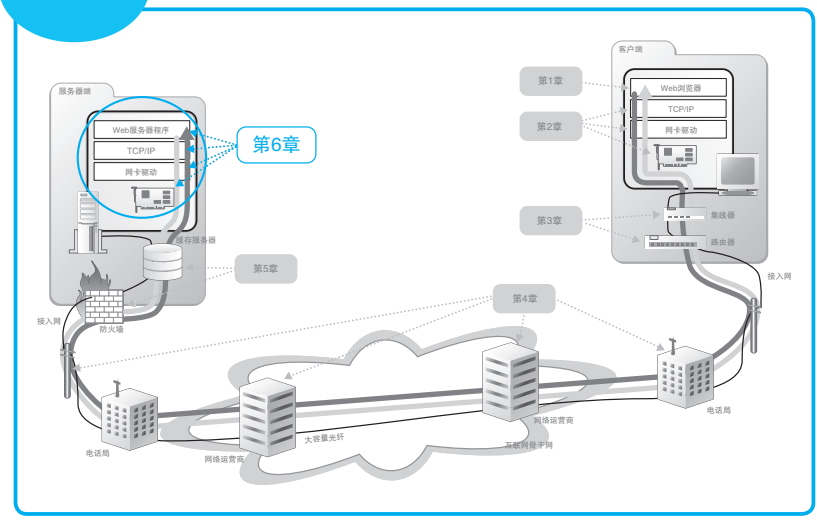
## 答案

1. √。路由器和交换机是不考虑请求包和响应包之间的关联的，而是将它们作为独立的包来对待，因此请求和响应是有可能通过不同的路由来传输的，具体走哪条路由，是由路由器的路由表和交换机的地址表中的配置来决定的。
2. √。无论任何计算机，协议栈的功能和工作方式都是相同的，因此客户端计算机也可以当作服务器来用。不过，客户端计算机和服务器相比其性能和可靠性都比较差，这一点必须要注意。
3. √。由于可以通过端口号来区分服务器上的应用程序，所以一台服务器上可以同时运行多个服务器程序，不仅限于 Web 和邮件。当然，这样做会增加服务器的负载，因此必须注意服务器的性能。

## 前情提要

上一章，我们探索了 Web 服务器前面的防火墙、缓存服务器、负载均衡器等设备，现在网络包已经通过这些设备，到达了 Web 服务器中。本章的探索之旅就从这里开始。

## 探索之旅的看点




### (1) 服务器概览

服务器的职责是响应客户端的请求，但仅从如何响应请求这一点是无法看清服务器的全貌的，这样是无法理解服务器的。因此，我们会先介绍一下服务器程序的整体结构，以及启动后要做的一些准备工作，这样大家就能够了解服务器到底是怎么一回事了。

### (2) 服务器的接收操作

搞清楚服务器的全貌之后，我们来探索一下服务器的协议栈是如何接收数据的。首先我们看一看服务器如何接收信号并将信号还原成数字形式

的网络包，然后从中提取出 HTTP 消息。在第 1 章、第 2 章介绍发送操作的时候我们也提了一些关于接收操作的内容，但那些介绍都比较零散，本章我们将对接收操作做一个整体性的探索。然后，我们将探索协议栈是如何将接收的消息通过 Socket 库传递给 Web 服务器程序的。

### （3）Web 服务器程序解释请求消息并作出响应

Web 服务器程序收到消息后，会查询其中的内容，并按照请求进行处理，将结果返回给客户端。例如，如果请求内容是获取某个网页的数据，那么就读取该文件并取出数据；如果请求某个 CGI 程序，就将相关参数传递给该程序并执行，然后获取程序输出的数据。接下来，这些数据会以响应消息的形式返回给客户端。我们将对上面这一系列操作进行探索。

### （4）浏览器接收响应消息并显示内容

Web 服务器返回的响应消息会通过互联网到达客户端计算机的浏览器。接下来，浏览器会将消息的内容显示在屏幕上。当客户端计算机上显示出网页的内容时，访问 Web 服务器的操作就全部完成了，这也是我们本次探索之旅的终点。

# 6.1 服务器概览

## 6.1.1 客户端与服务器的区别

当网络包到达 Web 服务器之后，服务器就会接收这个包并进行处理，但服务器的操作并不是一下子从这里开始的。在服务器启动之后，需要进行各种准备工作，才能接受客户端的访问。因此，处理客户端发来的请求之前，必须先完成相应的准备工作。要理解服务器的工作方式，搞清楚包括这些准备工作在内的服务器整体结构是很重要的，下面我们就来从整体上介绍一下服务器。

首先，服务器和客户端有什么区别呢？根据用途，服务器可以分为很多种类，其硬件和操作系统与客户端是有所不同的[①]。但是，网络相关的部分，如网卡、协议栈、Socket 库等功能和客户端却并无二致。无论硬件和 OS 如何变化，TCP 和 IP 的功能都是一样的，或者说这些功能规格都是统一的[②]。

不过，它们的功能相同，不代表用法也相同。在连接过程中，客户端发起连接操作，而服务器则是等待连接操作，因此在 Socket 库的用法上还是有一些区别的，即应用程序调用的 Socket 库的程序组件不同[③]。

此外，服务器的程序可以同时和多台客户端计算机进行通信，这也是一点区别。因此，服务器程序和客户端程序在结构上是不同的。

## 6.1.2 服务器程序的结构

服务器需要同时和多个客户端通信，但一个程序来处理多个客户端的请求是很难的，因为服务器必须把握每一个客户端的操作状态[④]。因此一般

① 客户端计算机也可以用作服务器。

② 如果每台设备的工作方式不同，那么网络就会因为混乱而无法工作了。

③ Socket 库和协议栈原本所具有的功能是没有区别的，因此客户端计算机也可以调用用来等待连接的程序零件，服务器程序在客户端计算机上也可以运行。

④ 尽管如此，仅用一个程序来处理多个客户端请求的服务器程序也是存在的，只不过这种程序编写起来难度较高。

的做法是，每有一个客户端连接进来，就启动一个新的服务器程序，确保服务器程序和客户端是一对一的状态。

具体来说，服务器程序的结构如图 6.1 所示。首先，我们将程序分成两个模块，即等待连接模块（图 6.1（a））和负责与客户端通信的模块（图 6.1（b））[①]。当服务器程序启动并读取配置文件完成初始化操作后，就会运行等待连接模块（a）。这个模块会创建套接字，然后进入等待连接的暂停状态。接下来，当客户端连发起连接时，这个模块会恢复运行并接受连接，然后启动客户端通信模块（b），并移交完成连接的套接字。接下来，客户端通信模块（b）就会使用已连接的套接字与客户端进行通信，通信结束后，这个模块就退出了。

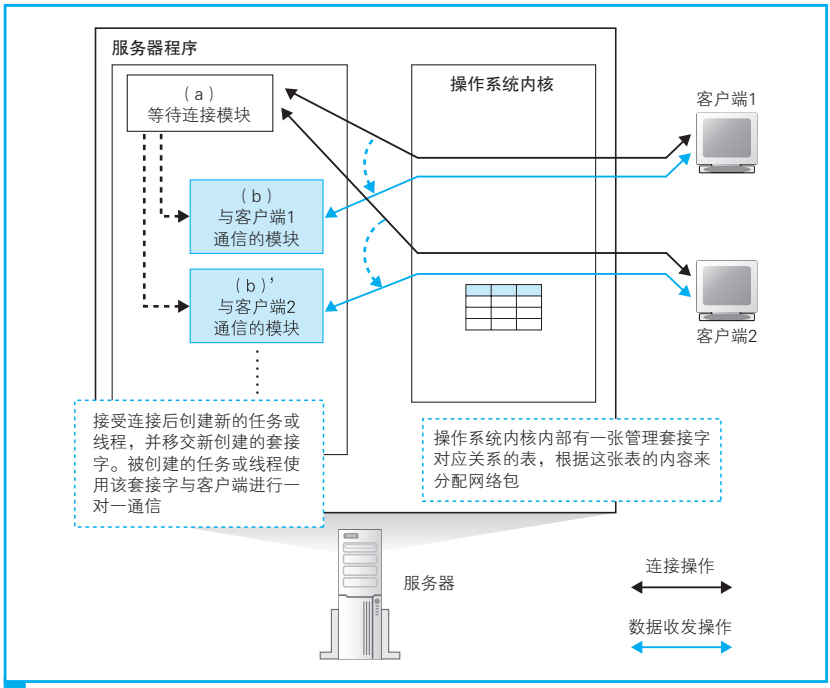



图 6.1　用不同的模块与每个客户端进行通信

① 可以分成两个可执行文件，但一般是在一个程序内部分成两个模块。

每次有新的客户端发起连接，都会启动一个新的客户端通信模块（b），因此（b）与客户端是一对一的关系。这样，（b）在工作时就不必考虑其他客户端的连接情况，只要关心自己对应的客户端就可以了。通过这样的方式，可以降低程序编写的难度。服务器操作系统具有多任务[①]、多线程[②]功能，可以同时运行多个程序[③]，服务器程序的设计正是利用了这一功能。

当然，这种方法在每次客户端发起连接时都需要启动新的程序，这个过程比较耗时，响应时间也会相应增加。因此，还有一种方法是事先启动几个客户端通信模块，当客户端发起连接时，从空闲的模块中挑选一个出来将套接字移交给它来处理。

### 6.1.3 服务器端的套接字和端口号

刚才我们介绍了服务器程序的大体结构，但如果不深入挖掘调用 Socket 库的具体过程，我们还是无法理解服务器是如何使用套接字来完成通信的。因此，下面就来看一看服务器程序是如何调用 Socket 库的。

首先，我们再来回忆一下客户端与服务器的区别。从数据收发的角度来看，区分“客户端”和“服务器”这两个固定的角色似乎不是一个好办法。现在大多数应用都是由客户端去访问服务器，但其实应用的形态不止这一种。为了能够支持各种形态的应用，最好是在数据收发层面不需要区分客户端和服务器，而是能够以左右对称的方式自由发送数据。TCP 也正是在这样的背景下设计出来的。

不过，这其中还是存在一个无法做到左右对称的部分，那就是连接操作。连接这个操作是在有一方等待连接的情况下，另一方才能发起连接，

① 多任务：操作系统提供的一种功能，可以让多个任务（程序）同时运行。实际上，一个处理器在某一个瞬间只能运行一个任务，但通过短时间内在不同的任务间切换，看起来就好像是同时运行多个任务一样。有些操作系统称之为“多进程”。

② 多任务和多线程的区别在于任务和线程的区别。在操作系统内部，任务是作为单独的程序来对待的，而线程则是一个程序中的一部分。

③ 客户端操作系统也具有多任务和多线程功能。

如果双方同时发起连接是不行的，因为在对方没有等待连接的状态下，无法单方面进行连接。因此，只有这个部分必须区分发起连接和等待连接这两个不同的角色。从数据收发的角度来看，这就是客户端与服务器的区别，也就是说，发起连接的一方是客户端，等待连接的一方是服务器。

这个区别体现在如何调用 Socket 库上。首先，客户端的数据收发需要经过下面 4 个阶段。

（1）创建套接字（创建套接字阶段）
（2）用管道连接服务器端的套接字（连接阶段）
（3）收发数据（收发阶段）
（4）断开管道并删除套接字（断开阶段）

相对地，服务器是将阶段（2）改成了等待连接，具体如下。

（1）创建套接字（创建套接字阶段）
（2-1）将套接字设置为等待连接状态（等待连接阶段）
（2-2）接受连接（接受连接阶段）
（3）收发数据（收发阶段）
（4）断开管道并删除套接字（断开阶段）

下面我们像前面介绍客户端时一样[①]，用伪代码来表示这个过程，如图 6.2 所示。我们一边看图，一边介绍一下服务器端的具体工作过程。

首先，协议栈调用 socket 创建套接字（图 6.2（1）），这一步和客户端是相同的[②]。

接下来调用 bind 将端口号写入套接字中（图 6.2（2-1））。在客户端发起连接的操作中，需要指定服务器端的端口号，这个端口号也就是在这一步设置的。具体的编号是根据服务器程序的种类，按照规则来确定的，例如

① 参见 2.1.3 节。
② 创建套接字操作的本质是分配用于套接字的内存空间，这一点上客户端和服务器是一样的。1.4.2 节有相关介绍。

Web 服务器使用 80 号端口[①]。

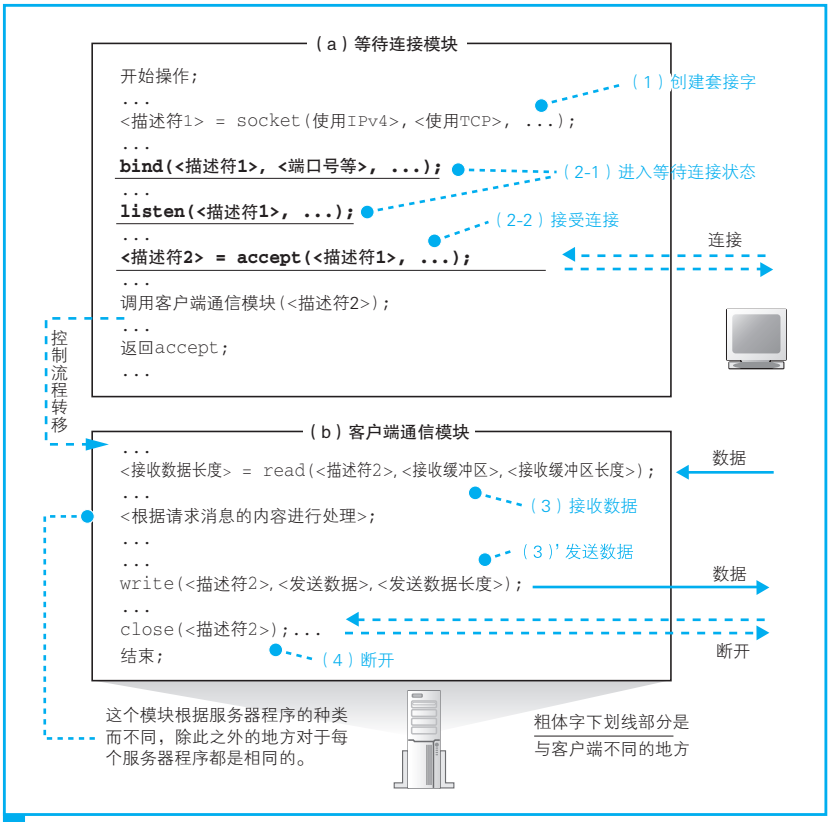



图 6.2 服务器程序的通信操作

设置好端口号之后，协议栈会调用 listen 向套接字写入等待连接状态这一控制信息（图 6.2（2-1））。这样一来，套接字就会开始等待来自客户端的连接网络包。

① Socket 库和协议栈其实并不受到这个规则的制约，它们只负责向套接字写入 bind 所指定的端口号，并等待来自该端口的连接。因此，我们也可以让 Web 服务器工作在 80 号之外的其他端口上。只不过，在这种情况下，客户端必须在 TCP 头部中指定这个端口号才能够完成连接。

然后，协议栈会调用 accept 来接受连接（图 6.2（2-2））。由于等待连接的模块在服务器程序启动时就已经在运行了，所以在刚启动时，应该还没有客户端的连接包到达。可是，包都没来就调用 accept 接受连接，可能大家会感到有点奇怪，不过没关系，因为如果包没有到达，就会转为等待包到达的状态，并在包到达的时候继续执行接受连接操作。因此，在执行 accept 的时候，一般来说服务器端都是处于等待包到达的状态，这时应用程序会暂停运行。在这个状态下，一旦客户端的包到达，就会返回响应包并开始接受连接操作。接下来，协议栈会给等待连接的套接字复制一个副本，然后将连接对象等控制信息写入新的套接字中（图 6.3）。刚才我们介绍了调用 accept 时的工作过程，到这里，我们就创建了一个新的套接字，并和客户端套接字连接在一起了。

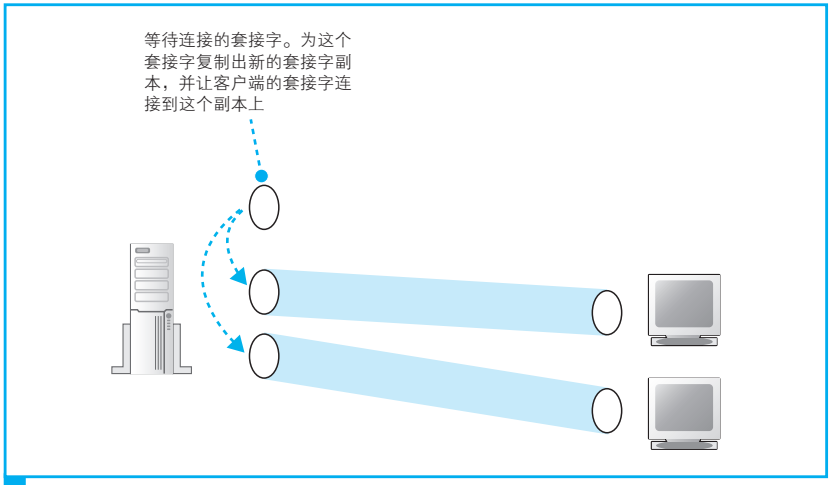



图 6.3　服务器端的套接字

当 accept 结束之后，等待连接的过程也就结束了，这时等待连接模块会启动客户端通信模块，然后将连接好的新套接字转交给客户端通信模块，由这个模块来负责执行与客户端之间的通信操作。之后的数据收发操作和刚才说的一样，与客户端的工作过程是相同的。

其实在这一系列操作中，还有一部分没有讲到，那就是在复制出一个新的套接字之后，原来那个处于等待连接状态的套接字会怎么样呢？其实它还会以等待连接的状态继续存在，当再次调用 accept，客户端连接包到达时，它又可以再次执行接受连接操作。接受新的连接之后，和刚才一样，协议栈会为这个等待连接的套接字复制一个新的副本，然后让客户端连接到这个新的副本套接字上。像这样每次为新的连接创建新的套接字就是这一步操作的一个关键点。如果不创建新副本，而是直接让客户端连接到等待连接的套接字上，那么就没有套接字在等待连接了，这时如果有其他客户端发起连接就会遇到问题。为了避免出现这样的情况，协议栈采用了这种创建套接字的新副本，并让客户端连接到这个新副本上的方法。

此外，创建新套接字时端口号也是一个关键点。端口号是用来识别套接字的，因此我们以前说不同的套接字应该对应不同的端口号，但如果这样做，这里就会出现问题。因为在接受连接的时候，新创建的套接字副本就必须和原来的等待连接的套接字具有不同的端口号才行。这样一来，比如客户端本来想要连接 80 端口上的套接字，结果从另一个端口号返回了包，这样一来客户端就无法判断这个包到底是要连接的那个对象返回的，还是其他程序返回的。因此，新创建的套接字副本必须和原来的等待连接的套接字具有相同的端口号。

但是这样一来又会引发另一个问题。端口号是用来识别套接字的，如果一个端口号对应多个套接字，就无法通过端口号来定位到某一个套接字了。当客户端的包到达时，如果协议栈只看 TCP 头部中的接收方端口号，是无法判断这个包到底应该交给哪个套接字的。

这个问题可以用下面的方法来解决，即要确定某个套接字时，不仅使用服务器端套接字对应的端口号，还同时使用客户端的端口号再加上 IP 地址，总共使用下面 4 种信息来进行判断（图 6.4）。

- 客户端 IP 地址
- 客户端端口号