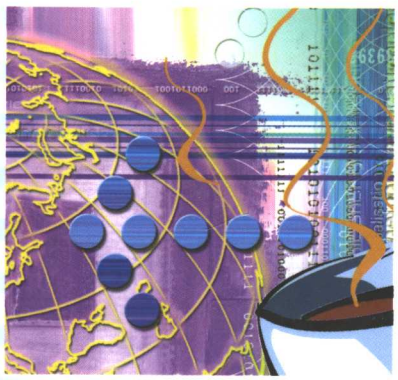
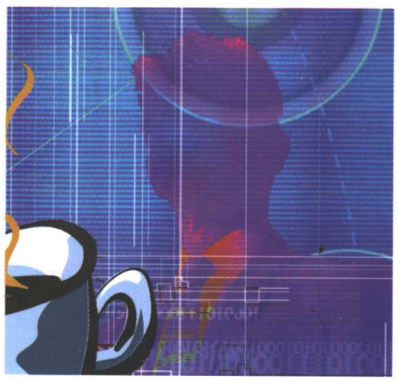
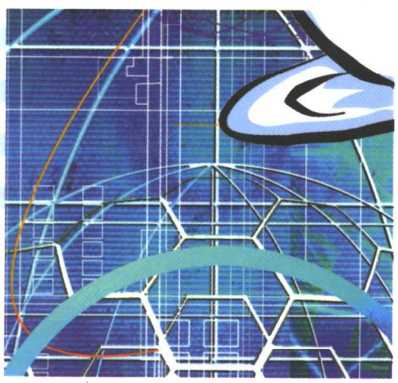
21世纪高等学校应用型教材

Java 网络应用编程

殷兆麟　周智仁　范宝德　编著


高等教育出版社
Higher Education Press

内 容 提 要

本书介绍应用 Java 进行网络编程所用到的技术。全书共 11 章，内容包括：Java 网络编程基础、Java 数据库连接技术 JDBC、JavaBean 组件技术、服务器端小应用程序 Servlet、JSP 技术、Java 与 XML 技术、命名服务与目录服务、Java 远程方法调用 RMI、J2EE 技术基础、Weblogic 开发环境安装与设置以及 J2EE 开发示例。学生在掌握这些知识的基础上，将具有使用 Java 技术开发电子商务、电子政务、企业信息集成等方面应用程序的基本能力。

本书可以作为应用型本科、高职高专计算机应用专业、计算机软件专业、电子商务技术专业、网络信息等专业开设的 Java 网络应用编程 Java 电子商务技术、Java 网络数据库应用技术、Java 网站开发技术、Web 技术等课程的教材。

本书配有 CAI 课件，需要者可以与作者联系：zhlyin@cumt.edu.cn。

图书在版编目（CIP）数据

Java 网络应用编程/殷兆麟，周智仁，范宝德编著.
北京：高等教育出版社，2004.4
ISBN 7-04-014152-3

Ⅰ.J… Ⅱ.①殷…②周…③范… Ⅲ.JAVA 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2004）第 026991 号

策划编辑 雷顺加　　**责任编辑** 萧潇
封面设计 王凌波　　**责任印制** 孔源

出版发行	高等教育出版社	**购书热线**	010-64054588
社　　址	北京市西城区德外大街 4 号	**免费咨询**	800-810-0598
邮政编码	100011	**网　　址**	http://www.hep.edu.cn
总　　机	010-82028899		http://www.hep.com.cn
经　　销	新华书店北京发行所		
印　　刷	河北新华印刷一厂		
开　　本	787×1092　1/16	**版　　次**	2004 年 4 月第 1 版
印　　张	18.25	**印　　次**	2004 年 4 月第 1 次印刷
字　　数	440 000	**定　　价**	25.00 元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

前　言

Java 网络应用编程课程是 Java 技术教学的第二阶段，该课程的先导课程为网页制作、Java 语言程序设计(或 Java 网络编程基础)、数据库应用和计算机网络。本书介绍了 Java Socket、JDBC、JavaBean、Servlet、JSP、XML、JNDI 命名服务、RMI、EJB 技术基础以及 Weblogic 环境下的 EJB 开发。学生在掌握这些知识基础上，具有利用 Java 技术开发电子商务、电子政务、企业 ERP 和信息化集成项目的基本能力。

教材的每章主题相对独立，按其内在联系来组织各章节顺序。本着“理论适度、重在会用”的原则确定编写各章内容的深度、广度。理论叙述尽量图示化。示例既注意简洁、又注意实用、代码的完整。教学实践表明：本书的内容先进、实用，有利于提高学生将程序设计语言、网页制作、数据结构、数据库、网络等众多课程知识综合起来解决问题的能力。

本书对高职高专学校建议讲授 56 学时，实训 36～46 学时。其中 20 学时为配合各章的上机实习。16～26 学时为课程设计。上机实习和课程设计的题目建议以“滚雪球”方式安排。也就是说，以往完成的上机实验可以成为新实验的基础，增加新功能。最后，课程设计可以独立完成一个用不同 Java 技术实现的一个适当简化了的电子商务、电子政务等领域的应用开发项目。

教材的组织编写得到江苏省教委“计算机网络人才培养模式的研究”项目和“百门优秀课程教材基金”的资助，配套的 CAI 课件得到中国矿业大学教改基金资助，系列教材建设列为中国矿业大学新世纪教材工程项目，在此对提供支持的单位表示衷心感谢。

本书由殷兆麟主编，参加编写的还有周智仁和范宝德。此外，在本书的编写过程中，李小斌、李卉、张爱娟、孙晋非、戴磊、孙锦程、谢轶才、王竹晓等作了大量示例选择、程序调试和开发课件的工作，本人深表感谢。

本教材配套的课件包括三个方面：

(1) 利用网页制作的教材电子文档。

(2) 利用 JDK 和 JBuilder9.0 环境开发的教材示例(包括开发步骤)和习题解答。

(3) 自评自测课件。按章节内容组织，既有理论测试、又有程序开发测试。

需要课件的读者可以与作者联系：zhlyin@cumt.edu.cn。

由于时间仓促，作者水平有限，书中难免有不妥和错误之处，恳请广大读者指正。

殷兆麟

2003.10.16 于江苏徐州

目　录

第1章

Java 网络编程基础

本章导读

本章介绍网络进程、网络进程通信、网络进程安全通信、网络编程、网络应用编程等术语，介绍关于 TCP/IP、TCP、UDP、应用层协议等基本概念。重点阐述了利用传输层协议(TCP/UDP)实现网络进程通信的类 Socket。

1.1 Java 网络编程基本概念

1.1.1 网络编程概述

现代计算机的网络编程一般指利用操作系统在不同通信协议层次上提供的接口(系统调用、库函数)实现网络进程安全通信。

网络进程就是网点机(连入网络的计算机)上运行的程序。这里不讲两个网点机之间的通信是因为一个网点机上可能同时运行多个程序，也就是说，多个进程并行运行。所以，两个网点机之间的通信的表述是不确切的。

网络进程通信是一个复杂的问题，它涉及不同网络结构、不同网点计算机硬件、不同计算机操作系统、不同通信方使用的语言、不同网络通信协议等。网络通信协议就是利用软件工程的方法(分层次、分模块等技术)来解决复杂网络进程通信的问题。

现代计算机操作系统，包括 PC 机的 Windows 操作系统，都具有不同程度支持网络通信的能力。实现 TCP/IP 协议族成为现代操作系统功能的一部分。

现代计算机操作系统一般提供网络层、传输层、会话层和应用层的网络通信接口。对于网络应用开发，更多的是利用应用层协议编程。

上述的不同层次网络编程与用户要解决的问题有关，与实现的环境有关，与使用的网络编程语言有关。例如在 CORBA 环境下实现对象远程方法调用、在 Java RMI 环境下实现对象远程方法调用以及在 Microsoft DCOM 环境下实现对象远程方法调用机制，尽管原理是一脉相通的，但是编程技术细节各自不同。

Java网络应用编程要注意两个问题:一是关注Java技术应用于电子商务和电子政务开发,应用于企业ERP开发的理论和方法;二是强调利用成熟的应用层协议(HTTP、Applet、Servlet、JSP和J2EE等)进行开发。

1.1.2 TCP/IP协议族

TCP/IP协议族是一组在Internet上的不同计算机之间进行通信的协议简称,它由TCP(Transport Control Protocol,传输控制协议)、IP(Internet Protocol,网际协议)、UDP(User Datagram Protocol,用户数据报协议)、HTTP(Hypertext Transfer Protocol,超文本传输协议)、FTP(File Transfer Protocol,文件传输协议)和SMTP(Simple Message Transfer Protocol,简单邮件传输协议)等一系列协议组成。TCP/IP从下往上可分为4层结构(如图1-1所示):

(1) 物理层:(以太网、令牌环和X.25等)对应OSI的物理层和数据链路层。

(2) 网络层:(IP协议)对应OSI的网络层。

(3) 传输层:(TCP协议、UDP协议)对应OSI的传输层。

(4) 应用层:(HTTP协议、FTP协议和SMTP协议等)对应OSI的应用层。

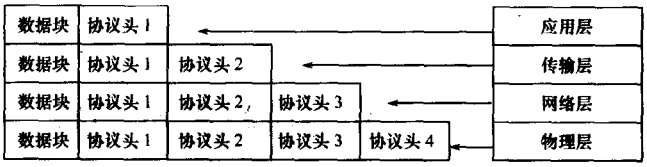


图1-1 TCP/IP分层结构及发送数据的变化图

从一台计算机发往另一台计算机的数据都是按以下顺序发送和接收的:

发送数据的进程先将数据从应用层加上应用层的协议头,由上往下传送;数据经过每一层时,所使用的协议都给数据加上一个协议头,然后将加上协议头的数据传到下一层;下一层所使用的协议再给它加上一个协议头,继续向下传递;最后由物理层经硬件设备发送到网络上。

接收数据的计算机则相反,数据是由下往上传递的,每经过一层时,都剥去相应的协议头,然后继续向上传递;最后传给用户的数据是剥去所有协议头的最原始的数据。

比如说,客户浏览器要从网络上运行的Web服务器上获取数据,则先将一个HTTP请求发送到TCP层,加上一个TCP端口地址之后发往IP层,IP层再给它加上一个IP地址,通过物理层将此请求发往网络上指定的计算机。

在接收数据的一方,每一层都剥去相应的地址信息,然后决定下一步该做什么。物理层在接收到信息后,将数据发往IP层;IP层发现这个包后,将剥去IP头的数据发往TCP层;TCP层将剥去TCP头的数据发送给计算机上运行的HTTP服务代理(即在服务器上运行的服务进程);HTTP代理根据这个请求进行相应的处理,然后将结果按相应的路径返回给请求者。

TCP协议是面向连接的。UDP协议是无连接的。IP协议处在通信子网的最高层。

1.1.3 Socket编程概述

Socket编程指利用网络协议传输层提供的接口作进一步开发,这是所有应用开发的基础。

1. 什么是 Socket

Socket(套接字)是进程之间通信的抽象连接点。要理解这个概念,首先举一个形象的例子。现代生活中,电话是人们联系的常用方法,双方只要使用两部电话机就可以交谈。在计算机世界中,两个程序就是使用 Socket 进行通信的,一个 Socket 就好比一部电话机,两个程序在网络上通过一个双向链路进行通信,这个双向链路的每一端就成为一个 Socket。

2. 什么是 Socket 编程

在 TCP/IP 协议的传输层提供了有关 Socket 的各种调用,目的是在这个协议层上实现网络进程之间的通信。

进程之间要通信,首先要建立各自的 Socket,就像要打电话一定要先有电话机一样。打电话时,每部电话机都可以说和听。与此类似,每个 Socket 也都可以进行读操作和写操作,进行读/写操作时分别要用到 Socket 中的输入流和输出流。

建立连接后,客户程序可以向 Socket 里写入请求,然后服务器会处理这个请求,并把处理结果通过 Socket 送回。服务器应用程序一般监听一个特定端口以等待客户的连接请求,当一个连接请求到达时,客户和服务器建立一个通信连接。在连接过程中,客户被分配一个本地端口号并且与一个 Socket 连接,客户通过写 Socket 来通知服务器,通过读 Socket 来获取信息。类似地,服务器也获取一个本地端口号,它需要一个新的端口号来监听原始端口上的其他连接请求。服务器也给它的本地端口连接一个 Socket,并读写来自客户的请求。

目前可以使用两种 Socket,即流式 Socket 和数据报式 Socket。流式 Socket 提供了双向的、有序的、无重复并且无记录边界的数据流服务。TCP 即是一种流式 Socket 协议;数据报式 Socket 支持双向的数据流,但并不保证可靠、有序、无重复。也就是说,一个以数据报式 Socket 接收信息的进程有可能发现信息重复了,或者和发出的顺序不同。数据报式 Socket 的一个重要特点是它保留了记录边界。UDP 即是一种数据报式 Socket 协议。

Java 的特性之一是提供灵活方便的网络支持,它用专门的类来处理较低层次的 TCP/IP 网络连接。程序员即使从来没有编过网络程序,也能在很短的时间内利用 Java 方便地访问 Internet 上的资源。

3. 端口(Port)

端口是一个逻辑概念。主机上每一个提供服务的程序都运行在该主机的一个对外开放的端口上。端口与使用该端口的服务进程是对应的。程序员可以在创建自己的服务程序时使用自定义的口(除了系统默认的端口)。端口常以整数编号,客户可指定一个端口号,通过这个端口号连接服务进程以接收服务。一些端口号是网络系统使用的,如 80 是 WWW 服务的默认端口号。

1.2 Java 网络 Socket 编程

Socket 是传输层提供的网络进程通信接口。它封装了通信协议族系的不同、同一族系传输层不同协议的差别。通信的主动方是客户,它利用 Socket 机制,向服务器(接收方)发送请求,服务器接收客户请求,返回服务结果。用户可以为 Socket 机制选取不同参数,使 Socket 机制支持不

同族系的通信协议(TCP/IP 或其他)、同族通信协议中不同质量要求的协议(TCP 或 UDP)。

1.2.1 使用 TCP 协议的 Socket 网络编程基础

TCP 是一种可靠的、基于连接的传输层网络协议,是在 Internet 上广泛使用的 TCP/IP 协议族的一员。网络上的两个进程采用 C/S(Client/Server)模式进行通信。当两台主机准备进行交谈时,都必须建立一个 Socket,其中一方作为服务器,打开一个 Socket 并监听来自网络的连接请求,另一方作为客户,向网络上的服务器发送请求,通过 Socket 与服务器传递信息。要建立连接,只需指定主机的 IP 地址和端口号即可。

1. Socket 类

(1) Socket 类的四种构造方法

① Socket(String host, int port) throws UnknownHostException,IOException

创建一个流 Socket(即 Socket 实体对象),并将其连接至特定主机的特定端口上。

参数: host 主机名

port 端口号

② Socket(String host,int port,boolean stream)

构造一个 Socket,并把它连接到特定主机的特定端口上。而此 Socket 是流式 Socket 还是数据报式 Socket,则由最后一个参数 stream 决定。

参数: host 主机名

port 端口号

stream 决定生成的 Socket 是流式 Socket 还是数据报式 Socket

③ Socket(InetAddress address,int port)

构造一个流式 Socket,并把它连接到特定主机的特定端口上。

参数: address 特定的地址

port 端口

④ Socket(InetAddress address,int port,boolean stream)

构造一个流式 Socket,并把它连接到特定 IP 地址的特定端口上。而此 Socket 是流式 Socket 还是数据报式 Socket,则由最后一个参数 stream 决定。

参数: host 主机名

port 端口号

stream 决定生成的 Socket 是流 Socket 还是数据报 Socket

(2) Socket 类提供的主要方法

① InetAddress getInetAddress()

返回该 Socket 所连接的 IP 地址。

② int getPort()

返回该 Socket 所连接的端口。

③ sychronized void close() throws IOException

关闭 Socket。

④ InputStream getInputStream( ) throws IOExcepiton

获得从 Socket 读入数据的输入流。

注:DataInputStream 为 InputStream 的子类。

⑤ InputStream getOutputStream( ) throws IOException

获得向 Socket 进行写操作的输出流。

注:PrintStream 为 OutputStream 的子类。

2. 服务器 Socket 类(ServerSocket)

(1) 服务器 Socket 类有两种构造方法

① ServerSocket(int port) throws IOException

在指定的端口上构造一个服务器 Socket,即构造一个 ServerSocket 实体对象。

参数: port 端口号

② ServerSocket(int port,int count)

构造一个服务器 Socket,即构造一个 ServerSocket 实体对象,并且该对象与指定的当地端口相连接。此外,可以对它进行监听。用户也可以通过将 port 设为 0 来将该对象与一个匿名端口相连接。

参数: port 端口号

count 对该 ServerSocket 实体对象与端口间的连接进行监听的次数。

(2) 服务器 Socket 类提供的主要方法

① Socket accept( ) throws IOException

等待一个连接,该方法将阻塞当前线程,直到连接成功。该方法返回一个 Socket 类对象,通过该对象,程序与连接的客户进行通信。

② void close( ) throws IOException

关闭 Socket。

1.2.2 使用 TCP 协议的 Socket 网络编程实现

1. TCP 协议通信的服务器端编程步骤

服务器端的编程步骤如下:

(1) 以某端口号为参数调用 ServerSocket 类的构造函数,创建一个 ServerSocket 对象,服务器端程序将在这个端口上监听、等待客户程序发来的请求。假设服务器工作在端口 8000 上,以下命令建立一个服务器 Socket,它监视端口 8000:

```
ServerSocket svrsoc = new ServerSocket(8000);
```

(2) 服务程序使用 ServerSocket 对象的方法 accept( ),等待接收某客户端程序发出的连接请求。服务器方会一直阻塞,直到有客户连接到该端口。一旦有客户发送正确请求,连接至该端口,accept( )方法就返回一个 Socket 对象,该对象代表和客户端建立的通信链路在服务程序内的通信端点,如下所示:

```
Socket soc = svrsoc.accept();
```

通过 Socket 对象可以获得客户端的相关信息，如客户机的 IP 地址：

```
clientIP = soc.getInetAddress();
```

可以利用 Socket 类提供的方法 getInputStream()和 getOutputStream()来创建输入/输出流。

```
InputStream is = soc. getInputStream();
OutputStream os = soc. getOutputStream();
```

(3) 无论是服务器端还是客户端，程序对 Socket 的输入/输出流进行读/写操作都和对普通的输入/输出流进行读/写基本一样。但是，为了便于读/写，需要在这两个流对象的基础上建立易于操作的数据流 DataInputStream、DataOutputStream 或 PrintStream，采用如下语句即可实现：

```
DataInputStream     in = new DataInputStream(is);
DataOutputStream    out = new DataOutputStream(os);
PrintStream         out = new PrintStream(Os);
```

几个常用的读/写方法如下：

① 读入一行文本：String DataInputStream.readline();

② 读入一个字节：Byte DataInputStream.readByte();

③ 写一行文本：void DataOutputStream.writeBytes(String);

④ 写一个字节：void DataOutputStream.writeByte(int);

⑤ 写一个 type 型数据：void PrintStream.writeln(type);

(4) 在退出前要关闭输入/输出流及 Socket，以此断开连接并且释放所有占用的资源。需要注意的是，要先将对应的输入/输出流关闭，然后再关闭 Socket 本身。

2. TCP 协议通信的客户端编程步骤

客户端的实现与服务器端的实现的主要区别在建立连接这一步。客户端首先创建一个指向一个服务器固定端口的 Socket。假如下述服务者在本机“localhost”、端口 8000，则以下命令：

```
Socket soc = new Socket("localhost",8000);
```

就建立了客户到服务器的连接，两端进行通信的通路即建立。

接下来，获得输入/输出流，读/写数据是类似的。

最后，关闭输入/输出流及 Socket，和服务器端的工作过程是类似的。

3. TCP 协议通信的编程示例

当两个进程准备利用 TCP 通信时，都必须建立各自的 Socket，其中一方作为客户，它主动向网络上的指定主机 IP 地址和端口号的服务器发送请求，建立连接；另一方作为服务器，它打开一个 Socket 并监听来自网络的连接请求。

例 1-1 本例要求实现客户端可以接收看到服务器端传来的问候语“Welcome!”，然后在客户端通过键盘输入字符，传送给服务器端，服务器端接受并显示出来，直到客户在客户端输入“quit”，双方结束通信。客户方与服务器方在同一主机上。图 1-2 描述了它们的通信流程图。服务器建立 ServerSocket 时，主要做的事(do)是构造 ServerSocket 对象，监听客户的请求消息。如到达一个客户的请求消息(exit 条件)，服务器进入接收消息(accept())，同时返回一个与原客户

Socketsoc 连接的新服务方 Socketsoc。

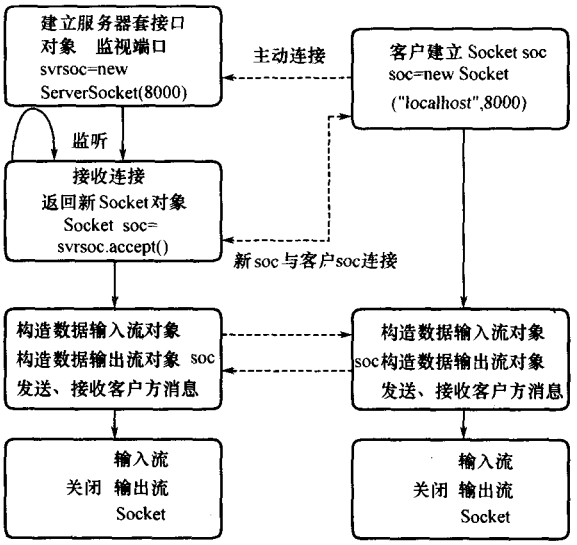


图 1-2 例 1-1 服务方、客户间的通信流程图

图 1-3 是服务器端不完全规范的 UML 模型简图。它描述了类 TcpServer 继承了 Object 类(可默认)以及 TcpServer 依赖的类,如:DataInputStream、InputStream、IOException、OutputStream、PrintStream 等。

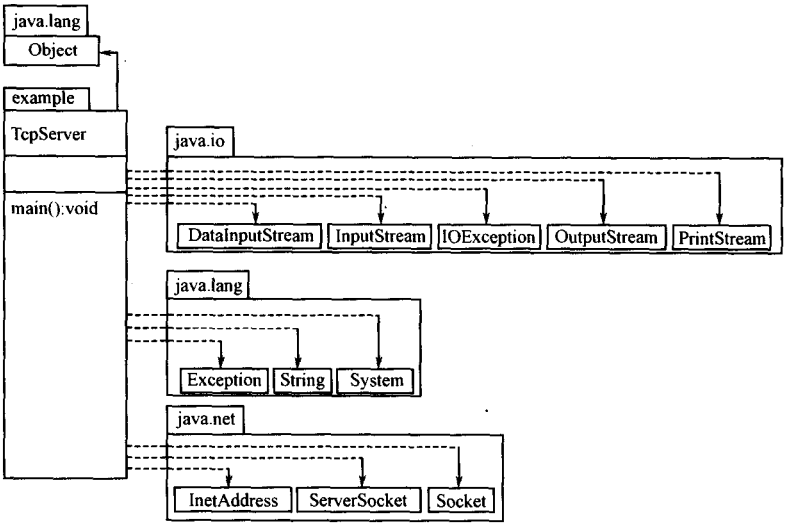


图 1-3 服务器端 UML 模型简图

(1) 服务器端源程序

```
package example;
import java.io.*;
```

```
import java.net.*;
public class TcpServer
{
static public void main(String args[]) throws IOException{
  ServerSocket svrsoc = null;
  Socket soc = null;
  InputStream is = null;
  OutputStream os = null;
  DataInputStream in = null;
  PrintStream out = null;
  try{
    svrsoc = new ServerSocket(8000);        //构造 serverSockets 对象,端口为 8000
    soc = svrsoc.accept();                  //服务器端等待一个连接,返回新 Socketsoc
    is = soc.getInputStream();              //创建输入流
    in = new DataInputStream(is);
    os = soc.getOutputStream();             //创建输出流
    out = new PrintStream(os);
    InetAddress clientIP = soc.getInetAddress();      //得到客户方的 IP 地址
    System.out.println( " Client's IP address: " + clientIP);
    int port;
    port = soc.getPort();                             //得到客户方的端口
    System.out.println( " Client's port: " + port);
    out.println( " Welcome! … " );
    String str = in.readLine();                       //在 in 上读一行
    while(!str.equals( " quit " )){                   //如读出的不是"quit",继续读
        System.out.println( " Client said: " + str);
        str = in.readLine();
    }
    System.out.println( " Client want to leave. " );
  }
  catch(Exception e) {
    System.out.println( " Error: " + e);
  }
  finally{
    is.close();                                       //关闭输入流
    os.close();                                       //关闭输出流
    soc.close();                                      //关闭 socket
    System.exit(0);
  }
 }
}
```

(2) 客户端源程序

图 1-4 描述了客户方不完全规范的 UML 模型简图。它描述了类 TcpServer 继承了 Object 类(可默认)以及 TcpServer 依赖的类。

代码如下:

```
package example;
```

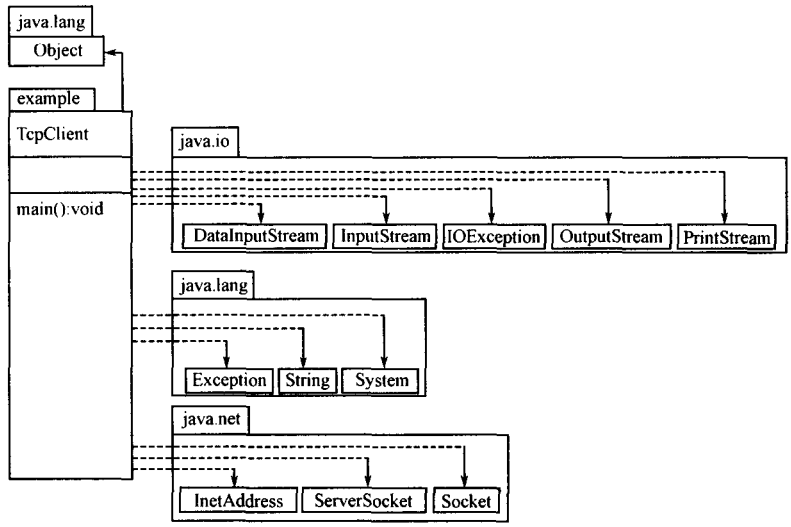


图 1-4 客户端 TcpClient UML 模型简图

```
import java.net.*;
import java.io.*;
public class TcpClient{
static public void main(String args[]) throws IOException{
  Socket soc=null;
  InputStream is=null;
  OutputStream os=null;
  DataInputStream in=null;
  PrintStream out=null;
  String strin=null;
  try{
    soc=new Socket("localhost",8000);
    System.out.println("Connecting to the Server…");
    is=soc.getInputStream();
    os=soc.getOutputStream();
    in=new DataInputStream(is);
    out=new PrintStream(os);
    strin=in.readLine();
    System.out.println("Server said:"+strin);
    byte bmsg[]=new byte[20];
    System.in.read(bmsg);
    String msg=new String(bmsg,0);
    msg=msg.trim();
    while(!msg.equals("quit")){
      out.println(msg);
      System.in.read(bmsg);
      msg=new String(bmsg,0);
      msg=msg.trim();
    }
```

```
            out.println(msg);
        }
        catch (Exception e) {
            System.out.println( " Error: " + e);
        }
        finally{
            is.close();
            os.close();
            soc.close();
            System.exit(0);
        }
    }
}
```

当运行服务器端程序时,开始没有任何显示,运行客户端程序后,将会显示客户端的IP地址和端口号。当客户端输入字符串后,服务器端的运行结果会显示相应的字符串。当客户端输入的字符串为“quit”时,服务器端显示“Client want to leave.”。

服务器端运行结果如图1-5所示。

客户端运行后,首先显示服务器端发送过来的问候语“Welcome!”,接着,可以任意输入字符串。此时,服务器端会有相应的输出,直到输入“quit”使客户端和服务器端程序都结束。

客户端运行结果如图1-6所示。

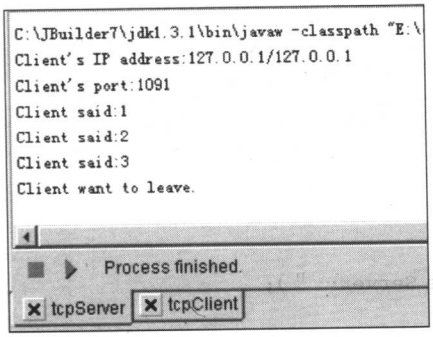


图1-5 服务器端运行结果

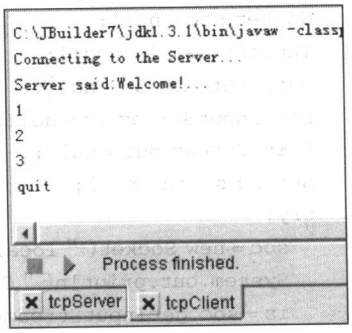


图1-6 客户端运行结果

本章小结

通信协议是通信双方的约定。通信协议是分层次的。TCP/IP协议是Internet网络层、传输层、应用层一系列通信协议的习惯称呼。

Socket(套接字)是进程之间通信的抽象连接点。Socket按使用的协议不同,分为面向连接的Socket和不连接的Socket。Socket的端口是定位主机上通信的网络进程。使用Socket通信的主动方叫客户,被动方叫服务器。客户和服务器利用Socket通信,双方的作用各不一样。通信前,它们首先要建立各自的Socket。客户程序可以向Socket里写入请求,然后服务器会处理这个请求,

并把处理结果通过 Socket 送回。

习 题 一

1-1 Socket 作用是什么？端口有什么意义？列举 Socket 类的特征和方法。

1-2 客户和服务者通过 Socket 通信，描述基于有连接和无连接通信时的流程图。

1-3 对照图 1-3(服务方 UML 模型简图)阅读服务器端实现的源程序，说明该 UML 模型中类在程序中的作用。对照图 1-4(客户方 UML 模型简图)阅读客户端实现的源程序，说明该 UML 模型中类在程序中的作用。

1-4 利用 Java 网络包 java.io 中 Socket 类和 ServerSocket 类分别实现有连接和无连接的通信。

1-5 利用 Java 网络包 java.io 中 Socket 类和 ServerSocket 类分别实现无连接的通信。

1-6* 聊天室有 n 个人在聊天，聊天服务器就应该为每个人连入的聊天 Applet 创建一个对应的线程，该线程监听对应的 Applet 是否有消息传来。如果有侧响所有的聊天 Applet 广播该消息。实现该网络聊天系统。

(* 表示有难度。参考程序见本书课件。)