第16章

网络通信

（视频讲解：33分钟）

Internet提供了大量、多样的信息，很少有人接触过Internet后能拒绝它的诱惑。计算机网络实现了多个计算机互联系统，相互连接的计算机之间彼此能够进行数据交流。网络应用程序就是在已连接的不同计算机上运行的程序，这些程序相互之间可以交换数据。而编写网络应用程序，首先必须明确网络应用程序所要使用的网络协议，TCP/IP协议是网络应用程序的首选。本章从介绍网络协议开始，向读者介绍TCP网络程序和UDP网络程序。

通过阅读本章，您可以：

- 了解常见的几种网络协议
- 理解端口和套接字
- 掌握InetAddress类的应用
- 掌握ServerSocket类的应用
- 学习编写TCP网络程序
- 掌握DatagramPacket类的应用
- 掌握DatagramSocket类的应用
- 学习编写UDP程序

16.1 网络程序设计基础

视频讲解：光盘\TM\lx\16\网络程序设计基础.exe

网络程序设计是指编写与其他计算机进行通信的程序。Java 已经将网络程序所需要的东西封装成不同的类。只要创建这些类的对象，使用相应的方法，即使设计人员不具备有关的网络知识，也可以编写出高质量的网络通信程序。

16.1.1 局域网与因特网

为了实现两台计算机的通信，必须要用一个网络线路连接两台计算机，如图 16.1 所示。

图 16.1 服务器、客户机和网络

服务器是指提供信息的计算机或程序，客户机是指请求信息的计算机或程序，而网络用于连接服务器与客户机，实现两者相互通信。但有时在某个网络中很难将服务器与客户机区分开。通常所说的“局域网”（Local Area Network，LAN），就是一群通过一定形式连接起来的计算机。它可以由两台计算机组成，也可以由同一区域内的上千台计算机组成。由 LAN 延伸到更大的范围，这样的网络称为“广域网”（Wide Area Network，WAN）。人们熟悉的因特网（Internet），则是由无数的 LAN 和 WAN 组成。

16.1.2 网络协议

网络协议规定了计算机之间连接的物理、机械（网线与网卡的连接规定）、电气（有效的电平范围）等特征以及计算机之间的相互寻址规则、数据发送冲突的解决、长的数据如何分段传送与接收等。就像不同的国家有不同的法律一样，目前网络协议也有多种，下面简单地介绍几个常用的网络协议。

1. IP 协议

IP 其实是 Internet Protocol 的简称，由此可知它就是一种“网络协议”。Internet 网络采用的协议是 TCP/IP 协议，其全称是 Transmission Control Protocol/Internet Protocol。Internet 依靠 TCP/IP 协议，在全球范围内实现不同硬件结构、不同操作系统、不同网络系统的互联。在 Internet 网上存在数以亿计的主机，每一台主机在网络上通过为其分配的 Internet 地址表示自己，这个地址就是 IP 地址。到目前为止 IP 地址用 4 个字节，也就是 32 位的二进制数来表示，称为 IPv4。为了便于使用，通常取用每个字节的十进制数，并且每个字节之间用圆点隔开来表示 IP 地址，如 192.168.1.1。现在人们正在试验使用 16 个字节来表示 IP 地址，这就是 IPv6，但 IPv6 还没有投入使用。

TCP/IP 模式是一种层次结构，共分为 4 层，分别为应用层、传输层、互联网层和主机到网络层。

各层实现特定的功能，提供特定的服务和访问接口，并具有相对的独立性，如图 16.2 所示。

2．TCP 与 UDP 协议

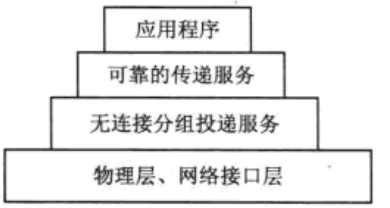

图 16.2　TCP/IP 层次结构

在 TCP/IP 协议栈中，有两个高级协议是网络应用程序编写者应该了解的，即“传输控制协议”（Transmission Control Protocol，TCP）与“用户数据报协议”（User Datagram Protocol，UDP）。

TCP 协议是一种以固接连线为基础的协议，它提供两台计算机间可靠的数据传送。TCP 可以保证数据从一端送至连接的另一端时能够确实送达，而且抵达的数据的排列顺序和送出时的顺序相同，因此 TCP 协议适合可靠性要求比较高的场合。就像拨打电话一样，必须先拨号给对方，等两端确定连接后，相互能听到对方的说话，也知道对方回应的是什么。

HTTP、FTP 和 Telnet 等都需要使用可靠的通信频道，例如 HTTP 从某个 URL 读取数据时，如果收到的数据顺序与发送时不相同，可能就会出现一个混乱的 HTML 文件或是一些无效的信息。

UDP 是无连接通信协议，不保证可靠数据的传输，但能够向若干个目标发送数据，接收发自若干个源的数据。UDP 是以独立发送数据包的方式进行。这种方式就像邮递员送信给收信人，可以寄出很多信给同一个人，而每一封信都是相对独立的，各封信送达的顺序并非重要，而收信人接收信件的顺序也不能保证与寄出信件的顺序相同。

UDP 协议适合于一些对数据准确性要求不高的场合，如网络聊天室、在线影片等。由于 TCP 协议在认证上存在额外耗费，有可能使传输速度减慢；UDP 协议可能会更适合这些对传输速度和时效要求非常高的网站，即使有一小部分数据包的遗失或传送顺序有所不同，并不会严重危害该项通信。

注意　一些防火墙和路由器会设置成不允许 UDP 数据包传输，因此若遇到 UDP 连接方面的问题，应先确定是否允许 UDP 协议。

16.1.3　端口和套接字

一般而言，一台计算机只有单一的连到网络的“物理连接”（Physical Connection），所有的数据都通过此连接对内、对外送达特定的计算机，这就是端口。网络程序设计中的端口（port）并非真实的物理存在，而是一个假想的连接装置。端口被规定为一个在 0～65535 之间的整数。HTTP 服务一般使用 80 端口，FTP 服务使用 21 端口。假如一台计算机提供了 HTTP、FTP 等多种服务，那么客户机通过不同的端口来确定连接到服务器的哪项服务上，如图 16.3 所示。

说明　通常 0～1023 之间的端口数用于一些知名的网络服务和应用，用户的普通网络应用程序应该使用 1024 以上的端口数，以避免端口号与另一个应用或系统服务所用端口冲突。

网络程序中套接字（Socket）用于将应用程序与端口连接起来。套接字是一个假想的连接装置，就

像插插头的设备“插座”，用于连接电器与电线，如图 16.4 所示。Java 将套接字抽象化为类，程序设计者只需创建 Socket 类对象，即可使用套接字。

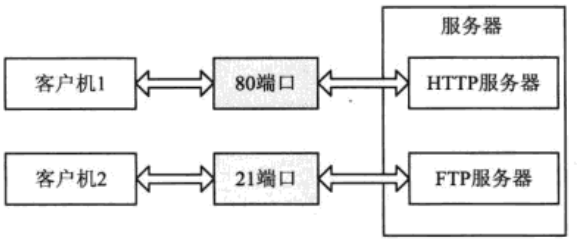

图 16.3　端口

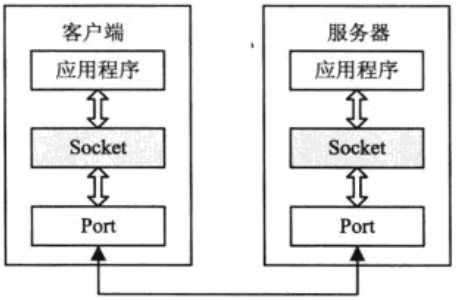

图 16.4　套接字

16.1.4　范例 1：获得内网的所有 IP 地址

在进行网络编程时，有时需要对局域网内的所有主机进行遍历，为此需要获得内网的所有 IP 地址。本范例将演示如何在 Java 应用程序中获得内网的所有 IP 地址。运行程序，单击“显示所有 IP”按钮，将在文本域中显示内网中所有主机的 IP 地址。运行结果如图 16.5 所示。（**实例位置：光盘\TM\sl\16\1**）

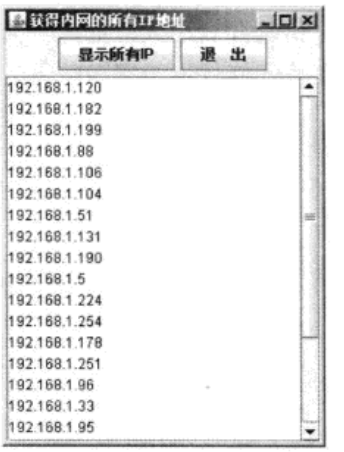

图 16.5　内网的所有 IP 地址

（1）在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 GainAllIpFrame。

（2）在 GainAllIpFrame 窗体的内容面板顶部位置放一个面板控件，并在该面板控件上放两个按钮，然后在内容面板的中部位置放一个滚动面板控件，并在滚动面板上放一个文本域控件，完成窗体界面的设置。

（3）在 GainAllIpFrame 窗体类中定义一个 gainAllIp()方法，用于获得内容中的所有 IP 地址，并追加到文本域中显示。

（4）在 GainAllIpFrame 窗体类中定义一个 PingIpThread 线程类，用于 Ping 指定的 IP 地址，并判断其是否为有效的内网 IP 地址，如果是就添加到集合对象中。代码如下：

```
public String ip;                                          //表示 IP 地址的成员变量
public PingIpThread(String ip) {                           //参数为需要判断的 IP 地址
    this.ip = ip;
}
public void run() {
    try {
        //获得所 Ping 的 IP 进程，-w 280 是等待每次回复的超时时间，-n 1 是要发送的回显请求数
        Process process = Runtime.getRuntime().exec(
                "ping " + ip + " -w 280 -n 1");
```

```
            InputStream is = process.getInputStream();                    //获得进程的输入流对象
            InputStreamReader isr = new InputStreamReader(is);            //创建 InputStreamReader 对象
            BufferedReader in = new BufferedReader(isr);                  //创建缓冲字符流对象
            String line = in.readLine();                                  //读取信息
            while (line != null) {
                if (line != null && !line.equals("")) {
                    if (line.substring(0, 2).equals("来自")
                            || (line.length() > 10 && line.substring(0, 10)
                                    .equals("Reply from"))) {             //判断是 Ping 通过的 IP 地址
                        pingMap.put(ip, "true");                          //向集合中添加 IP
                    }
                }
                line = in.readLine();                                     //再读取信息
            }
        } catch (IOException e) {
        }
    }
```

16.1.5　范例 2：解析网页中的内容

本范例演示了如何在 Java 应用程序中解析网页的内容。运行程序，在输入网址标签右侧的文本框中输入网址，单击“解析网页”按钮，将在文本域中显示解析的网页内容。运行结果如图 16.6 所示。**（实例位置：光盘\TM\sl\16\2）**

图 16.6　解析网页中的内容

（1）在 Eclipse 中新建项目 02，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承 JFrame 类的 InternetContentFrame 窗体类。

（3）在 InternetContentFrame 窗体类中创建用于解析网页内容的 getURLConnection()方法，代码如下：

```
public Collection<String> getURLConnecnnon(String urlString){
    URL url = null;                                          //声明 URL
    URLConnection conn = null;                               //声明 URLConnection
    Collection<String> urlCollection = new ArrayList<String>();  //创建集合对象
    try{
        url = new URL(urlString);                            //创建 URL 对象
        conn = url.openConnection();                         //获得连接对象
        conn.connect();                                      //打开到 URL 引用资源的通信链接
        InputStream is = conn.getInputStream();              //获取流对象
        //转换为字符流
        InputStreamReader in = new InputStreamReader(is,"UTF-8");
        BufferedReader br = new BufferedReader(in);          //创建缓冲流对象
        String nextLine = br.readLine();                     //读取信息，解析网页
        while (nextLine !=null){
            urlCollection.add(nextLine);                     //解析网页的全部内容，添加到集合中
            nextLine = br.readLine();                        //读取信息，解析网页
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }
    return urlCollection;
}
```

（4）在 InternetContentFrame 窗体类中为“解析网页”按钮添加事件代码，实现调用 getURLConnection()方法解析网页内容并显示在文本域中。代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        String address = tf_address.getText().trim();        //获得输入的网址
        Collection urlCollection = getURLCollection(address);//调用方法，获得网页内容的集合对象
        Iterator it = urlCollection.iterator();              //获得集合的迭代器对象
        while(it.hasNext()){
            ta_content.append((String)it.next()+"\n");       //在文本域中显示解析的内容
        }
    }
});
```

16.2 TCP 程序设计基础

视频讲解：光盘\TM\lx\16\TCP 程序设计基础.exe

TCP 网络程序设计是指利用 Scoket 类编写通信程序。利用 TCP 协议进行通信的两个应用程序是有主次之分的，一个称为服务器程序，另一个称为客户机程序。两者的功能和编写方法大不一样。服务

器端与客户端的交互过程如图 16.7 所示。

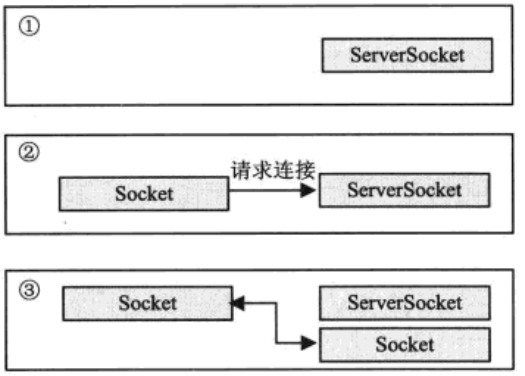

图 16.7　服务器端与客户端的交互

①——服务器程序创建一个 ServerSocket（服务器端套接字），调用 accept()方法等待客户机来连接。

②——客户端程序创建一个 Socket，请求与服务器建立连接。

③——服务器接收客户机的连接请求，同时创建一个新的 Socket 与客户建立连接。服务器继续等待新的请求。

16.2.1　InetAddress 类

java.net 包中的 InetAddress 类是与 IP 地址相关的类，利用该类可以获取 IP 地址、主机地址等信息。InetAddress 类的常用方法如表 16.1 所示。

表 16.1　InetAddress 类的常用方法

返　回　值	方　　法	说　　明
InetAddress	getByName(String host)	获取与 host 相对应的 InetAddress 对象
String	getHostAddress()	获取 InetAddress 对象所含的 IP 地址
String	getHostName()	获取此 IP 地址的主机名
InetAddress	getLocalHost()	返回本地主机的 InetAddress 对象

说明　这里只给出了 InetAddress 类的常用方法，该类还提供了许多其他方法，如果需要其他方法，可以查询 JDK 的 API 文档。

【例 16.1】　使用 InetAddress 类的 getHostName()和 getHostAddress()方法获得本地主机名、本机 IP 地址。（实例位置：光盘\TM\sl\16\3）

```
import java.net.InetAddress;
import java.net.UnknownHostException;
public class Address {                                          //创建类
    public static void main(String[] args) {
```

```
        InetAddress ip;                                          //声明 InetAddress 对象
        try {                                                    //try 语句块捕捉可能出现的异常
            ip = InetAddress.getLocalHost();                     //实例化 InetAddress 对象
            String localname = ip.getHostName();                 //获取本地主机名
            String localip = ip.getHostAddress();                //获取本地主机的 IP 地址
            System.out.println("本地主机名：" + localname);        //输出本地主机名
            System.out.println("本机 IP 地址：" + localip);        //输出本地主机的 IP 地址
        } catch (UnknownHostException e) {
            e.printStackTrace();                                 //输出异常信息
        }
    }
}
```

运行结果如图 16.8 所示。

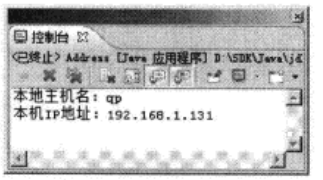

图 16.8　获得本地主机名和 IP 地址

注意 InetAddress 类的方法会抛出 UnknownHostException 异常，所以必须进行异常处理。这个异常在主机不存在或网络连接错误时发生。

16.2.2　ServerSocket 类

java.net 包中的 ServerSocket 类用于表示服务器套接字，其主要功能是等待来自网络上的“请求”，它可通过指定的端口来等待连接的套接字。服务器套接字一次可以与一个套接字连接。如果多台客户机同时提出连接请求，服务器套接字会将请求连接的客户机存入队列中，然后从中取出一个套接字，与服务器新建的套接字连接起来。若请求连接数大于最大容纳数，则多出的连接请求被拒绝。队列的默认大小为 50。

ServerSocket 类的构造方法都抛出 IOException 异常，分别有以下几种形式。

- ☑ ServerSocket()：创建非绑定服务器套接字。
- ☑ ServerSocket(int port)：创建绑定到特定端口的服务器套接字。
- ☑ ServerSocket(int port, int backlog)：利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号。
- ☑ ServerSocket(int port, int backlog, InetAddress bindAddress)：使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器。这种情况适用于计算机上有多块网卡和多个 IP 的情况，可以明确规定 ServerSocket 在哪块网卡或 IP 地址上等待客户的连接请求。

ServerSocket 类的常用方法如表 16.2 所示。

表 16.2　ServerSocket 类的常用方法

返 回 值	方　法	说　明
Socket	accept()	等待客户机的连接。若连接，则创建一套接字
boolean	isBound()	判断 ServerSocket 的绑定状态
InetAddress	getInetAddress()	返回此服务器套接字的本地地址
boolean	isClosed()	返回服务器套接字的关闭状态
void	close()	关闭服务器套接字
void	bind(SocketAddress endpoint)	将 ServerSocket 绑定到特定地址（IP 地址和端口号）
int	getInetAddress()	返回服务器套接字等待的端口号

说明　这里只给出了 ServerSocket 类的常用方法，该类还提供了许多其他方法，在使用时可以查询 JDK 的 API 文档。

调用 ServerSocket 类的 accept()方法，将返回一个与客户端 Socket 对象相连接的 Socket 对象，服务器端的 Socket 对象使用 getOutputStream()方法获得的输出流对象，将指向客户端 Socket 对象使用 getInputStream()方法获得的输入流对象；同样，服务器端的 Socket 对象使用 getInputStream()方法获得的输入流对象，将指向客户端 Socket 对象使用 getOutputStream()方法获得的输出流对象。也就是说当服务器向输出流写入信息时，客户端通过相应的输入流就能读取。反之亦然。

注意　accept()方法会阻塞线程的继续执行，直到接收到客户的呼叫。如果没有客户呼叫服务器，那么下面的代码“System.out.println("连接中")”将不会执行。如果没有客户请求的情况下 accept()方法没有发生阻塞，则一定是程序出现了问题，这种问题通常是使用了一个还在被其他程序占用的端口号，ServerSocket 绑定没有成功。

```
yu = server.accept();
System.out.println("连接中");
```

16.2.3　TCP 网络程序

知道了 TCP 程序的工作过程，就可以编写 TCP 服务器程序了。在网络编程中如果只要求客户机向服务器发送消息，不用服务器向客户机发送消息，称为单向通信。客户机套接字和服务器套接字连接成功后，客户机通过输出流发送数据，而服务器会使用输入流接收数据。下面是简单的单向通信的实例。

【例 16.2】　本实例是一个 TCP 服务器端程序，在 getserver()方法中建立服务器套接字，调用 getClientMessage()方法获取客户端信息。（**实例位置：光盘\TM\sl\16\4**）

```
public class MyTcp {                                          //创建 MyTcp 类
    private BufferedReader reader;                            //创建 BufferedReader 对象
    private ServerSocket server;                              //创建 ServerSocket 对象
```

```
    private Socket socket;                                          //创建 Socket 对象 socket
    void getserver() {
        try {
            server = new ServerSocket(8998);                        //实例化 Socket 对象
            System.out.println("服务器套接字已经创建成功");              //输出信息
            while (true) {                                          //如果套接字是连接状态
                System.out.println("等待客户机的连接");                 //输出信息
                socket = server.accept();                           //实例化 Socket 对象
                reader = new BufferedReader(new InputStreamReader(socket
                        .getInputStream()));                        //实例化 BufferedReader 对象
                getClientMessage();                                 //调用 getClientMessage()方法
            }
        } catch (Exception e) {
            e.printStackTrace();                                    //输出异常信息
        }
    }
    private void getClientMessage() {
        try {
            while (true) {                                          //如果套接字是连接状态
                System.out.println("客户机：" + reader.readLine());   //获得客户端信息
            }
        } catch (Exception e) {
            e.printStackTrace();                                    //输出异常信息
        }
        try {
            if (reader != null) {
                reader.close();                                     //关闭流
            }
            if (socket != null) {
                socket.close();                                     //关闭套接字
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {                       //主方法
        MyTcp tcp = new MyTcp();                                    //创建本类对象
        tcp.getserver();                                            //调用方法
    }
}
```

运行结果如图 16.9 所示。

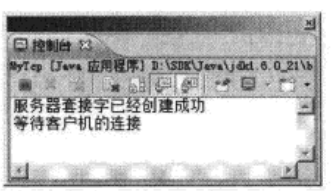

图 16.9　服务器端运行结果

注意 本实例是服务器端程序，当本实例运行后，如果在服务器没有停止运行的情况下再次运行本实例将发生异常，这是由于服务器程序使用的端口号已经被占用，所以在服务器程序没有停止的情况下再次运行就发生了异常。

从图 16.9 中可以看出，服务器端程序运行后，将输出提示信息，并等待客户机的连接。下面再来看一下客户端程序。

【例 16.3】 客户端程序，实现将用户在文本框中输入的信息发送至服务器端，并将文本框中输入的信息显示在客户端的文本域中。（**实例位置：光盘\TM\sl\16\5**）

```
public class MyClien extends JFrame {                                    //创建类继承 JFrame 类
    private PrintWriter writer;                                          //声明 PrintWriter 类对象
    Socket socket;                                                       //声明 Socket 对象
    private JTextArea ta = new JTextArea();                              //创建 JTextArea 对象
    private JTextField tf = new JTextField();                            //创建 JTextField 对象
    Container cc;                                                        //声明 Container 对象
    public MyClien(String title) {                                       //构造方法
        super(title);                                                    //调用父类的构造方法
        cc = this.getContentPane();                                      //实例化对象
        cc.add(ta, "North");                                             //将文本域放在窗体的上部
        cc.add(tf, "South");                                             //将文本框放在窗体的下部
        tf.addActionListener(new ActionListener() {                      //绑定事件
                    public void actionPerformed(ActionEvent e) {
                        writer.println(tf.getText());                    //将文本框中信息写入流
                        ta.append(tf.getText() + '\n');                  //将文本框中信息显示在文本域中
                        tf.setText(" ");                                 //将文本框清空
                    }
                });
    }
    private void connect() {                                             //连接套接字方法
        ta.append("尝试连接\n");                                          //文本域中提示信息
        try {                                                            //捕捉异常
            socket = new Socket("192.168.1.131", 8998);                  //实例化 Socket 对象
            writer = new PrintWriter(socket.getOutputStream(), true);    //获得输出流对象
            ta.append("完成连接\n");                                      //文本域中提示信息
        } catch (Exception e) {
            e.printStackTrace();                                         //输出异常信息
        }
    }
    public static void main(String[] args) {                             //主方法
        MyClien clien = new MyClien("向服务器送数据");                    //创建本例对象
        clien.setBounds(300,260,340,220);                                //设置窗体位置和大小
        clien.setVisible(true);                                          //将窗体显示
        clien.connect();                                                 //调用连接方法
    }
}
```

运行结果如图 16.10 所示。

注意 为了使本实例能够正常运行，必须先运行例 16.3 的服务器端程序，并且实例的端口号要与例 16.3 中服务器套接字的端口号一致，否则程序会发生错误。

说明 当一台机器上安装了多个网络应用程序时，很可能指定的端口号已被占用。有时可能还会遇到以前很好的网络程序突然运行不起来了，这种情况很可能也是由于端口被其他程序占用了。此时可以运行 netstat -help 来获得帮助，使用命令 netstat -an 来查看该程序所使用的端口，如图 16.11 所示。

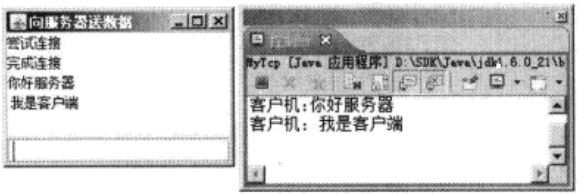

图 16.10 客户端的运行结果

图 16.11 在命令行查看端口

16.2.4 范例 3：设置等待连接的超时时间

在进行网络编程时，由于进行网络连接是比较消耗资源的，因此，可以对连接的等待时间进行设置，如果在规定的时间没有进行连接，则进行其他处理。运行程序，等待 10 秒钟后，将弹出消息框提示连接超时。运行结果如图 16.12 所示。（**实例位置：光盘\TM\sl\16\6**）

（1）在 Eclipse 中新建项目 06，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ConnectionTimeoutSetFrame。

（3）在 ConnectionTimeoutSetFrame 窗体的内容面板上添加一个 JScrollPane 滚动面板控件，然后在滚动面板的视图上放置一个 JTextArea 文本域控件，文本域控件的名称为 ta_info，用于显示服务器等待客户端连接的相关信息，并显示等待连接超时的信息。

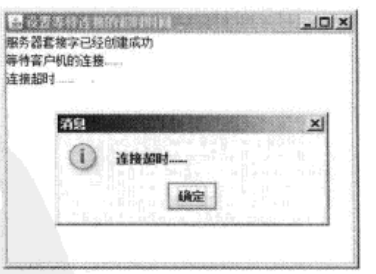

图 16.12 设置等待连接的超时时间

（4）实现创建服务器套接字和设置等待连接超时时间的代码定义在 getServer()方法中，代码如下：

```
public void getServer() {
    try {
        server = new ServerSocket(1978);                    //实例化 Socket 对象
```

```
            server.setSoTimeout(10000);                                    //设置连接超时时间为 10 秒
            ta_info.append("服务器套接字已经创建成功\n");                    //输出信息
            while (true) {                                                 //如果套接字是连接状态
                ta_info.append("等待客户机的连接.....\n");                   //输出信息
                server.accept();                                           //等待客户机连接
            }
        } catch (SocketTimeoutException e) {
            ta_info.append("连接超时.....");
            JOptionPane.showMessageDialog(null, "连接超时.....");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

16.2.5　范例 4：获得 Socket 信息

本范例演示在网络通信程序中，如何获取远程服务器和客户机的 IP 地址与端口号。运行程序，结果如图 16.13 和图 16.14 所示，其中图 16.13 是服务器端程序，用于等待客户端的连接；图 16.14 是客户端程序，用于获得远程服务器和客户机的 IP 地址与端口号。（**实例位置：光盘\TM\sl\16\7**）

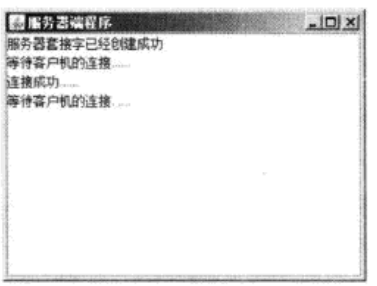

图 16.13　服务器端程序

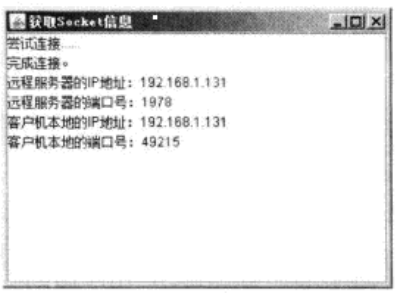

图 16.14　获取 Socket 信息

（1）在 Eclipse 中新建项目 07，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

（3）在服务器窗体类 ServerSocketFrame 的内容面板上添加一个 JScrollPane 滚动面板控件，然后在滚动面板的视图上放置一个 JTextArea 文本域控件。文本域控件的名称为 ta_info，用于显示客户端与服务器的连接信息。

（4）在客户端窗体类 ClientSocketFrame 的内容面板上添加一个 JScrollPane 滚动面板控件，然后在滚动面板的视图上放置一个 JTextArea 文本域控件。文本域控件的名称为 ta_info，用于显示远程服务器和客户机的 IP 地址与端口号。

（5）客户端窗体类 ClientSocketFrame 中，用于显示远程服务器和客户机的 IP 地址与端口号的代码定义在 connect()方法中，该方法的代码如下：

```
private void connect() {                                                    //连接套接字方法
    ta.append("尝试连接.....\n");                                            //文本域中信息
    try {                                                                   //捕捉异常
        socket = new Socket("192.168.1.131", 1978);                         //实例化 Socket 对象
        ta.append("完成连接。\n");                                           //文本域中提示信息
        InetAddress netAddress = socket.getInetAddress();                   //获得远程服务器的地址
        String netIp = netAddress.getHostAddress();                         //获得远程服务器的 IP 地址
        int netPort = socket.getPort();                                     //获得远程服务器的端口号
        InetAddress localAddress = socket.getLocalAddress();                //获得客户端的地址
        String localIp = localAddress.getHostAddress();                     //获得客户端的 IP 地址
        int localPort = socket.getLocalPort();                              //获得客户端的端口号
        ta.append("远程服务器的 IP 地址：" + netIp + "\n");
        ta.append("远程服务器的端口号：" + netPort + "\n");
        ta.append("客户机本地的 IP 地址：" + localIp + "\n");
        ta.append("客户机本地的端口号：" + localPort + "\n");
    } catch (Exception e) {
        e.printStackTrace();                                                //输出异常信息
    }
}
```

16.3 UDP 程序设计基础

视频讲解：光盘\TM\lx\16\UDP 程序设计基础.exe

用户数据包协议（UDP）是网络信息传输的另一种形式。基于 UDP 的通信和基于 TCP 的通信不同，基于 UDP 的信息传递更快，但不提供可靠的保证。使用 UDP 传递数据时，用户无法知道数据能否正确地到达主机，也不能确定到达目的地的顺序是否和发送的顺序相同。虽然 UDP 是一种不可靠的协议，但有时需要较快地传输信息，能容忍小的错误，可以考虑使用 UDP。

基于 UDP 通信的基本模式如下：

☑ 将数据打包（称为数据包），然后将数据包发往目的地。

☑ 接收别人发来的数据包，然后查看数据包。

发送数据包：

（1）使用 DatagramSocket()方法创建一个数据包套接字。

（2）使用 DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)方法创建要发送的数据包。

（3）使用 DatagramSocket 类的 send()方法发送数据包。

接收数据包：

（1）使用 DatagramSocket(int port)方法创建数据包套接字，绑定到指定的端口。

（2）使用 DatagramPacket(byte[] buf, int length)方法创建字节数组来接收数据包。

（3）使用 DatagramPacket 类的 receive()方法接收 UDP 包。

使用 DatagramSocket 类的 receive()方法接收数据时，如果还没有可以接收的数据，在正常情况下 receive()方法将阻塞，一直等到网络上有数据到来才接收该数据并返回。如果网络上没有数据发送过来，receive()方法也没有阻塞，肯定是程序有问题，大多数是使用了一个被其他程序占用的端口号。

16.3.1　DatagramPacket 类

java.net 包的 DatagramPacket 类用来表示数据包。DatagramPacket 类的构造函数如下：

☑ DatagramPacket(byte[] buf, int length)

☑ DatagramPacket(byte[] buf, int length, InetAddress address, int port)

第一种构造函数创建 DatagramPacket 对象，指定了数据包的内存空间和大小。第二种构造函数不仅指定了数据包的内存空间和大小，而且指定了数据包的目标地址和端口。在发送数据时，必须指定接收方的 Socket 地址和端口号，因此使用第二种构造函数可创建发送数据的 DatagramPacket 对象。

16.3.2　DatagramSocket 类

java.net 包中的 DatagramSocket 类用于表示发送和接收数据包的套接字。该类的构造函数如下：

☑ DatagramSocket()

☑ DatagramSocket(int port)

☑ DatagramSocket(int port, InetAddress addr)

第一种构造函数创建 DatagramSocket 对象，构造数据包套接字并将其绑定到本地主机上任何可用的端口。使用第二种构造函数创建 DatagramSocket 对象，创建数据包套接字并将其绑定到本地主机上的指定端口。第三种构造函数创建 DatagramSocket 对象，创建数据包套接字，将其绑定到指定的本地地址。第三种构造函数适用于有多块网卡和多个 IP 的情况。

在接收程序时，必须指定一个端口号，不要让系统随机产生。可以使用第二种构造函数。就像有个朋友要你给他写信，可他的地址不确定是不行的。在发送程序时，通常使用第一种构造函数，不指定端口号，这样系统就会为我们分配一个端口号。就像寄信一样，不需要到指定的邮局去寄。

16.3.3　UDP 网络程序

根据前面所讲的网络编程的基本知识，以及 UDP 网络编程的特点，下面创建一个广播数据包程序。广播数据包是一种较新的技术，类似于广播电台，广播电台需要在指定的波段和频率上广播信息，收听者也要将收音机调到指定的波段、频率才可以收听广播内容。

【例 16.4】 主机不断地重复播出节目预报，可以保证加入到同一组的主机随时可接收到广播信息。接收者将正在接收的信息放在一个文本域中，并将接收的全部信息放在另一个文本域中。（**实例位置：光盘\TM\sl\16\8**）

（1）广播主机程序不断地向外发出广播信息，代码如下：

```
public class Weather extends Thread {                                    //创建类，该类为多线程执行程序
    String weather = "节目预报：八点有大型晚会，请收听";                    //需要发出的广播信息
    int port = 9898;                                                     //定义端口
    InetAddress iaddress = null;                                         //创建 InetAddress 对象
    MulticastSocket socket = null;                                       //声明多点广播套接字
    Weather() {                                                          //构造方法
        try {
            iaddress = InetAddress.getByName("224.255.10.0");            //实例化 InetAddress，指定地址
            socket = new MulticastSocket(port);                          //实例化多点广播套接字
            socket.setTimeToLive(1);                                     //指定发送范围是本地网络
            socket.joinGroup(iaddress);                                  //加入广播组
        } catch (Exception e) {
            e.printStackTrace();                                         //输出异常信息
        }
    }
    public void run() {                                                  //run()方法
        while (true) {
            DatagramPacket packet = null;                                //声明 DatagramPacket 对象
            byte data[] = weather.getBytes();                            //声明字节数组
            packet = new DatagramPacket(data, data.length, iaddress, port); //将数据打包
            System.out.println(new String(data));                        //将广播信息输出
            try {
                socket.send(packet);                                     //发送数据
                sleep(3000);                                             //线程休眠
            } catch (Exception e) {
                e.printStackTrace();                                     //输出异常信息
            }
        }
    }
    public static void main(String[] args) {                            //主方法
        Weather w = new Weather();                                       //创建本类对象
        w.start();                                                       //启动线程
    }
}
```

运行结果如图 16.15 所示。

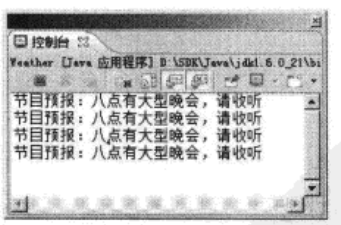

图 16.15 广播主机程序的运行结果

（2）接收广播程序。当单击“开始接收”按钮时，开始接收主机播出的信息；当单击“停止接收”按钮时，系统会停止接收广播主机播出的信息。代码如下：

```
public class Receive extends JFrame implements Runnable, ActionListener {
    int port;                                                          //定义 int 型变量
    InetAddress group = null;                                          //声明 InetAddress 对象
    MulticastSocket socket = null;                                     //创建多点广播套接字对象
    JButton ince = new JButton("开始接收");                             //创建按钮对象
    JButton stop = new JButton("停止接收");
    JTextArea inceAr = new JTextArea(10, 10);                          //显示接收广播的文本域
    JTextArea inced = new JTextArea(10, 10);
    Thread thread;                                                     //创建 Thread 对象
    boolean b = false;                                                 //创建 boolean 型变量
    public Receive() {                                                 //构造方法
        super("广播数据报");                                            //调用父类方法
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        thread = new Thread(this);                                     //创建线程对象
        ince.addActionListener(this);                                  //绑定按钮 ince 的单击事件
        stop.addActionListener(this);                                  //绑定按钮 stop 的单击事件
        inceAr.setForeground(Color.blue);                              //指定文本域中文字颜色
        JPanel north = new JPanel();                                   //创建 JPanel 对象
        north.add(ince);                                               //将按钮添加到面板 north 上
        north.add(stop);
        add(north, BorderLayout.NORTH);                                //将 north 放置在窗体的上部
        JPanel center = new JPanel();                                  //创建面板对象 center
        center.setLayout(new GridLayout(1, 2));                        //设置面板布局
        center.add(inceAr);                                            //将文本域添加到面板上
        center.add(inced);
        add(center, BorderLayout.CENTER);                              //设置面板布局
        validate();                                                    //刷新
        port = 9898;                                                   //设置端口号
        try {
            group = InetAddress.getByName("224.255.10.0");             //指定接收地址
            socket = new MulticastSocket(port);                        //绑定多点广播套接字
            socket.joinGroup(group);                                   //加入广播组
        } catch (Exception e) {
            e.printStackTrace();                                       //输出异常信息
        }
        setBounds(100, 50, 360, 380);                                  //设置布局
        setVisible(true);                                              //将窗体设置为显示状态
    }
    public void run() {                                                //run()方法
        while (true) {
            byte data[] = new byte[1024];                              //创建 byte 数组
            DatagramPacket packet = null;                              //创建 DatagramPacket 对象
            packet = new DatagramPacket(data, data.length, group, port); //待接收的数据包
            try {
                socket.receive(packet);                                //接收数据包
                String message = new String(packet.getData(), 0, packet
```

```
                    .getLength());                                    //获取数据包中内容
                inceAr.setText("正在接收的内容：\n" + message);          //将接收内容显示在文本域中
                inced.append(message + "\n");                         //每条信息为一行
            } catch (Exception e) {
                e.printStackTrace();                                  //输出异常信息
            }
            if (b == true) {                                          //当变量等于 true 时，退出循环
                break;
            }
        }
    }
    public void actionPerformed(ActionEvent e) {                      //单击事件
        if (e.getSource() == ince) {                                  //单击按钮 ince 触发的事件
            ince.setBackground(Color.red);                            //设置按钮颜色
            stop.setBackground(Color.yellow);
            if (!(thread.isAlive())) {                                //如线程不处于新建状态
                thread = new Thread(this);                            //实例化 Thread 对象
            }
            thread.start();                                           //启动线程
            b = false;                                                //设置变量值
        }
        if (e.getSource() == stop) {                                  //单击按钮 stop 触发的事件
            ince.setBackground(Color.yellow);                         //设置按钮颜色
            stop.setBackground(Color.red);
            b = true;                                                 //设置变量值
        }
    }
    public static void main(String[] args) {                         //主方法
        Receive rec = new Receive();                                  //创建本类对象
        rec.setSize(460, 200);                                        //设置窗体大小
    }
}
```

运行结果如图 16.16 所示。

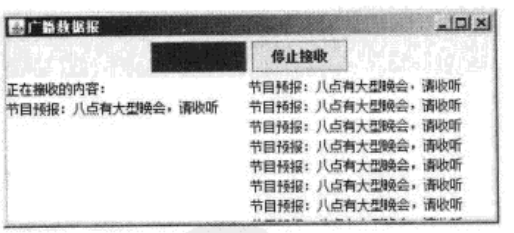

图 16.16　接收广播的运行结果

说明　要广播或接收广播的主机地址必须加入到一个组内，地址在 224.0.0.0～224.255.255.255 之间，这类地址并不代表某个特定主机的位置。加入到同一个组的主机可以在某个端口上广播信息，也可以在某个端口上接收信息。

16.4　经 典 范 例

网络通信在以后的开发中非常重要，现在做几个关于网络通信的例子。

16.4.1　经典范例 1：聊天室服务器端

视频讲解：光盘\TM\lx\16\聊天室服务器端.exe

本范例实现了聊天室服务器端的功能，运行程序，服务器端等待客户端的连接，并显示客户端的连接信息。运行结果如图 16.17 所示。（**实例位置：光盘\TM\sl\16\9**）

（1）在 Eclipse 中新建项目 09，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ChatServerFrame，然后在窗体上添加一个滚动面板，并在滚动面板上添加一个文本域控件，完成窗体界面的设计。

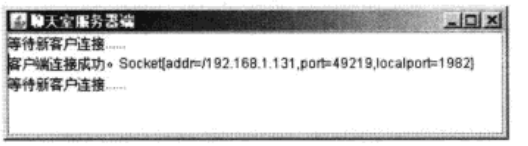

图 16.17　聊天室服务器端

（3）在服务器窗体类 ChatServerFrame 的成员声明区定义一个 Hashtable 对象，用于存储登录用户的用户名和套接字对象。代码如下：

```
//用于存储连接到服务器的用户名和客户端套接字对象
private Hashtable<String, Socket> map = new Hashtable<String, Socket>();
```

（4）在服务器窗体类 ChatServerFrame 中，定义 createSocket()方法，用于创建服务器套接字对象、获得连接到服务器的客户端套接字对象以及启动线程对象对客户端发送的信息进行处理。代码如下：

```
public void createSocket() {
    try {
        server = new ServerSocket(1982);                          //创建服务器套接字对象
        while (true) {
            ta_info.append("等待新客户连接.....\n");
            socket = server.accept();                             //获得套接字对象
            ta_info.append("客户端连接成功。" + socket + "\n");
            new ServerThread(socket).start();                     //创建并启动线程对象
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

（5）在服务器窗体类 ChatServerFrame 中定义内部线程类 ServerThread，用于对客户端的连接信息以及发送的信息进行处理和转发。代码如下：

```
class ServerThread extends Thread {
    //省略了部分代码
    public void run() {
        try {
            //获得输入流对象
            ObjectInputStream ins = new ObjectInputStream(socket.getInputStream());
            while (true) {
                //省略了部分代码
                if (v != null && v.size() > 0) {
                    for (int i = 0; i < v.size(); i++) {
                        String info = (String) v.get(i);                    //读取信息
                        String key = "";
                        if (info.startsWith("用户：")) {                    //添加登录用户到客户端列表
                            //省略了部分代码
                        } else if (info.startsWith("退出：")) {
                            //省略了部分代码
                        } else {                                            //转发接收的消息
                            //获得接收方的 key 值，即接收方的用户名
                            key = info.substring(info.indexOf("：发送给：") + 5,
                                    info.indexOf("：的信息是："));
                            //获得发送方的 key 值，即发送方的用户名
                            String sendUser = info.substring(0, info
                                    .indexOf("：发送给："));
                            Set<String> set = map.keySet();                 //获得集合中所有键的 Set 视图
                            Iterator<String> keyIt = set.iterator();        //获得所有键的迭代器
                            while (keyIt.hasNext()) {
                                String receiveKey = keyIt.next();           //获得表示接收信息的键
                                //与接受用户相同，但不是发送用户
                                if (key.equals(receiveKey) && !sendUser.equals(receiveKey)) {
                                    Socket s = map.get(receiveKey); //获得与该键对应的套接字对象
                                                                            //创建输出流对象
                                    PrintWriter out = new PrintWriter(s.getOutputStream(), true);
                                    out.println("MSG:" + info);             //发送信息
                                    out.flush();                            //刷新输出缓冲区
                                }
                            }
                        }
                    }
                }
            }
        } catch (IOException e) {
            ta_info.append(socket + "已经退出。\n");
        }
    }
}
```

16.4.2　经典范例 2：聊天室客户端

视频讲解：光盘\TM\lx\16\聊天室客户端.exe

本范例实现了聊天室客户端，运行程序，用户登录服务器后，可以从用户列表中选择单个用户进行聊天，也可以选择多个用户进行聊天。运行结果如图 16.18 所示。（**实例位置：光盘\TM\sl\16\10**）

图 16.18　聊天室客户端

（1）在 Eclipse 中新建项目 10，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的客户端窗体类 ChatClientFrame，用于进行用户登录，发送聊天信息和显示聊天信息，在该类中完成窗体界面的设计。

（3）在客户端窗体类 ChatClientFrame 中定义 createClientSocket() 方法，用于创建套接字对象、创建输出流对象以及启动线程对象对服务器转发的信息进行处理。代码如下：

```
public void createClientSocket() {
    try {
        Socket socket = new Socket("192.163.1.131", 1982);          //创建套接字对象
        out = new ObjectOutputStream(socket.getOutputStream());     //创建输出流对象
        new ClientThread(socket).start();                           //创建并启动线程对象
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

（4）在客户端窗体类 ChatClientFrame 中，定义内部线程类 ClientThread，用于对服务器端转发的信息进行处理，并显示在相应的控件中。

（5）在客户端窗体类 ChatClientFrame 中，为“登录”按钮添加实现用户登录功能的代码。“登录”按钮的事件代码如下：

```
if (loginFlag) {                                                    //已登录标记为 true
    JOptionPane.showMessageDialog(null, "在同一窗口只能登录一次。");
    return;
}
String userName = tf_newUser.getText().trim();                      //获得登录用户名
Vector v = new Vector();                                            //定义向量，用于存储登录用户
v.add("用户：" + userName);                                         //添加登录用户
try {
    out.writeObject(v);                                             //将用户向量发送到服务器
    out.flush();                                                    //刷新输出缓冲区
} catch (IOException ex) {
```

```
        ex.printStackTrace();
    }
    tf_newUser.setEnabled(false);                                    //禁用用户文本框
    loginFlag = true;                                                //将已登录标记设置为 true
```

（6）在客户端窗体类 ChatClientFrame 中定义发送聊天信息的 send()方法。代码如下：

```
private void send() {
    if (!loginFlag) {
        JOptionPane.showMessageDialog(null, "请先登录。");
        return;                                                      //如果用户没登录则返回
    }
    String sendUserName = tf_newUser.getText().trim();               //获得登录用户名
    String info = tf_send.getText();                                 //获得输入的发送信息
    if (info.equals("")) {
        return;                                                      //如果没输入信息则返回，即不发送
    }
    Vector<String> v = new Vector<String>();                         //创建向量对象，用于存储发送的消息
    Object[] receiveUserNames = user_list.getSelectedValues();       //获得选择的用户数组
    if (receiveUserNames.length <= 0) {
        return;                                                      //如果没选择用户则返回
    }
    for (int i = 0; i < receiveUserNames.length; i++) {
        String msg = sendUserName + "：发送给：" + (String) receiveUserNames[i]
                + "：的信息是：  " + info;                            //定义发送的信息
        v.add(msg);                                                  //将信息添加到向量
    }
    try {
        out.writeObject(v);                                          //将向量写入输出流，完成信息的发送
        out.flush();                                                 //刷新输出缓冲区
    } catch (IOException e) {
        e.printStackTrace();
    }
    DateFormat df = DateFormat.getDateInstance();                    //获得 DateFormat 实例
    String dateString = df.format(new Date());                       //格式化为日期
    df = DateFormat.getTimeInstance(DateFormat.MEDIUM);              //获得 DateFormat 实例
    String timeString = df.format(new Date());                       //格式化为时间
    String sendUser = tf_newUser.getText().trim();                   //获得发送信息的用户
    String receiveInfo = tf_send.getText().trim();                   //显示发送的信息
     //在文本域中显示信息
    ta_info.append("   "+sendUser + "      " +dateString+"   "+timeString+"\n   "+receiveInfo+"\n");
    tf_send.setText(null);                                           //清空文本框
    ta_info.setSelectionStart(ta_info.getText().length()-1);         //设置选择的起始位置
    ta_info.setSelectionEnd(ta_info.getText().length());             //设置选择的结束位置
    tf_send.requestFocus();                                          //使发送信息文本框获得焦点
}
```

16.5 本章小结

本章向读者介绍了Java网络编程知识，对于网络协议等内容，程序设计人员都应该有所了解，有兴趣的读者还可以查阅其他资料来获取更详细的信息。TCP、UDP网络编程的区别、java.net包中提供的网络应用程序的常用类，以及这些类中的常用方法是本章的重点。通过本章的学习，读者应该自己尝试着编写一些网络程序来巩固所学知识。

16.6 实战练习

1．在使用套接字进行网络编程时，有时需要通过Socket传递对象，做一个使用Socket传递对象的例子。（**答案位置：光盘\TM\sl\16\11**）

2．在使用套接字进行网络编程时，有时需要通过Socket传输图片，做一个通过套接字传输图片的例子，要求在服务器端选择图片，单击“发送”按钮，就会将图片发送到客户端。（**答案位置：光盘\TM\sl\16\12**）

3．在使用套接字进行网络编程时，使用Socket传递汉字有时会出现乱码，做一个例子防止出现汉字乱码的问题。（**答案位置：光盘\TM\sl\16\13**）