第 1 章　Java 网络编程入门

所有上过网的人都熟悉这样的过程：打开浏览器程序，输入一个 URL 地址，这个地址指向的网页就会从远程 Web 服务器传到客户端，然后在浏览器中显示出来。网络编程的最基础的任务就是开发像浏览器这样的客户程序，以及像 Web 服务器这样的服务器程序，并且使两者能有条不紊地交换数据。

张三给李四邮寄一封信，张三不必亲自把信送到李四家里，送信的任务由邮政网络来完成。张三只需提供李四的地址，邮政网络就会准确的把这封信送达目标地址。同样，服务器程序与客户程序只需关心发送什么样的数据给对方，而不必考虑如何把这些数据传输给对方，传输数据的任务由计算机网络来完成，参见图 1-1。

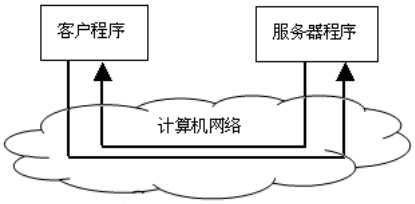

图 1-1　计算机网络负责传输通信数据

由此可见，网络应用程序建立在计算机网络的基础上。本章介绍了计算机网络的一些基本概念，重点介绍了网络的分层思想和 TCP/IP 协议。最后以 EchoServer 和 EchoClient 为例，介绍如何利用套接字来创建一个简单的 Java 服务器程序和 Java 客户程序。

1.1　进程之间的通信

进程是指运行中的程序，进程的任务就是执行程序中的代码。以下例程 1-1 的 EchoPlayer 类是一个独立的 Java 程序，它可以在任意一台安装了 JDK 的主机上运行。EchoPlayer 类不断读取用户从控制台输入的任意字符串 XXX，然后输出 echo:XXX。如果用户输入的字符串为“bye”，就结束程序。例程 1-1 为 EchoPlayer 类的源代码。

例程 1-1　EchoPlayer.java

```
import java.io.*;
public class EchoPlayer {
  public String echo(String msg) {
    return "echo:"+msg;
  }
  public void talk()throws IOException {
    BufferedReader br=
            new BufferedReader(new InputStreamReader(System.in));
    String msg=null;
    while((msg=br.readLine())!=null){
      System.out.println(echo(msg));
      if(msg.equals("bye"))  //当用户输入"bye"，结束程序
        break;
    }
```

```
    }

    public static void main(String arg[])throws IOException{
      new EchoPlayer().talk();
    }
  }
```

运行“java EchoPlayer”命令，就启动了 EchoPlayer 进程，该进程执行 EchoPlayer 类的 main()方法。图 1-2 演示了 EchoPlayer 进程的运行过程，它从本地控制台中获得标准输入流和标准输出流。本地控制台为用户提供了基于命令行的用户界面，用户通过控制台与 EchoPlayer 进程交互。

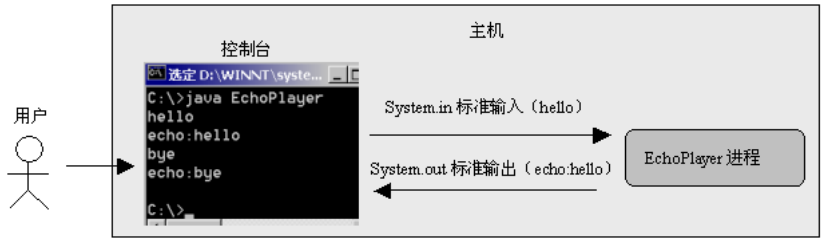

图 1-2　EchoPlayer 进程的运行过程

确切地说，运行“java EchoPlayer”命令，就启动了一个 JVM（Java Virtual Machine，Java 虚拟机）进程，该进程执行 EchoPlayer 类的 main()方法。本书为了叙述的方便，把运行 EchoPlayer 类的 main()方法的 JVM 进程直接称为 EchoPlayer 进程。

EchoPlayer 类的 echo(String msg)方法负责生成响应结果。如果需要把生成响应结果的功能（即 echo(String msg)方法）移动到一个远程主机上，那么上面的 EchoPlayer 类无法满足这一需求。在这种情况下，要创建两个程序：客户程序 EchoClient 和服务器程序 EchoServer。EchoClient 程序有两个作用：

- 与用户交互，从本地控制台获得标准输入流和标准输出流。
- 与远程的 EchoServer 通信，向 EchoServer 发送用户输入的字符串，接收 EchoServer 返回的响应结果，再把响应结果写到标准输出流。

EchoServer 程序负责接收 EchoClient 发送的字符串，然后把响应结果发送给 EchoClient。图 1-3 演示了 EchoClient 与 EchoServer 的通信过程。客户机和远程服务器是通过网络连接的两台主机。客户机上运行 EchoClient 进程，远程服务器上运行 EchoServer 进程。

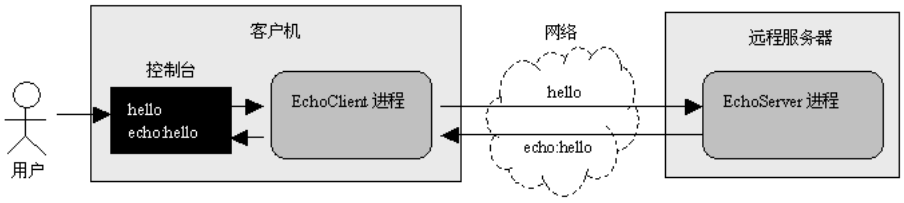

图 1-3　EchoClient 与 EchoServer 通信

张三给李四打电话，两者顺利通话的前提条件是他们各自的电话机都连接到了电话网络上。张三和李四只需关注他们谈话的具体内容，而不必考虑如何把自己的话音传输到对方的电话机上。传输语音信息的任务是由电话网络来完成的。

同样，两个进程顺利通信的前提条件是它们所在的主机都连接到了计算机网络上。EchoClient 与 EchoServer 只需关注它们通信的具体内容，例如 EchoClient 发送信息“hello”，那么 EchoServer 返回信息“echo:hello”。EchoClient 和 EchoServer 都无需考虑如何把信息传

输给对方。传输信息的任务是由计算机网络来完成的。

Java 开发人员的任务是编写 EchoClient 和 EchoServer 程序，接下来在两台安装了 JDK 的主机上分别运行它们，两个进程就会有条不紊地通信。

由于进程之间的通信建立在计算机网络的基础上，Java 开发人员有必要对计算机网络有基本的了解，这有助于更容易的掌握 Java 网络编程技术。

1.2　计算机网络的概念

计算机网络是现代通信技术与计算机技术相结合的产物。所谓计算机网络，是指把分布在不同地理区域的计算机用通信线路互联起来的一个具有强大功能的网络系统。在计算机网络上，众多计算机可以方便地互相通信，共享硬件、软件和数据信息等资源。通俗地说，计算机网络就是通过电缆、电话线、或无线通讯设施等互联的计算机的集合。

网络中每台机器称为节点（node）。大多数节点是计算机，此外，打印机、路由器、网桥、网关和哑终端等也是节点。在本书中，用节点一词指代网络中的任意一个设备，用主机指代网络中的计算机节点。

如图 1-4 所示，人与人之间通过某种语言来交流，网络中的主机之间也通过“语言”来交流，这种语言称为网络协议，这是对网络协议的通俗解释，在本章第 1.3 节，还会更深入地介绍网络协议的概念。

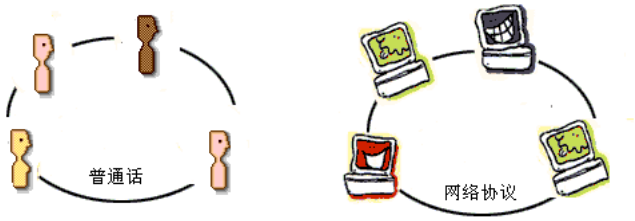

图 1-4　网络协议是网络中主机之间通信的语言

网络中的每个主机都有地址，它是用于标识主机的一个字节序列。字节序列越长，可以表示的地址数目就越多，这意味着可以有更多的设备连入网络。

按照计算机连网的区域大小，可以把网络分为局域网（LAN，Local Area Network）和广域网（WAN，Wide Area Network）。局域网（LAN）是指在一个较小地理范围内的各种计算机互联在一起的通信网络，可以包含一个或多个子网，通常局限在几千米的范围之内。例如在一个房间、一座大楼，或是在一个校园内的网络可称为局域网。广域网（WAN）连接地理范围较大，常常是一个国家或是一个洲，其目的是为了让分布较远的各局域网互联。

到 Internet 海洋去冲浪，如今已成为一种时尚。Internet 是指国际互联网，也叫做因特网，是全球范围内的广域网，为世界上不同类型的计算机交换数据提供了通信媒介。

Internet 目前已经覆盖了世界上的大部分国家和地区，连接着数百万子网和上亿台电脑主机，Internet 的用户超过十亿。Internet 上汇聚了成千上万的信息资源，成为世界上信息资源最丰富的公共计算机网络。

Internet 提供的服务包括 WWW（World-Wide Web）服务、电子邮件（E-mail）服务、文件传输（FTP）服务和远程登录（Telnet）服务等等，全球用户可以通过这些服务，来获取 Internet 上的信息或者开展各种业务。

Internet 是由许多小的网络互联成的国际性大网络，在各个小网络内部使用不同的协议，那么如何使不同的网络之间能进行信息交流呢？如图 1-5 所示，上海人讲上海方言，广东人讲广东方言，上海人与广东人用普通话沟通。与此相似，不同网络之间的互联靠网络上的标

准语言—TCP/IP 协议。如图 1-6 所示，一个网络使用协议 A，另一个网络使用协议 B，这两个网络通过 TCP/IP 协议进行互联。本章第 1.4 节对 TCP/IP 协议做了进一步介绍。

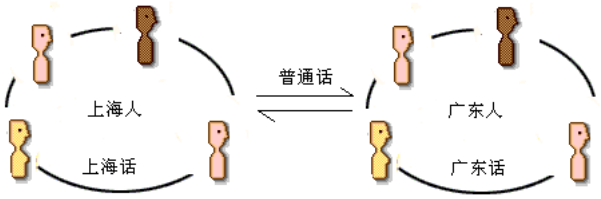

图 1-5　上海人与广东人用普通话沟通

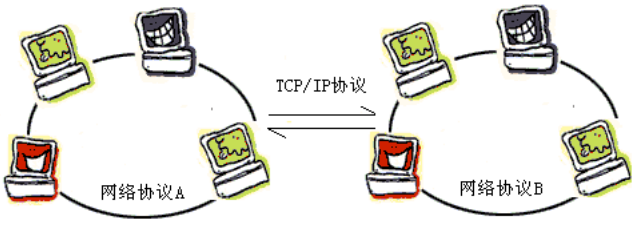

图 1-6　不同的网络通过 TCP/IP 协议互联

1.3　OSI 参考模型

在计算机网络产生之初，每个计算机厂商都有一套自己的网络体系结构，它们之间互不相容。为此，国际标准化组织（ISO，International Organization for Standization）在 1979 年建立了一个分委员会，来专门研究一种用于开放系统互联（Open System Interconnection,简称 OSI）的体系结构，“开放”这个词意味着：一个网络系统只要遵循 OSI 模型，就可以和位于世界上任何地方的、也遵循 OSI 模型的其他网络系统连接。这个分委员会提出了 OSI 参考模型，它为各种异构系统互联提供了概念性的框架。

OSI 参考模型把网络分为 7 层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层，参见图 1-7。每一层使用下层提供的服务，并为上层提供服务。

不同主机之间的相同层称为对等层。例如主机 A 中的表示层和主机 B 中的表示层互为对等层，主机 A 中的会话层和主机 B 中的会话层互为对等层。

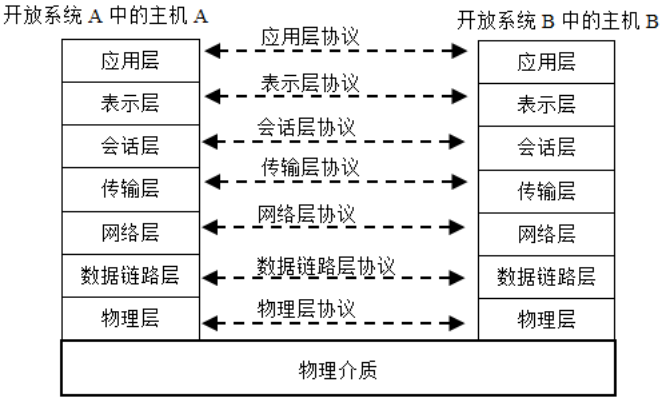

图 1-7　OSI 参考模型的分层结构

OSI 参考模型中各层的主要功能如下。

（1）物理层（Physical Layer）

传输信息离不开物理介质，如双纽线和同轴电缆等，但物理介质并不在 OSI 的 7 层之内，有人把物理介质当作 OSI 的第零层。物理层的任务就是为它的上一层提供物理连接，以及规定通信节点之间的机械和电气等特性，如规定电缆和接头的类型，传送信号的电压等。在这一层，数据作为原始的比特（bit）流传输。本层的典型设备是 Hub（集线器）。

（2）数据链路层（Data Link Layer）

数据链路层负责在两个相邻结点间的线路上，无差错的传送以帧为单位的数据。每一帧包括一定数量的数据和一些必要的控制信息。数据链路层要负责建立、维持和释放数据链路的连接。在传送数据时，如果接收方检测到所传数据中有差错，就要通知发送方重发这一帧。本层的典型设备是 Switch（交换机）。

（3）网络层（Network Layer）

在计算机网络中进行通信的两个计算机之间可能会经过很多个数据链路，也可能还要经过很多通信子网。网络层的任务就是选择合适的网间路由和交换结点，确保数据及时传送到目标主机。网络层将数据链路层提供的帧组成数据包，包中封装有网络层包头，包头中含有逻辑地址信息—源主机和目标主机的的网络地址。本层的典型设备是 Router（路由器）。

如图 1-8 所示，主机 A 发送的数据先后经过节点 1 和节点 4，最后到达主机 B。相领两个节点之间的线路称为数据链路，比如主机 A 与节点 1、节点 1 与节点 4，以及节点 4 与主机 B 之间的线路都是数据链路。数据链路层负责数据链路上的数据传输。从主机 A 到主机 B 的整个路径称为路由，网络层负责选择合适的路由。

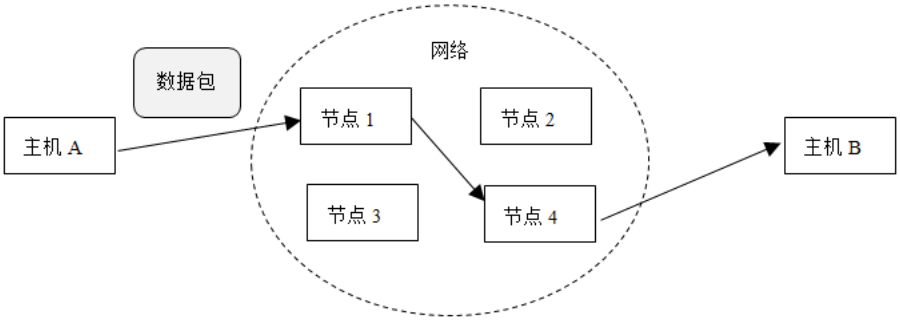

图 1-8　从主机 A 到主机 B 的路由以及数据链路

（4）传输层（Transport Layer）

该层的任务是根据通信子网的特性来充分利用网络资源，为两个端系统（也就是源主机和目标主机）的会话层提供建立、维护和取消传输连接的功能，以可靠方式或者不可靠方式传输数据。所谓可靠方式，是指保证把源主机发送的数据正确地送达目标主机；所谓不可靠方式，则是指不保证把源主机发送的数据正确地送达目标主机，数据有可能丢失，或出错。在这一层，信息的传送单位是报文。

（5）会话层（Session Layer）

这一层也可以称为会晤层或对话层，在会话层及以上层次中，数据传送的单位不再另外命名，统称为报文。会话层管理进程之间的会话过程，即负责建立、管理、终止进程之间的会话。会话层还通过在数据中插入校验点来实现数据的同步。

（6）表示层（Presentation Layer）

表示层对上层数据进行转换,以保证一个主机的应用层的数据可以被另一个主机的应用程理解。表示层的数据转换包括对数据的加密、解密、压缩、解压和格式转换等。

（7）应用层（Application Layer）

应用层确定进程之间通信的实际用途，以满足用户实际需求。浏览 Web 站点、收发 E-Mail、上传或下载文件以及远程登入服务器等都可以看作是进程之间通信的实际用途。

如图 1-9 所示，当源主机向目标主机发送数据，在源主机方，数据先由上层向下层传递，每一层会给上一层传递来的数据加上一个信息头（header），然后向下层发出，最后通过物理介质传输到目标主机，在目标主机方，数据再由下层向上层传递，每一层先对数据进行处理，把信息头去掉，再向上层传输，最后到达最上层，就会还原成实际的数据。各个层加入的信息头有着不同的内容，比如网络层加入的信息头中包括源地址和目标地址信息；传输层加入的信息头中包括报文类型、源端口和目标端口、序列号和应答号等。在图 1-9 中，AH、PH、SH、TH、NH 和 DH 分别表示各个层加入的信息头，数据链路层还会为数据加上信息尾 DT。

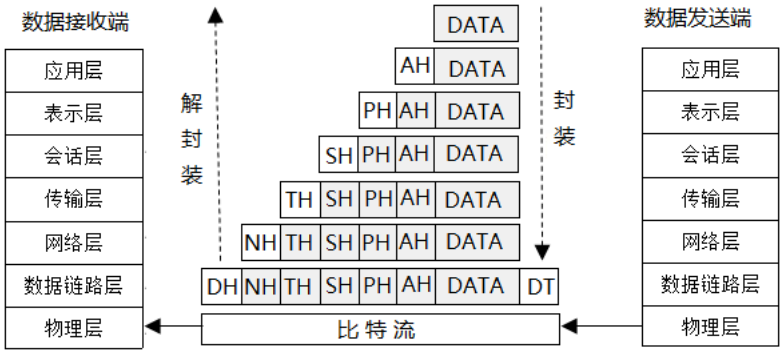

图 1-9　数据在上下层之间的封装和解封装过程

在发送方，数据由上层向下层传递，每一层把数据封装后再传给下层；在接收方，数据由下层向上层传递，每一层把数据解封装后再传给上层。在生活中，也常常采用这种方式来传输实际物品。比如张三给李四邮寄一封信，真正要传输的内容是信，为了保证信能正确到达目的地，在发送方，需要把信封装到一个信封中，上面写上发信人和收信人地址。邮件到了接收方，需要拆开信封，才能得到里面的信件。

OSI 参考模型把网络分为多个层次，每个层次有明确的分工，这简化了网络系统的设计过程。例如在设计应用层时，只需考虑如何创建满足用户实际需求的应用，在设计传输层时，只需考虑如何在两个主机之间传输数据，在设计网络层时，只需考虑如何在网络上找到一条发送数据的路径，即路由。

对等层之间互相通信需要遵守一定的规则，如通信的内容和通信的方式，这种规则称为网络协议（Protocol）。值得注意的是，OSI 参考模型并没有具体的实现方式，它没有在各层制定网络协议，但它为其他计算机厂商或组织制定网络协议提供了参考框架。网络的各个层次都有相应的协议，以下归纳了 OSI 各个层的一些典型协议，这些协议均由第三方提供：

- 物理层协议：EIA/TIA RS-232、EIA/TIA RS-449、V.35、RJ-45 等。
- 数据链路层协议：HDLC、PPP、IEEE 802.3/802.2 等。
- 网络层协议：IP、IPX、ICMP、IGMP、AppleTalk DDP 等。
- 传输层协议：TCP、UDP、SPX 等。
- 会话层协议：NetBIOS、RPC、NFS、AppleTalk 等。
- 表示层协议：ASCII、GIF、JPEG、MPEG 等。
- 应用层协议：TELNET、FTP、HTTP、SNMP、SMTP 等。

1.4　TCP/IP 参考模型和 TCP/IP 协议

国际标准化组织 ISO 制定的 OSI 参考模型提出了网络分层的思想，这种思想对网络的发展具有重要的指导意义。但由于 OSI 参考模型过于庞大和复杂，使它难以投入到实际运用中。与 OSI 参考模型相似的 TCP/IP 参考模型吸取了网络分层的思想，但是对网络的层次做了简化，并且在网络各层（除了主机-网络层外）都提供了完善的协议，这些协议构成了 TCP/IP 协议集，简称 TCP/IP 协议。TCP/IP 协议是目前最流行的商业化协议，相对于 OSI，它是当前的工业标准或“事实标准”， TCP/IP 协议主要用于广域网，在一些局域网中也有运用。

TCP/IP 参考模型是美国国防部高级研究计划局计算机网（Advanced Research Project

Agency Network，ARPANET）以及后来的 Internet 使用的参考模型。ARPANET 是由美国国防部（U.S．Department of Defense，DoD）赞助的研究网络。最初，它只连接了美国境内的四所大学。随后的几年中，它通过租用的电话线连接了数百所大学和政府部门。最终，ARPANET 发展成为全球规模最大的互联网络—Internet。最初的 ARPANET 则于 1990 年永久关闭。图 1-10 把 TCP/IP 参考模型和 OSI 参考模型作了对比。

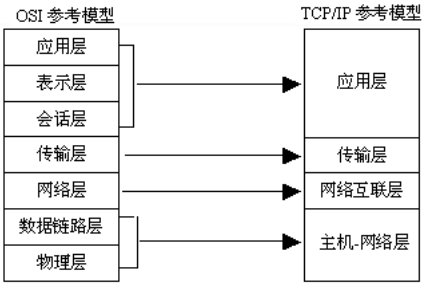

图 1-10 比较 TCP/IP 参考模型和 OSI 参考模型

TCP/IP 参考模型分为四个层次：应用层、传输层、网络互联层和主机-网络层。在每一层都有相应的协议。确切地说，TCP/IP 协议应该称为 TCP/IP 协议集，它是 TCP/IP 参考模型的除了主机-网络层以外的其他三层的协议的集合，而 IP 协议和 TCP 协议则是协议集的最核心的两个协议。表 1-1 列出了各层的主要协议，其中主机-网络层的协议是由第三方提供的。

表 1-1 TCP/IP 参考模型的各层的协议

应用层	FTP、TELNET、HTTP	SNMP、DNS
传输层	TCP	UDP
网络互联层	IP	
主机-网络层	以太网： IEEE802.3	
	令牌环网：IEEE802.4	

在 TCP/IP 参考模型中，去掉了 OSI 参考模型中的会话层和表示层，这两层的功能被合并到应用层，同时将 OSI 参考模型中的数据链路层和物理层合并到主机-网络层。下面分别介绍各层的主要功能。

（1）主机-网络层

实际上 TCP/IP 参考模型没有真正提供这一层的实现，也没有提供协议。它只是要求第三方实现的主机-网络层能够为上层—网络互联层提供一个访问接口，使得网络互联层能利用主机-网络层来传递 IP 数据包。

IEEE（Institute of Electrical and Electronics Engineer，美国电气及电子工程师学会）制定了 IEEE802.3 和 IEEE802.4 协议集，它们位于 OSI 参考模型的物理层和数据链路层，相当于位于 TCP/IP 参考模型的主机-网络层。采用 IEEE802.3 协议集的网络称为以太网，采用 IEEE802.4 协议集的网称为令牌环网。以太网和令牌环网都向网络互联层提供了访问接口。

（2）网络互联层

网络互联层是整个参考模型的核心。它的功能是把 IP 数据包发送到目标主机。为了尽快地发送数据，IP 协议把原始数据分为多个数据包，然后沿不同的路径同时传递数据包。如图 1-11 所示，由主机 A 发出的原始数据被分为三个数据包，然后沿不同的路径到达主机 B，可谓殊途同归。数据包到达的先后顺序和发送的先后顺序可能不同，这就需要上层—传输层对数据包重新排序，还原为原始数据。

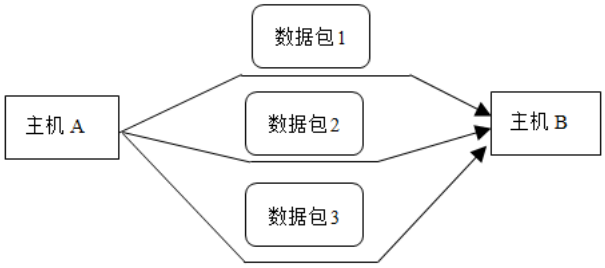

图 1-11　三个数据包沿不同的路径到达主机 B

网络互联层具备连接异构网的功能。图 1-12 显示了连接以太网和令牌环网的方式。以太网和令牌环网是不同类型的网，两者有不同的网络拓扑结构。以太网和令牌环网都向网络互联层提供了通一的访问接口，访问接口向网络互联层隐藏了下层网络的差异，使得两个网络之间可以顺利传递数据包。

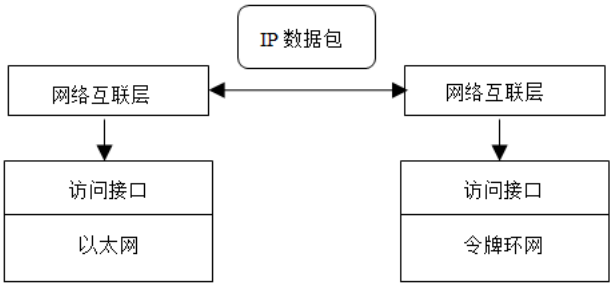

图 1-12　网络互联层连接以太网和令牌环网

网络互联层采用 IP 协议（Internet Protocol），它规定了数据包的格式，并且规定了为数据包寻找路由的流程。

（3）传输层

传输层的功能是使源主机和目标主机上的进程可以进行会话。在传输层定义了两种服务质量不同的协议，即 TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Datagram Protocol，用户数据报协议）。TCP 协议是一种面向连接的、可靠的协议。它将源主机发出的字节流无差错地发送给互联网上的目标主机。在发送端，TCP 协议负责把上层传送下来的数据分成报文段并传递给下层。在接收端，TCP 协议负责把收到的报文进行重组后递交给上层。TCP 协议还要处理端到端的流量控制，以避免接收速度缓慢的接收方没有足够的缓冲区来接收发送方发送的大量数据。应用层的许多协议，如 HTTP、FTP 和 TELNET 协议等都建立在 TCP 协议基础上。

UDP 协议是一个不可靠的、无连接协议，主要适用于不需要对报文进行排序和流量控制的场合。UDP 不能保证数据报的接收顺序同发送顺序相同，甚至不能保证它们是否全部到达目标主机。应用层的一些协议，如 SNMP 和 DNS 协议就建立在 UDP 协议基础上。如果要求可靠的传输数据，则应该避免使用 UDP 协议，而要使用 TCP 协议。

（4）应用层

TCP/IP 模型将 OSI 参考模型中的会话层和表示层的功能合并到应用层实现。针对各种各样的网络应用，应用层引入了许多协议。基于 TCP 协议的应用层协议主要包括：

- FTP（File Transfer Protocol）：文件传输协议，允许在网络上传输文件。
- TELNET：虚拟终端协议，允许从主机 A 登入到远程主机 B，使得主机 A 充当远

程主机 B 的虚拟终端。

- HTTP（Hyper Text Transfer Protocol）：超文本传输协议，允许在网络上传送超文本，本书第 5 章（创建非阻塞的 HTTP 服务器）对此做了进一步介绍。
- HTTPS（Secure Hypertext Transfer Protocol）：安全超文本传输协议，允许在网络上安全地传输超文本，网上传输的是经过加密的数据，到达目的地后再对数据解秘。
- POP3（Post Office Protocol - Version 3）：邮局协议-版本 3，允许用户在客户程序中访问在远程服务器上的电子邮件。
- IMAP4（Internet Message Access Protocol Version 4）：Internet 消息访问协议-版本 4，允许用户访问和操纵远程服务器上的邮件和邮件夹。IMAP4 改进了 POP3 的不足，用户可以通过浏览信件头来决定是不是要下载此信，还可以在服务器上创建或更改文件夹或邮箱，删除信件或检索信件的特定部分。在 POP3 中，信件是保存在服务器上的，当用户阅读信件时，所有内容都会被立刻下载到用户的机器上。IMAP4 服务器可以看成是一个远程文件服务器，而 POP3 服务器可以看成是一个存储转发服务器。
- SMTP（Simple Mail Transfer Protocol）：简单邮件传送协议，是发送电子邮件的协议。

基于 UDP 协议的的应用层协议主要包括：

- SNMP（Simple Network Management Protocol）：网络管理协议，为管理本地和远程的网络设备提供了一个标准化途径，是分布式环境中的集中化管理协议。
- DNS（Domain Name System）：域名系统协议，把主机的域名转换为对应的 IP 地址。

1.4.1 IP 协议

IP 网络（即在网络层采用 IP 协议的网）中每台主机都有唯一的 IP 地址，IP 地址用于标识网络中的每个主机。IP 地址分为两种：IPv4 和 IPv6

（1）IPv4

用 32 位的二进制序列来表示主机地址。为了便于在上层应用中方便地表示 IP 地址，可以把 32 位的二进制序列分为四个单元，每个单元占 8 位，然后用十进制整数来表示每个单元，这些十进制整数的取值范围是 0~255。如某一台主机的 IP 地址可为：192.166.3.4。

相对于 IPv6，IPv4 是更早期出现的 IP 地址形式，但现在仍然使用广泛。本书后文默认情况下采用 IPv4 地址。

（2）IPv6

用 128 位的二进制序列来表示主机地址，大大扩充了可用地址的数目，IPv6 是新一代的互联网络 IP 地址标准。

IPv6 的 128 位地址通常分成 8 段，每段为四个十六进制数。例如：

```
AD80:0000:0000:0000:ABAA:0000:00C2:0002
```

以上地址比较长，不易于阅读和书写。零压缩法可以用来缩减其长度。如果有一个段或几个连续段的值都是 0，那么这些 0 就可以简单地以“::”来表示。上述地址可写成：

```
AD80::ABAA:0000:00C2:0002
```

值得注意的是，这种零压缩简化只能用一次。例如上例中的“ABAA”后面的“0000”就不能再次简化。这种限制的目的是为了能准确还原被压缩的 0，不然就无法确定每个“::”代表多少个 0。

另外，如果一个段中包含 4 个 0，那么可以用一个 0 来表示。以下三个 IPv6 地址是等

价的：

```
2000:0000:0000:0000:0000:0000:0000:1
2000:0:0:0:0:0:0:1
2000::1
```

此外，段中前导的零也可以省略，因此以下两个 IPv6 地址是等价的：

```
2001:0DB8:02DE::0E13
2001:DB8:2DE::E13
```

在 IPv6 地址中可以嵌入 IPv4 地址，用 IPv6 和 IPv4 的混合体来表示。例如：

```
::FFFF:192.168.89.9
```

在以上地址中，“::FFFF”采用 IPv6 形式，“192.168.89.9”采用 IPv4 形式。以上地址等价于以下的 IPv6 地址：

```
0000:0000:0000:0000:0000:FFFF:C0A8:5909
```

IP 地址的组成

下面以 IPv4 为例，介绍 IP 地址的组成。IP 地址由两部分组成：IP 网址和 IP 主机地址。IP 网址表示网络的地址，IP 主机地址表示网络中的主机的地址。网络掩码用来确定 IP 地址中哪部分是网址，哪部分是主机地址。

网络掩码的形式与 IP 地址相同，但有一定的限制。在网络掩码的二进制序列中，前面部分都为 1，后面部分都为 0。假定 IP 地址 192.166.3.4 的网络掩码为 255.255.255.0。这个网络掩码的二进制序列为 11111111.11111111. 11111111.00000000。把网络掩码与 IP 地址进行二进制与操作，得到的结果就是 IP 网址。因此，IP 地址 192.166.3.4 的网址为 192.166.3.0。如果把网络掩码设为 255.255.0.0，那么 IP 网址为 192.166.0.0。

在 Internet 上，每个主机必须有全球范围内唯一的 IP 地址。国际机构 NIC（Internet Network Information Center）统一负责全球地址的规划和管理，与此同时，InterNIC、APNIC、RIPE 和 CNNIC 等机构具体负责美国及全球其他地区的 IP 地址分配。中国地区的 IP 地址分配由 CNNIC 机构负责，它的网址为：http://www.cnnic.net.cn。

图 1-13 显示了两个互联的网络的配置，从该图可以看出，每个网络都有 IP 网址，两个网络之间用路由器连接。

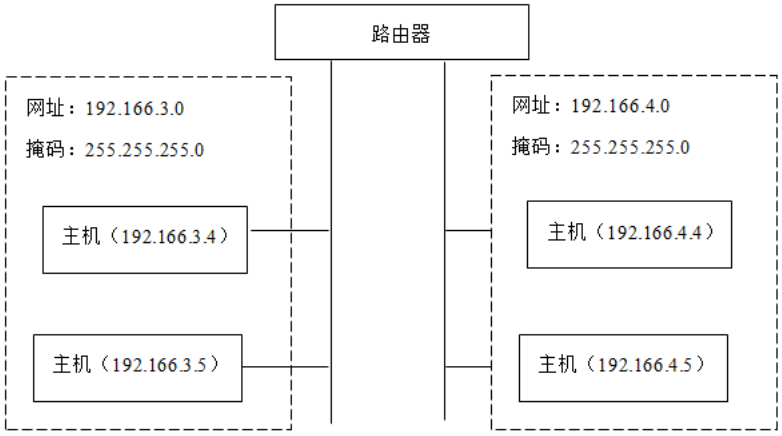

图 1-13　每个 IP 网络有自己的网址，通过路由器与其他网络连接

子网划分

一个公司可能拥有一个网址和多个主机。例如，如果网址为 192.166.0.0，则可以有 2^{16}（即 65536）个主机加入网中。为了更好地管理网络，提高网络性能和安全性，可以把网络划分为多个子网。子网可包括某地理位置内（如某大楼或相同局域网中）的所有主机。例如对于网址为“192.166.0.0”的网络，可从整个网络中分出三个子网，这三个子网的网址分别为：192.166.1.0，192.166.2.0 和 192.166.3.0，这些子网的掩码都为 255.255.255.0。

发送数据包的过程

IP 是面向包的协议，即数据被分成若干小数据包，然后分别传输它们。IP 网络上的主机只能直接向本地网上的其他主机（也就是具有相同 IP 网址的主机）发送数据包。主机实际上有两个不同性质的地址：物理地址和 IP 地址。物理地址是由主机上的网卡来标识的，物理地址才是主机的真实地址。如图 1-14 所示，主机 A 向同一个网络上的另一个主机 B 发包时，会通过地址解析协议（ARP，Address Resolution Protocol）获得对方的物理地址，然后把包发给对方。ARP 协议的运行机制为：主机 A 在网络上广播一个 ARP 消息：“要寻找地址为 192.166.3.5 的主机”，接着，具有这个 IP 地址的主机 B 就会做出响应，把自身的物理地址告诉主机 A。

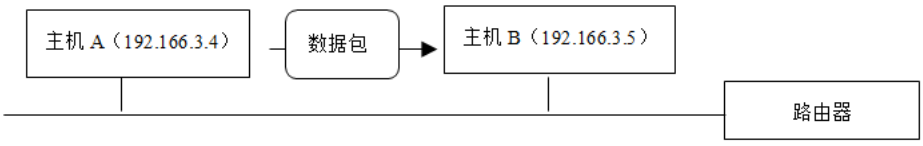

图 1-14　在同一个网络中，主机 A 直接向主机 B 发送 IP 数据包

当主机 A 向另一个网络上的主机 B 发送包时，主机 A 利用 ARP 协议找到本地网络上的路由器的物理地址，把包转发给它。路由器会按照如下步骤处理数据包：

（1）如果数据包的生命周期已到，则该数据包被抛弃。

（2）搜索路由表，优先搜索路由表中的主机，如果能找到具有目标 IP 地址的主机，则将数据包发送给该主机。

（3）如果匹配主机失败，则继续搜索路由表，匹配同子网的路由器，如果找到匹配的路由器，则将数据包转发给该路由器。

（4）如果匹配同子网的路有器失败，则继续搜索路由表，匹配同网络的路由器，如果找到匹配的路由器，则将数据包转发给该路由器。

（5）如果以上匹配操作都失败，就搜索默认路由，如果默认路由存在，则按照默认路由发送数据包，否则丢弃数据包。

从以上路由器的处理步骤可以看出，IP 协议并不保证一定把数据包送达目标主机，在发送过程中，会因为数据包结束生命周期，或者找不到路由而丢弃数据包。

域名

尽管 IP 地址能够唯一标识网络上的主机，但 IP 地址是数字型的，用户记忆数字型的 IP 地址很不方便，于是人们又发明了另一种字符型的地址，即所谓的域名（Domain Name）。域名地址具有易于理解的字面含义，便于记忆。IP 地址和域名一一对应。例如 JavaThinker 网站的域名为 www.javathinker.net，对应的 IP 地址为：43.247.68.17。

域名是从右至左来表述其意义的，最右边的部分为顶层域，最左边的则是这台主机的机器名称。域名一般可表示为：主机机器名.单位名.网络名.顶层域名。如：mail.xyz.edu.cn，这里的 mail 是 xyz 学校的一个主机的机器名，xyz 代表一个学校的名字，edu 代表中国教育科研网，cn 代表中国，顶层域一般是网络机构或所在国家地区的名称缩写。

DNS（Domain Name System）协议采用 DNS 服务器来提供把域名转换为 IP 地址的服务。DNS 服务器分布在网络的各个地方，它们存放了域名与 IP 地址的映射信息。用户需要访问网络上某个主机时，只需提供主机的直观的域名， DNS 协议首先请求地理上比较近的 DNS 服务器进行域名到 IP 地址的转换，如果在该服务器中不存在此域名信息，DNS 协议再让远方的 DNS 服务器提供服务。

URL（统一资源定位器）

URL 是 Uniform Resource Locator 的缩写，表示统一资源定位器。它是专为标识网络上资源位置而设的一种编址方式，大家熟悉的网页地址就属于 URL。URL 一般由三部分组成：

```
应用层协议://主机 IP 地址或域名/资源所在路径/文件名
```

例如 JavaThinker 网站提供的 JDK 安装软件包的 URL 为：

```
http://www.javathinker.net/software/jdk8.exe
```

其中“http”指超文本传输协议，“www.javathinker.net”是 Web 服务器的域名，“software”是文件所在路径，“jdk8.exe”才是相应的文件。

在 URL 中，常见的应用层协议还包括 ftp 和 file 等，比如：

```
ftp://www.javathinker.net/image/
file:///C:\tomcat\webapps\javathinker\index.jsp
```

以上 file 协议用于访问本地计算机上的文件，使用这种协议的 URL 以“file:///”开头。

1.4.2 TCP 协议以及端口

IP 协议在发送数据包时，途中会遇到各种状况，例如可能路由器突然崩溃，使包丢失。再例如一个包可能沿低速链路移动，而另一个包可能沿高速链路移动而超过前面的包，最后使得包的顺序搞乱。

TCP 协议使两台主机上的进程顺利通信，不必担心包丢失或包顺序搞乱。TCP 跟踪包顺序，并且在包顺序搞乱时按正确顺序重组包。如果包丢失，则 TCP 会请求源主机重发包。

如图 1-15 所示，两台主机上都会运行许多进程。当主机 A 上的进程 A1 向主机 B 上的进程 B1 发送数据时，IP 协议根据主机 B 的 IP 地址，把进程 A1 发送的数据送达主机 B。接下来 TCP 需要决定把数据发送到主机 B 中的哪个进程。TCP 采用端口来区分进程。端口不是物理设备，而是用于标识进程的逻辑地址，更确切地说，是用于标识 TCP 连接的端点的逻辑地址。当两个进程进行一次通信，就意味着建立了一个 TCP 连接，TCP 连接的两个端

点用端口来标识。在图 1-15 中，进程 A1 与进程 B1 之间建立了一个 TCP 连接，进程 B1 的端口为 80，因此进程 B1 的地址为：主机 B:80。进程 A1 的端口为 1000，因此进程 A1 的地址为：主机 A :1000。每个进程有了唯一的地址，TCP 就能保证把数据顺利送达到特定的进程。

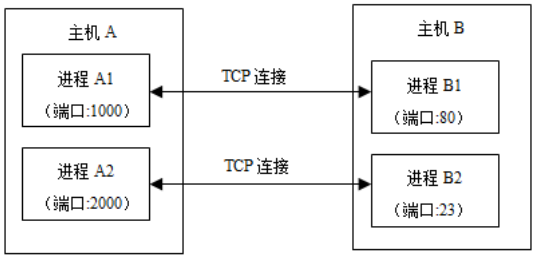

图 1-15　TCP 采用端口来区分进程间的通信

在客户/服务器模型中，客户进程可能会与服务器进程同时建立多个 TCP 连接，在客户端，每一个 TCP 连接都会分配一个端口。参见本章 1.5.2 节的图 1-21。

端口号的范围为 0 到 65535，其中 0 到 1023 的端口号一般固定分配给一些服务。比如 21 端口分配给 FTP 服务，25 端口分配给 SMTP（简单邮件传输）服务，80 端口分配给 HTTP（超级文本传输）服务，135 端口分配给 RPC（远程过程调用）服务等等。

从 1024 到 65535 的端口号供用户自定义的服务使用。比如假定本章范例中的 EchoServer 服务使用 8000 端口。当 EchoServer 程序运行时，就会占用 8000 端口，当程序运行结束，就会释放所占用的端口。

客户进程的端口一般由所在主机的操作系统动态分配，当客户进程要求与一个服务器进程进行 TCP 连接，操作系统为客户进程随机地分配一个还未被占用的端口，当客户进程与服务器进程断开连接，这个端口就被释放。

此外还要指出的是，TCP 和 UDP 都用端口来标识进程。在一个主机中，TCP 端口与 UDP 端口的取值范围是各自独立的，允许存在取值相同的 TCP 端口与 UDP 端口。如图 1-16 所示，在主机 A 中，进程 A1 占用 FTP 端口 1000，进程 A2 占用 UDP 端口 1000，这是允许的。

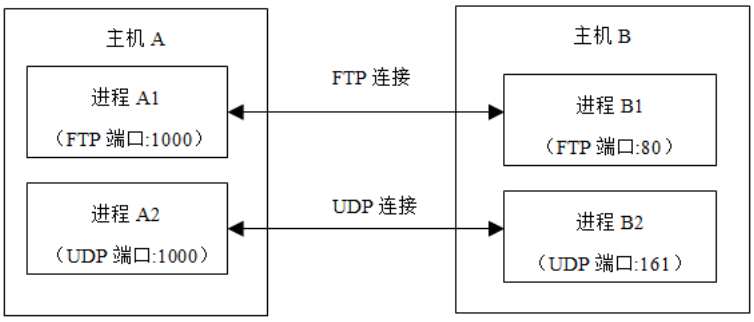

图 1-16　TCP 端口与 UDP 端口的取值范围各自独立

1.4.3　RFC 简介

TCP/IP 协议是以 RFC（Request For Comment）文档的形式发布的。RFC 是描述互联网相关技术规范的文档。

RFC 由个人编写，这些人自愿编写某一新协议或规范的提议草案，并提交给 IETF（The Internet Engineering Task Force ，Internet 工程任务组织）。IETF 负责审阅和发布这些统称为

RFC 的文档，每个文档都有一个 RFC 编号，并且处于以下六种类型之一：

- 标准协议：Internet 的官方标准协议。
- 标准协议草案：正在积极的考虑和审阅以便成为标准协议。
- 标准协议提议：将来可能变成标准协议。
- 实验性协议：为实验目的而设计的协议。实验性协议不投入实际运用。
- 报告性协议：由其他标准组织开发的协议。
- 历史性协议：已经过时的协议，被其他协议代替。

FRC 的官方网站为：http://www.ietf.org/rfc.html。在该网站上已经发布了 8000 多份 RFC 文档。表 1-2 列出了与 TCP/IP 协议相关的 RFC 文档编号。

表 1-2　与 TCP/IP 协议相关的 RFC 文档编号

RFC 编号	协议
768	用户数据报协议 (UDP)
783	日常文件传输协议 (TFTP)
791	Internet 协议 (IP)
792	Internet 控制消息协议 (ICMP)
793	传输控制协议 (TCP)
821	邮件传输协议（SMTP）
826	地址解析协议 (ARP)
854	Telnet 协议 (TELNET)
862	回应协议 (ECHO)
959	文件传输协议 (FTP)
1157	简单网络管理协议 (SNMP)
1939	邮局协议-版本 3（POP3）
1945	超级文本传输协议-版本 1.0（HTTP/1.0）
2060	Internet 消息访问协议-版本 4（IMAP4）
2068	超级文本传输协议-版本 1.1（HTTP/1.1）
7540	超级文本传输协议-版本 2（HTTP/2）

在 FRC 的官方网站上输入网址：http://www.ietf.org/rfc/rfcXXXX.txt，就能察看相关的 FRC 文档，这里的 XXXX 表示文档编号。例如 FTP 协议的 RFC 文档的网址为：http://www.ietf.org/rfc/rfc959.txt。

RFC 文档一旦正式发布，其编号和内容就不允许改变。如果需要更新 RFC 文档，则会对更新后的 RFC 文档赋予新的编号，再将它发布。例如 HTTP1.0 协议对应的 RFC 文档为 RFC1945，它的升级版本 HTTP1.1 协议对应的 RFC 文档为 RFC2068。

1.4.4　客户/服务器通信模式

TCP/UDP 协议推动了客户/服务器通信模式的广泛运用。在通信的两个进程中，一个进程为客户进程，另一个进程为服务器进程。客户进程向服务器进程发出要求某种服务的请求，服务器进程响应该请求。如图 1-17 所示，通常，一个服务器进程会同时为多个客户进程服务，图中服务器进程 B1 同时为客户进程 A1、A2 和 B2 提供服务。以下伪代码演示了服务器进程的大致工作流程：

```
while(true){
  监听端口，等待客户请求;
  响应客户请求;
}
```

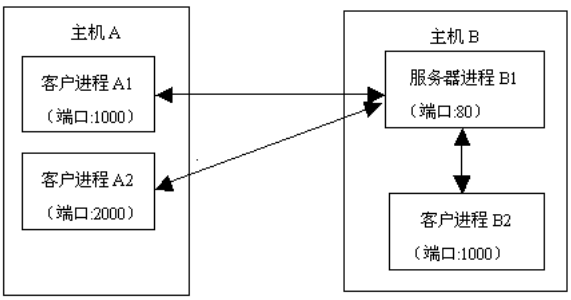

图 1-17　客户进程 A1、A2 和 B2 请求服务器进程 B1 的服务

服务器进程可以提供各种各样的服务，例如本章 1.1 节提到的 EchoServer 提供的服务为：根据 EchoClient 发出的字符串 XXX，返回字符串“echo: XXX”。除了象 EchoServer 这样的由用户自定义的服务外，网络上还有许多众所周知的通用服务，最典型的要算 HTTP 服务。网络应用层的协议规定了客户程序与这些通用服务器程序的通信细节，例如 HTTP 协议规定了 HTTP 客户程序发出的请求的格式，还规定了 HTTP 服务器程序发回的响应的格式。

在现实生活中，有些重要的服务机构的电话是固定的，这有助于人们方便地记住电话和获得服务，比如众所周知的电话 110、120 和 119 分别是报警、急救和火警电话。同样，在网络上有些通用的服务有着固定的端口，表 1-3 对常见的服务以及相应的协议和端口做了介绍。

表 1-3　应用层的一些通用服务使用的端口

服务	端口	协议
文件传输服务	21	FTP
远程登入服务	23	TELNET
传输邮件服务	25	SMTP
用于万维网（WWW）的超文本传输服务	80	HTTP
访问远程服务器上的邮件服务	110	POP3
互联网消息存取服务	143	IMAP4
安全的超文本传输服务	443	HTTPS
安全的远程登入服务	992	TELNETS
安全的互联网消息存取服务	993	IMAPS

1.5　用 Java 编写客户/服务器程序

本书介绍的 Java 网络程序都建立在 TCP/IP 协议基础上，致力于实现应用层。传输层向应用层提供了套接字 Socket 接口，Socket 封装了下层的数据传输细节，应用层的程序通过 Socket 来建立与远程主机的 TCP 连接以及进行数据传输。

站在应用层的角度，两个进程之间的一次通信过程从建立 TCP 连接开始，接着交换数据，到断开连接结束。套接字可看过是通信线路两端的收发器，进程通过套接字来收发数据，参见图 1-18。

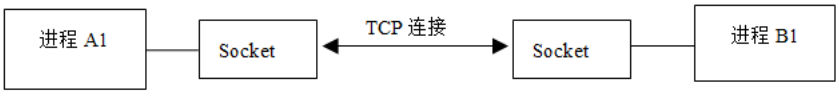

图 1-18　套接字可看过是通信线路两端的收发器

在图 1-18 中，如果把进程 A1 和程 B1 比作两个人，那么图中的两个 Socket 就像两个人各自持有的电话机的话筒，只要拨通了电话，两个人就能通过各自的话筒进行通话。

在 Java 中，有三种套接字类：java.net.Socket、java.net.ServerSocket 和 DatagramSocket。其中 Socket 和 ServerSocket 类建立在 TCP 协议基础上，DatagramSocket 类建立在 UDP 协议基础上。Java 网络程序都采用客户/服务通信模式。

本节以 EchoServer 和 EchoClient 为例，介绍如何用 ServerSocket 和 Socket 来编写服务器程序和客户程序。

在本书中，服务器程序有时简称为服务器，客户程序有时简称为客户。如果没有特别说明，服务器是指服务器程序，而不是指运行服务器程序的主机。

1.5.1 创建 EchoServer

服务器程序通过一直监听端口，来接收客户程序的连接请求。在服务器程序中，需要先创建一个 ServerSocket 对象，在构造方法中指定监听的端口：

```
ServerSocket server=new ServerSocket(8000); //监听 8000 端口
```

ServerSocket 的构造方法负责在操作系统中把当前进程注册为服务器进程。服务器程序接下来调用 ServerSocket 对象的 accept()方法，该方法一直监听端口，等待客户的连接请求，如果接收到一个连接请求，accept()方法就会返回一个 Socket 对象，这个 Socket 对象与客户端的 Socket 对象形成了一条通信线路：

```
Socket socket=server.accept();  //等待客户的连接请求
```

Socket 类提供了 getInputStream()方法和 getOutputStream()方法，分别返回输入流 InputStream 对象和输出流 OutputStream 对象。程序只需向输出流写数据，就能向对方发送数据；只需从输入流读数据，就能接收来自对方的数据。图 1-19 演示了服务器与客户利用 ServerSocket 和 Socket 来通信的过程。

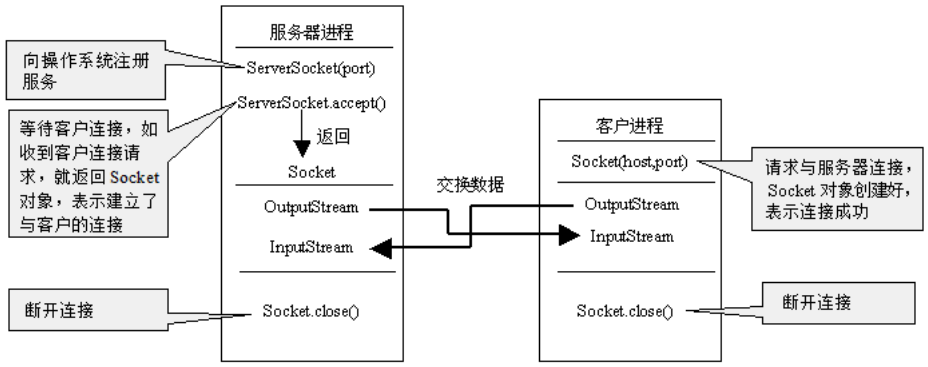

图 1-19　服务器与客户利用 ServerSocket 和 Socket 来通信

与普通 I/O 流一样，Socket 的输入流和输出流也可以用过滤流来装饰。在以下代码中，先获得输出流，然后用 PrintWriter 装饰它，PrintWriter 的 println()方法能够写一行数据；以下代码接着获得输入流，然后用 BufferedReader 装饰它，BufferedReader 的 readLine()方法能够读入一行数据：

```
OutputStream socketOut = socket.getOutputStream();
//参数 true 表示每写一行，PrintWriter 缓存就自动溢出，把数据写到目的地
PrintWriter pw=PrintWriter(socketOut,true);
```

```
InputStream socketIn = socket.getInputStream();
BufferedReader br=new BufferedReader(new InputStreamReader(socketIn));
```

以下例程 1-2 是 EchoServer 的源程序。

例程 1-2　EchoServer.java

```
import java.io.*;
import java.net.*;
public class EchoServer {
  private int port=8000;
  private ServerSocket serverSocket;

  public EchoServer() throws IOException {
    serverSocket = new ServerSocket(port);
    System.out.println("服务器启动");
  }

  public String echo(String msg) {
    return "echo:" + msg;
  }

  private PrintWriter getWriter(Socket socket)throws IOException{
    OutputStream socketOut = socket.getOutputStream();
    return new PrintWriter(socketOut,true);
  }

  private BufferedReader getReader(Socket socket)throws IOException{
    InputStream socketIn = socket.getInputStream();
    return new BufferedReader(new InputStreamReader(socketIn));
  }

  public void service() {
    while (true) {
      Socket socket=null;
      try {
        socket = serverSocket.accept();  //等待客户连接
        System.out.println("New connection accepted "
                    +socket.getInetAddress() + ":" +socket.getPort());
        BufferedReader br =getReader(socket);
        PrintWriter pw = getWriter(socket);

        String msg = null;
        while ((msg = br.readLine()) != null) {
          System.out.println(msg);
          pw.println(echo(msg));
          if (msg.equals("bye")) //如果客户发送的消息为“bye”，就结束通信
            break;
        }
      }catch (IOException e) {
         e.printStackTrace();
      }finally {
         try{
          if(socket!=null)socket.close();  //断开连接
```

```
            }catch (IOException e) {e.printStackTrace();}
          }
        }
      }

      public static void main(String args[])throws IOException {
        new EchoServer().service();
      }
    }
```

EchoServer 类最主要的方法为 service()方法，它不断等待客户的连接请求，当 serverSocket.accept()方法返回一个 Socket 对象，就意味着与一个客户建立了连接。接下来从 Socket 对象中得到输出流和输入流，并且分别用 PrintWriter 和 BufferedReader 来装饰它们。然后不断调用 BufferedReader 的 readLine()方法读取客户发来的字符串 XXX，再调用 PrintWriter 的 println()方法向客户返回字符串 echo:XXX。当客户发来的字符串为“bye”，就会结束与客户的通信，调用 socket.close()方法断开连接。

1.5.2 创建 EchoClient

在 EchoClient 程序中，为了与 EchoServer 通信，需要先创建一个 Socket 对象：

```
String host="localhost";
String port=8000;
Socket socket=new Socket(host,port);
```

在以上 Socket 的构造方法中，参数 host 表示 EchoServer 进程所在的主机的名字，参数 port 表示 EchoServer 进程监听的端口。当参数 host 的取值为“localhost”，表示 EchoClient 与 EchoServer 进程运行在同一个主机上。如果 Socket 对象成功创建，就表示建立了 EchoClient 与 EchoServer 之间的连接。接下来，EchoClient 从 Socket 对象中得到了输出流和输入流，就能与 EchoServer 交换数据。

例程 1-3 为 EchoClient 的源程序。

例程 1-3　EchoClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class EchoClient {
  private String host="localhost";
  private int port=8000;
  private Socket socket;

  public EchoClient()throws IOException{
     socket=new Socket(host,port);
  }
  public static void main(String args[])throws IOException{
    new EchoClient().talk();
  }
  private PrintWriter getWriter(Socket socket)throws IOException{
    OutputStream socketOut = socket.getOutputStream();
    return new PrintWriter(socketOut,true);
  }
  private BufferedReader getReader(Socket socket)throws IOException{
    InputStream socketIn = socket.getInputStream();
```

```
    return new BufferedReader(new InputStreamReader(socketIn));
  }
  public void talk()throws IOException {
    try{
      BufferedReader br=getReader(socket);
      PrintWriter pw=getWriter(socket);
      BufferedReader localReader=
            new BufferedReader(new InputStreamReader(System.in));
      String msg=null;
      while((msg=localReader.readLine())!=null){

        pw.println(msg);
        System.out.println(br.readLine());

        if(msg.equals("bye"))
          break;
      }
    }catch(IOException e){
       e.printStackTrace();
    }finally{
      try{socket.close();}catch(IOException e){e.printStackTrace();}
    }
  }
}
```

在 EchoClient 类中，最主要的方法为 talk()方法。该方法不断读取用户从控制台输入的字符串，然后把它发送给 EchoServer，再把 EchoServer 返回的字符串打印到控制台。如果用户输入的字符串为“bye”，就会结束与 EchoServer 的通信，调用 socket.close()方法断开连接。

运行范例时，需要打开两个 DOS 界面，先在一个 DOS 界面中运行“java EchoServer”命令，再在另一个 DOS 界面中运行“java EchoClient”命令。图 1-20 显示了运行这两个程序的 DOS 界面。在 EchoClient 控制台，用户输入字符串“hi”，程序就会输出“echo:hi”。

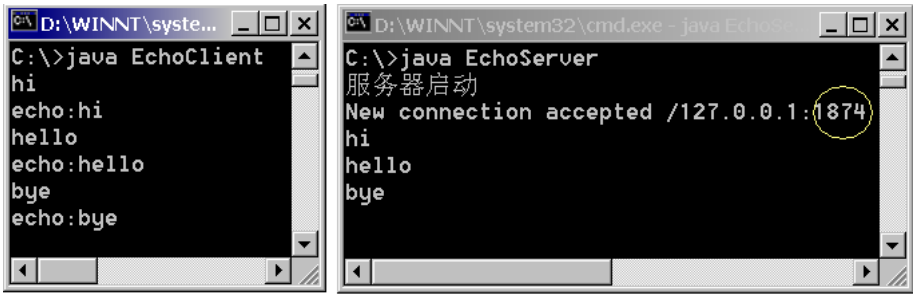

图 1-20　运行 EchoServer 和 EchoClient 程序

如果希望在一个 DOS 控制台中同时运行 EchoServer 和 EchoClient 程序，那么可以先运行“start java EchoServer”命令，再运行“java EchoClient”命令。“start java EchoServer”命令中“start”的作用是打开一个新的 DOS 控制台，然后在该控制台中运行“java EchoServer”命令。

在 EchoServer 程序的 service()方法中，每当 serverSocket.accept()方法返回一个 Socket 对象，就表示建立了与一个客户的连接，这个 Socket 对象中包含了客户的地址和端口信息，只需调用 Socket 对象的.getInetAddress()和 getPort()方法就能分别获得这些信息：

```
socket = serverSocket.accept();  //等待客户连接
System.out.println("New connection accepted "
            +socket.getInetAddress() + ":" +socket.getPort());
```

从图 1-20 可以看 出，EchoServer 的控制台显示 EchoClient 的 IP 地址为 127.0.0.1，端口为 1874。127.0.0.1 是本地主机的 IP 地址，表明 EchoClient 与 EchoServer 在同一个主机上。EchoClient 作为客户程序，它的端口是由操作系统随机产生的。每当客户程序创建一个 Socket 对象，操作系统就会为客户分配一个端口。假定在客户程序中先后创建了两 Socket 对象，这意味着客户与服务器之间同时建立了两个连接：

```
Socket socket1=new Socket(host,port);
Socket socket2=new Socket(host,port);
```

操作系统为客户的每个连接分配一个唯一的端口，参见图 1-21。

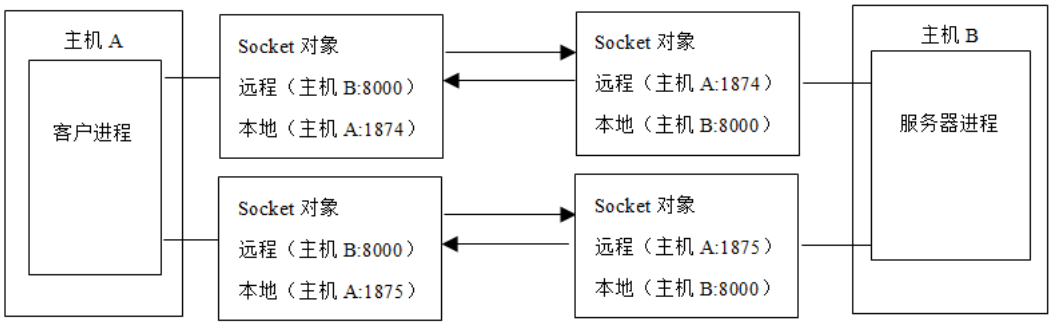

图 1-21　客户与服务器进程之间同时建立了两个连接

在客户进程中，Socket 对象包含了本地以及对方服务器进程的地址和端口信息，在服务器进程中，Socket 对象也包含了本地以及对方客户进程的地址和端口信息，Socket 类的以下方法用于获取这些信息：

- getInetAddress()：获得远程被连接进程的 IP 地址。
- getPort()：获得远程被连接进程的端口。
- getLocalAddress()：获得本地的 IP 地址。
- getLocalPort()：获得本地的端口。

客户进程允许建立多个连接，每个连接都有唯一的端口。在图 1-21 中，客户进程占用两个端口：1874 和 1875。在编写网络程序时，一般只需要显式的为服务器程序中的 ServerSocket 设置端口，而不必考虑客户程序所用的端口。

1.6　小结

简单地理解，计算机网络的任务就是传输数据。为了完成这一复杂的任务，国际标准化组织 ISO 提供了 OSI 参考模型，这种模型把互联网络分为 7 层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。每个层有明确的分工，并且在层与层之间，下层为上层提供服务。这种分层的思想简化了网络系统的设计过程。例如在设计应用层时，只需考虑如何创建满足用户实际需求的应用，在设计传输层时，只需考虑如何在两个主机之间传输数据，在设计网络层时，只需考虑如何在网络上找到一条发送数据的路径，即路由。

由于 OSI 参考模型过于庞大和复杂，使它难以投入到实际运用中。与 OSI 参考模型相似的 TCP/IP 参考模型吸取了网络分层的思想，但是对网络的层次做了简化，并且在网络各层（除了主机-网络层外）都提供了完善的协议，这些协议构成了 TCP/IP 协议集，简称 TCP/IP

协议。TCP/IP 参考模型分为四个层：应用层、传输层、网络互联层和主机-网络层。在每一层都有相应的协议，IP 协议和 TCP 协议是协议集中最核心的两个协议。

IP 协议位于网络互联层，用 IP 地址来标识网络上的各个主机，IP 协议把数据分为若干数据包，然后为这些数据包确定合适的路由。路由就是指把数据包从源主机发送到目标主机的路经。

TCP 协议位于传输层，保证两个进程之间可靠的传输数据。每当两个进程之间进行通信，就会建立一个 TCP 连接，TCP 协议用端口来标识 TCP 连接的两个端点。在传输层还有一个 UDP 协议，它与 TCP 协议的区别是，UDP 不保证可靠的传输数据。

建立在 TCP/IP 协议基础上的网络程序一般都采用客户/服务器通信模式。服务器程序提供服务，客户程序请求获得服务。服务器程序一般昼夜运行，时刻等待客户的请求并及时做出响应。

Java 网络程序致力于实现应用层。传输层向应用层提供了套接字 Socket 接口，Socket 封装了下层的数据传输细节，应用层的程序通过 Socket 来建立与远程主机的 TCP 连接以及进行数据传输。在 Java 中，有三种套接字类：java.net.Socket、java.net.ServerSocket 和 DatagramSocket。其中 Socket 和 ServerSocket 类建立在 TCP 协议基础上；DatagramSocket 类建立在 UDP 协议基础上。

1.7 练习题

1．Java 网络程序位于 TCP/IP 参考模型的哪一层？（单选）

a) 网络互联层

b) 应用层

c) 传输层

d) 主机-网络层

2. 以下哪些协议位于传输层？（多选）

a) TCP　b) HTTP　c) SMTP　d) UDP　e) IP

3. 假定一个进程已经占用 TCP 的 80 端口，它还能否占用 UDP 的 80 端口？（单选）

a) 可以　b) 不可以

4．一个客户进程执行以下代码：

```
Socket socket1=new Socket(host,port);
Socket socket2=new Socket(host,port);
```

以下哪些说法正确？（多选）

a) socket1 与 socket2 占用不同的本地端口。

b) Socket 构造方法中的 port 参数指定客户端占用的本地端口。

c) 当 Socket 构造方法成功返回，就表明建立了与服务器的一个 TCP 连接。

d) 执行第二行程序代码会抛出异常，因为一个客户进程只能与服务器建立一个 TCP 连接。

5．有一种协议规定：如果客户端发送一行字符串“date”，服务器端就返回当前日期信息，如果客户端发送一行字符串“exit”，服务器端就结束与客户端的通信。这种协议应该是哪一层的协议？（单选）

a) 网络互联层　b) 应用层　c) 传输层　d) 主机-网络层

6．HTTP 协议规定默认情况下，HTTP 服务器占用的 TCP 端口号是什么？（单选）

a) 21　　b) 23　　c) 80　　d) 任意一个未被占用的端口号

7．在客户/服务器通信模式中，客户程序与服务器程序的主要任务是什么？（多选）

a) 客户程序在网络上找到一条到达服务器的路由。

b) 客户程序发送请求，并接收服务器的响应。

c) 服务器程序接收并处理客户请求，然后向客户发送响应结果。

d) 客户程序和服务器程序都会保证发送的数据不会在传输途中丢失。

8．从哪里可以找到描述 TCP/IP 协议的具体文档？（单选）

a) JDK 的 JavaDoc 文档

b) NIC 的官方网站

c) 国际标准化组织（ISO）的官方网站

d) RFC 的官方网站

9．一个服务器进程执行以下代码：

```
ServerSocket serverSocket=new ServerSocket(80);
Socket socket=serverSocket.accept();
int port=socket.getPort();
```

以下哪些说法正确？（多选）

a) 服务器进程占用 80 端口。

b) socket.getPort()方法返回服务器进程占用的本地端口号，此处返回值是 80。

c) 当 serverSocket.accept()方法成功返回，就表明服务器进程接收到了一个客户连接请求。

d) socket.getPort()方法返回客户端套接字占用的端口。

答案：(1)a　(2)a,d　(3)a　(4)a,c　(5)b　(6)c　(7)b,c　(8)d　(9)a,c,d