


第19章

网络通信

( 视频讲解：24 分钟)

Internet 提供了大量、多样的信息，很少有人能在接触过 Internet 后拒绝它的诱惑。计算机网络实现了多个计算机互连系统，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。本章将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：

- » 了解网络程序设计基础
- » 学会编写 TCP 程序
- » 学会编写 UDP 程序

19.1 网络程序设计基础

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。

19.1.1 局域网与因特网

 视频讲解：光盘\TM\lx\19\局域网与因特网.exe

为了实现两台计算机的通信，必须用一个网络线路连接两台计算机，如图 19.1 所示。



图 19.1 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。我们通常所说的局域网（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）。我们熟悉的因特网（Internet），就是由无数的 LAN 和 WAN 组成的。

19.1.2 网络协议

 视频讲解：光盘\TM\lx\19\网络协议.exe

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 是 Internet Protocol 的简称，它是一种网络协议。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 网络上存在数以亿计的主机，每一台主机在网络上用为其分配的 Internet 地址代表自己，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 19.2 所示。

2. TCP 与 UDP 协议

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 可以保证从一端数据送至连接的另一端时, 数据能够确实送达, 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, 必须先拨号给对方, 等两端确定连接后, 相互才能听到对方说话, 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不重要, 收信人接收信件的顺序也不能保证与寄出信件顺序相同。

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。这是由于 TCP 协议在认证上存在额外耗费, 可能使传输速度减慢, 而 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遗失或传送顺序有所不同, 也不会严重危害该项通信。

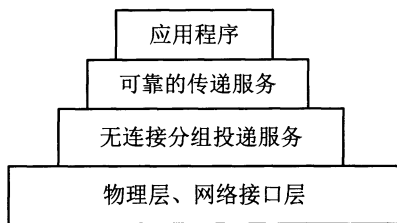


图 19.2 TCP/IP 层次结构



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, 应先确定所在网络是否允许 UDP 协议。

19.1.3 端口和套接字

视频讲解: 光盘\TM\19\端口和套接字.exe

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, 那么客户机会通过不同的端口来确定连接到服务器的哪项服务上, 如图 19.3 所示。

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。

网络程序中的套接字 (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一样，如图 19.4 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。

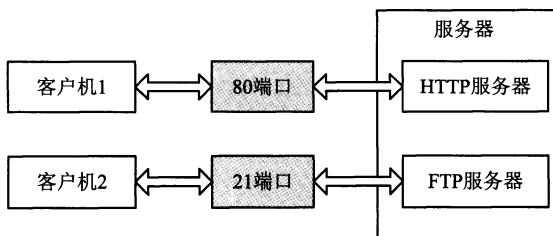


图 19.3 端口

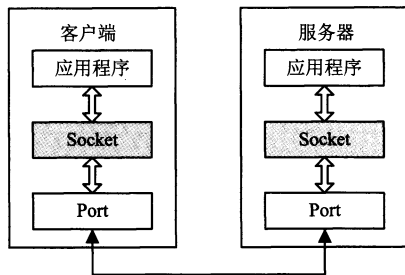


图 19.4 套接字

19.2 TCP 程序设计基础

TCP 网络程序设计是指利用 Socket 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 19.5 所示。

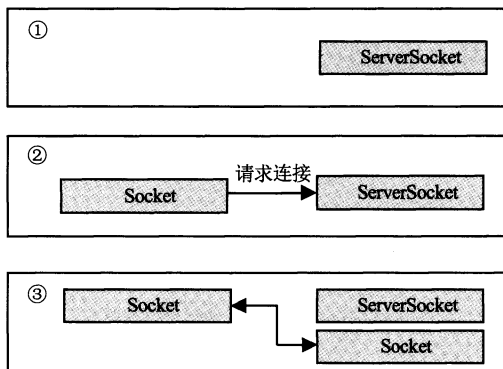


图 19.5 服务器端与客户端的交互

- 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

19.2.1 InetAddress 类

 视频讲解：光盘\TM\lx\19\InetAddress 类.exe

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 19.1 所示。

表 19.1 InetAddress 类的常用方法

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

【例 19.1】 使用 InetAddress 类的 getHostName() 和 getHostAddress() 方法获得本地主机的本机名、本机 IP 地址。（实例位置：光盘\TM\19\19.01）

```
import java.net.*;           //导入 java.net 包
public class Address {       //创建类
    public static void main(String[] args) {
        InetAddress ip;      //创建 InetAddress 对象
        try {                //使用 try 语句块捕捉可能出现的异常
            ip = InetAddress.getLocalHost(); //实例化对象
            String localname = ip.getHostName(); //获取本机名
            String localip = ip.getHostAddress(); //获取本机 IP 地址
            System.out.println("本机名: " + localname); //将本机名输出
            System.out.println("本机 IP 地址: " + localip); //将本机 IP 地址输出
        } catch (UnknownHostException e) {
            e.printStackTrace(); //输出异常信息
        }
    }
}
```

运行结果如图 19.6 所示。

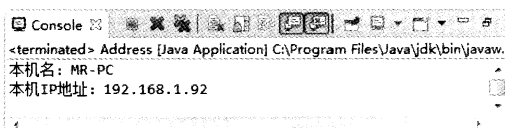


图 19.6 例 19.1 的运行结果



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。

19.2.2 ServerSocket 类

 视频讲解：光盘\TM\19\ServerSocket 类.exe

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。队列的默认大小是 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 地址的情况，用于可以明确规定 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 19.2 所示。

表 19.2 ServerSocket 类的常用方法

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

调用 ServerSocket 类的 accept() 方法会返回一个和客户端 Socket 对象相连接的 Socket 对象，服务器端的 Socket 对象使用 getOutputStream() 方法获得的输出流将指向客户端 Socket 对象使用 getInputStream() 方法获得的那个输入流；同样，服务器端的 Socket 对象使用 getInputStream() 方法获得的输入流将指向客户端 Socket 对象使用 getOutputStream() 方法获得的那个输出流。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 System.out.println("连接中") 语句将不会执行。语句如果没有客户请求，accept() 方法没有发生阻塞，肯定是程序出现了问题。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");
```

19.2.3 TCP 网络程序

 视频讲解：光盘\TM\lx\19\TCP 网络程序.exe

明白了 TCP 程序工作的过程，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, 不要求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, 服务器则通过输入流接收数据。下面是简单的单向通信的实例。

【例 19.2】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\19.02)

```
import java.io.*;           //导入 java.io 包
import java.net.*;         //导入 java.net 包
public class MyTcp {       //创建类 MyTcp
    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
                //获得客户端信息
                System.out.println("客户机:" + reader.readLine());
            }
        } 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 19.7 所示。

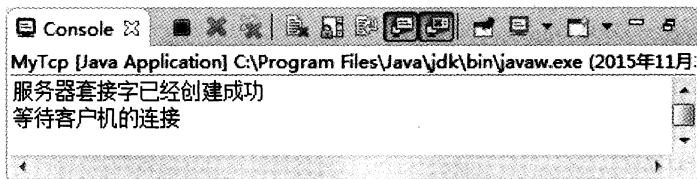


图 19.7 例 19.2 的运行结果

运行服务器端程序，将输出提示信息，等待客户呼叫。下面再来看一下客户端程序。

【例 19.3】 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\19.03）

```
package com.lzw;
public class MyClien extends JFrame {
    private PrintWriter writer;           //创建类继承 JFrame 类
    Socket socket;                        //声明 PrintWriter 类对象
    private JTextArea ta = new JTextArea(); //声明 Socket 对象
    private JTextField tf = new JTextField(); //创建 JTextArea 对象
    Container cc;                         //创建 JTextField 对象
    public MyClien(String title) {        //声明 Container 对象
        super(title);                    //构造方法
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //调用父类的构造方法
        cc = this.getContentPane();      //实例化对象
        final JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBorder(new BevelBorder(BevelBorder.RAISED));
        getContentPane().add(scrollPane, BorderLayout.CENTER);
        scrollPane.setViewportViewView(ta);
        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) {
                //将文本框中的信息写入流
                writer.println(tf.getText());
                //将文本框中的信息显示在文本域中
                ta.append(tf.getText() + '\n');
                ta.setSelectionEnd(ta.getText().length());
                tf.setText("");             //将文本框清空
            }
        });
    }
    private void connect() {
        ta.append("尝试连接\n");          //连接套接字方法
        try {                             //文本域中提示信息
            socket = new Socket("127.0.0.1", 8998); //捕捉异常
            writer = new PrintWriter(socket.getOutputStream(), true); //实例化 Socket 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) {
    MyClien clien = new MyClien("向服务器送数据");
    clien.setSize(200, 200);
    clien.setVisible(true);
    clien.connect();
}
}
```

//主方法
 //创建本例对象
 //设置窗体大小
 //将窗体显示
 //调用连接方法

运行服务器端，再运行这个客户端，运行结果如图 19.8 所示。

从图 19.8 中可以看出，客户端与服务器端已经创建了连接。向文本框中输入信息，会发现输入的信息在服务器端输出，并在客户端的文本域中显示，如图 19.9 和图 19.10 所示。

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。还可能遇到以前运行良好的网络程序突然运行不了的情况，这种情况很可能也是由于端口被别的程序占用了。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 19.11 所示。



图 19.8 例 19.3 的运行结果

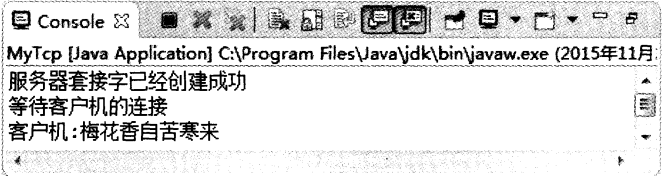


图 19.9 服务器端运行结果

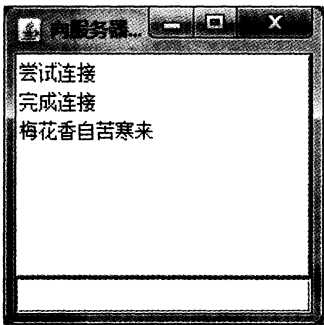


图 19.10 客户端运行结果

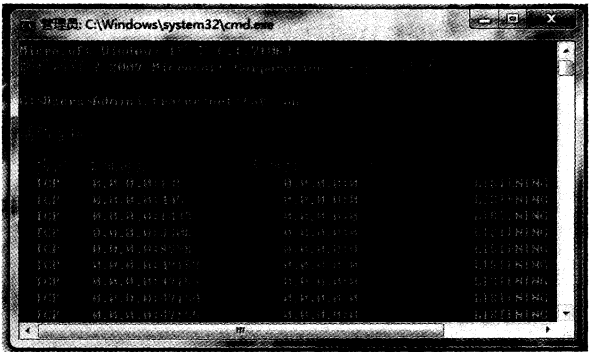


图 19.11 查看端口

19.3 UDP 程序设计基础

用户数据报协议（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，但如果需要较快地传输信息，并能容忍小的错误，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。

下面是总结的 UDP 程序的步骤。

发送数据包：

- （1）使用 `DatagramSocket()` 创建一个数据包套接字。
- （2）使用 `DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)` 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)` 创建数据包套接字，绑定到指定的端口。
- （2）使用 `DatagramPacket(byte[] buf, int length)` 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。



注意

`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传来，`receive()` 方法接收该数据并返回。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。

19.3.1 DatagramPacket 类



视频讲解：光盘\TM\lx\19\DatagramPacket 类.exe

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

- ☑ `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，还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。

19.3.2 DatagramSocket 类

 视频讲解：光盘\TM\lx\19\DatagramSocket 类.exe

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

- ☑ DatagramSocket()。
- ☑ DatagramSocket(int port)。
- ☑ DatagramSocket(int port, InetAddress addr)。

第一种构造函数创建 DatagramSocket 对象，构造数据报套接字并将其绑定到本地主机上任何可用的端口。第二种构造函数创建 DatagramSocket 对象，创建数据报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，创建数据报套接字并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 地址的情况。

在接收程序时，必须指定一个端口号，不要让系统随机产生，此时可以使用第二种构造函数。比如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。就像寄信不需要到指定的邮局去寄一样。

19.3.3 UDP 网络程序

 视频讲解：光盘\TM\lx\19\UDP 网络程序.exe

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报程序。广播数据报是一种较新的技术，类似于电台广播，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 19.4】 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\19.04）

(1) 广播主机程序不断地向外播出信息，代码如下：

```
import java.net.*;
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";
    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 {
            //实例化 InetAddress，指定地址
            iaddress = InetAddress.getByName("224.255.10.0");
            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
        //将数据打包
        packet = new DatagramPacket(data, data.length, iaddress, port);
        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 19.12 所示。

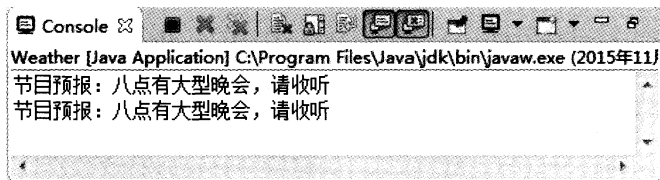


图 19.12 广播主机程序的运行结果

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

```

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import javax.swing.*;
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;
public Receive() {
    super("广播数据报");
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    thread = new Thread(this);
    ince.addActionListener(this);
    stop.addActionListener(this);
    inceAr.setForeground(Color.blue);
    JPanel north = new JPanel();
    north.add(ince);
    north.add(stop);
    add(north, BorderLayout.NORTH);
    JPanel center = new JPanel();
    center.setLayout(new GridLayout(1, 2));
    center.add(inceAr);
    center.add(incd);
    add(center, BorderLayout.CENTER);
    validate();
    port = 9898;
    try {
        group = InetAddress.getByName("224.255.10.0");
        socket = new MulticastSocket(port);
        socket.joinGroup(group);
    } catch (Exception e) {
        e.printStackTrace();
    }
    setBounds(100, 50, 360, 380);
    setVisible(true);
}

public void run() {
    while (true) {
        byte data[] = new byte[1024];
        DatagramPacket packet = null;
        //待接收的数据包
        packet = new DatagramPacket(data, data.length, group, port);
        try {
            socket.receive(packet);
            String message = new String(packet.getData(), 0, packet
                .getLength());
            //将接收内容显示在文本域中
            inceAr.setText("正在接收的内容: \n" + message);
            incd.append(message + "\n");
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (b == true) {
            break;
        }
    }
}

```

//创建 boolean 型变量
 //构造方法
 //调用父类方法
 //绑定按钮 ince 的单击事件
 //绑定按钮 stop 的单击事件
 //指定文本域中文字的颜色
 //创建 JPanel 对象
 //将按钮添加到面板 north 上
 //将 north 放置在窗体的上部
 //创建面板对象 center
 //设置面板布局
 //将文本域添加到面板上
 //设置面板布局
 //刷新
 //设置端口号
 //指定接收地址
 //绑定多点广播套接字
 //加入广播组
 //输出异常信息
 //设置布局
 //将窗体设置为显示状态
 //run()方法
 //创建 byte 数组
 //创建 DatagramPacket 对象
 //接收数据包
 //获取数据包中的内容
 //每条信息为一行
 //输出异常信息
 //当变量等于 true 时, 退出循环

```

}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == ince) {
        ince.setBackground(Color.red);
        stop.setBackground(Color.yellow);
        if (!(thread.isAlive())) {
            thread = new Thread(this);
        }
        thread.start();
        b = false;
    }
    if (e.getSource() == stop) {
        ince.setBackground(Color.yellow);
        stop.setBackground(Color.red);
        b = true;
    }
}
public static void main(String[] args) {
    Receive rec = new Receive();
    rec.setSize(460, 200);
}
}

```

//单击事件
 //单击按钮 ince 触发的事件
 //设置按钮颜色
 //如线程不处于“新建状态”
 //实例化 Thread 对象
 //启动线程
 //设置变量值
 //单击按钮 stop 触发的事件
 //设置按钮颜色
 //设置变量值 s
 //主方法
 //创建本类对象
 //设置窗体大小

运行结果如图 19.13 所示。

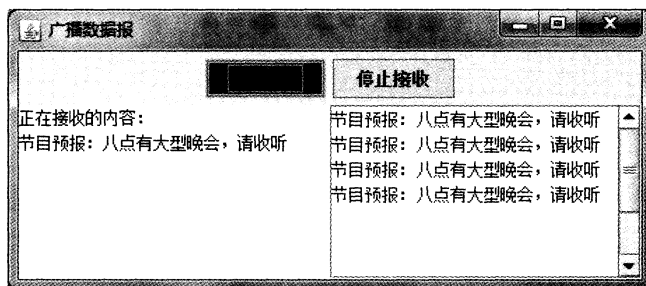


图 19.13 接收广播的运行结果

说明

要广播或接收广播的主机地址必须加入到一个组内，地址范围为 224.0.0.0~224.255.255.255，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。

19.4 小 结

本章介绍了 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 与 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, 读者应该能够自己尝试着编写一些网络程序来巩固所学知识。

19.5 实践与练习

1. 编写 Java 程序, 获得指定端口的主机名、主机地址和本机地址。(答案位置: 光盘\TM\sl\19.05)
2. 编写 TCP 服务器程序, 实现创建一个在 8001 端口上等待的 ServerSocket 对象, 当接收到一个客户机的连接请求后, 程序从与客户机建立了连接的 Socket 对象中获得输入/输出流。通过输出流向客户机发送信息。(答案位置: 光盘\TM\sl\19.06)
3. 编写聊天室程序。(答案位置: 光盘\TM\sl\19.07)