互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

17.4 基于 UDP 协议的网络编程

UDP 协议是一种不可靠的网络协议，它在通信实例的两端各建立一个 Socket，但这两个 Socket 之间并没有虚拟链路，这两个 Socket 只是发送、接收数据报的对象。Java 提供了 DatagramSocket 对象作为基于 UDP 协议的 Socket，使用 DatagramPacket 代表 DatagramSocket 发送、接收的数据报。

17.4.1 UD P 协议基础

UDP 协议是英文 User Datagram Protocol 的缩写，即用户数据报协议，主要用来支持那些需要在计算机之间传输数据的网络连接。UDP 协议从问世至今已经被使用了很多年，虽然 UDP 协议目前应用不如 TCP 协议广泛，但 UDP 协议依然是一个非常实用和可行的网络传输层协议。尤其是在一些实时性很强的应用场景中，比如网络游戏、视频会议等，UDP 协议的快速更具有独特的魅力。

UDP 协议是一种面向非连接的协议，面向非连接指的是在正式通信前不必与对方先建立连接，不管对方状态就直接发送。至于对方是否可以接收到这些数据内容，UDP 协议无法控制，因此说 UDP 协议是一种不可靠的协议。UDP 协议适用于一次只传送少量数据、对可靠性要求不高的应用环境。

与前面介绍的 TCP 协议一样，UDP 协议直接位于 IP 协议之上。实际上，IP 协议属于 OSI 参考模型的网络层协议，而 UDP 协议和 TCP 协议都属于传输层协议。

因为 UDP 协议是面向非连接的协议，没有建立连接的过程，因此它的通信效率很高；但也正因为如此，它的可靠性不如 TCP 协议。

UDP 协议的主要作用是完成网络数据流和数据报之间的转换——在信息的发送端，UDP 协议将网络数据流封装成数据报，然后将数据报发送出去；在信息的接收端，UDP 协议将数据报转换成实际数据内容。

提示：

可以认为 UDP 协议的 Socket 类似于码头，数据报则类似于集装箱；码头的作用就是负责发送、接收集装箱，而 DatagramSocket 的作用则是发送、接收数据报。因此对于基于 UDP 协议的通信双方而言，没有所谓的客户端和服务器端的概念。

UDP 协议和 TCP 协议简单对比如下。

- TCP 协议：可靠，传输大小无限制，但是需要连接建立时间，差错控制开销大。
- UDP 协议：不可靠，差错控制开销较小，传输大小限制在 64KB 以下，不需要建立连接。

17.4.2 使用 DatagramSocket 发送、接收数据

Java 使用 DatagramSocket 代表 UDP 协议的 Socket，DatagramSocket 本身只是码头，不维护状态，不能产生 IO 流，它的唯一作用就是接收和发送数据报，Java 使用 DatagramPacket 来代表数据报，DatagramSocket 接收和发送的数据都是通过 DatagramPacket 对象完成的。

先看一下 DatagramSocket 的构造器。

- DatagramSocket()：创建一个 DatagramSocket 实例，并将该对象绑定到本机默认 IP 地址、本机所有可用端口中随机选择的某个端口。
- DatagramSocket(int prot)：创建一个 DatagramSocket 实例，并将该对象绑定到本机默认 IP 地址、指定端口。
- DatagramSocket(int port, InetAddress laddr)：创建一个 DatagramSocket 实例，并将该对象绑定到指定 IP 地址、指定端口。

通过上面三个构造器中的任意一个构造器即可创建一个 DatagramSocket 实例，通常在创建服务器时，创建指定端口的 DatagramSocket 实例——这样保证其他客户端可以将数据发送到该服务器。一旦得到了 DatagramSocket 实例之后，就可以通过如下两个方法来接收和发送数据。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- receive(DatagramPacket p)：从该 DatagramSocket 中接收数据报。
- send(DatagramPacket p)：以该 DatagramSocket 对象向外发送数据报。

从上面两个方法可以看出，使用 DatagramSocket 发送数据报时，DatagramSocket 并不知道将该数据报发送到哪里，而是由 DatagramPacket 自身决定数据报的目的地。就像码头并不知道每个集装箱的目的地，码头只是将这些集装箱发送出去，而集装箱本身包含了该集装箱的目的地。

下面看一下 DatagramPacket 的构造器。

- DatagramPacket(byte[] buf,int length)：以一个空数组来创建 DatagramPacket 对象，该对象的作用是接收 DatagramSocket 中的数据。
- DatagramPacket(byte[] buf, int length, InetAddress addr, int port)：以一个包含数据的数组来创建 DatagramPacket 对象，创建该 DatagramPacket 对象时还指定了 IP 地址和端口——这就决定了该数据报的目的地。
- DatagramPacket(byte[] buf, int offset, int length)：以一个空数组来创建 DatagramPacket 对象，并指定接收到的数据放入 buf 数组中时从 offset 开始，最多放 length 个字节。
- DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)：创建一个用于发送的 DatagramPacket 对象，指定发送 buf 数组中从 offset 开始，总共 length 个字节。

提示：

当 Client/Server 程序使用 UDP 协议时，实际上并没有明显的服务器端和客户端，因为两方都需要先建立一个 DatagramSocket 对象，用来接收或发送数据报，然后使用 DatagramPacket 对象作为传输数据的载体。通常固定 IP 地址、固定端口的 DatagramSocket 对象所在的程序被称为服务器，因为该 DatagramSocket 可以主动接收客户端数据。

在接收数据之前，应该采用上面的第一个或第三个构造器生成一个 DatagramPacket 对象，给出接收数据的字节数组及其长度。然后调用 DatagramSocket 的 receive()方法等待数据报的到来，receive()将一直等待（该方法会阻塞调用该方法的线程），直到收到一个数据报为止。如下代码所示：

```
// 创建一个接收数据的 DatagramPacket 对象
DatagramPacket packet=new DatagramPacket(buf, 256);
// 接收数据报
socket.receive(packet);
```

在发送数据之前，调用第二个或第四个构造器创建 DatagramPacket 对象，此时的字节数组里存放了想发送的数据。除此之外，还要给出完整的目的地址，包括 IP 地址和端口号。发送数据是通过 DatagramSocket 的 send()方法实现的，send()方法根据数据报的目的地址来寻径以传送数据报。如下代码所示：

```
// 创建一个发送数据的 DatagramPacket 对象
DatagramPacket packet = new DatagramPacket(buf, length, address, port);
// 发送数据报
socket.send(packet);
```

提示：

使用 DatagramPacket 接收数据时，会感觉 DatagramPacket 设计得过于烦琐。开发者只关心该 DatagramPacket 能放多少数据，而 DatagramPacket 是否采用字节数组来存储数据完全不想关心。但 Java 要求创建接收数据用的 DatagramPacket 时，必须传入一个空的字节数组，该数组的长度决定了该 DatagramPacket 能放多少数据，这实际上暴露了 DatagramPacket 的实现细节。接着 DatagramPacket 又提供了一个 getData()方法，该方法又可以返回 Datagram Packet 对象里封装的字节数组，该方法更显得有些多余——如果程序需要获取 DatagramPacket 里封装的字节数组，直接访问传给 DatagramPacket 构造器的字节数组实参即可，无须调用该方法。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

当服务器端（也可以是客户端）接收到一个 DatagramPacket 对象后，如果想向该数据报的发送者“反馈”一些信息，但由于 UDP 协议是面向非连接的，所以接收者并不知道每个数据报由谁发送过来，但程序可以调用 DatagramPacket 的如下 3 个方法来获取发送者的 IP 地址和端口。

- InetAddress getAddress()：当程序准备发送此数据报时，该方法返回此数据报的目标机器的 IP 地址；当程序刚接收到一个数据报时，该方法返回该数据报的发送主机的 IP 地址。
- int getPort()：当程序准备发送此数据报时，该方法返回此数据报的目标机器的端口；当程序刚接收到一个数据报时，该方法返回该数据报的发送主机的端口。
- SocketAddress getSocketAddress()：当程序准备发送此数据报时，该方法返回此数据报的目标 SocketAddress；当程序刚接收到一个数据报时，该方法返回该数据报的发送主机的 SocketAddress。

提示：

getSocketAddress()方法的返回值是一个 SocketAddress 对象，该对象实际上就是一个 IP 地址和一个端口号。也就是说，SocketAddress 对象封装了一个 InetAddress 对象和一个代表端口的整数，所以使用 SocketAddress 对象可以同时代表 IP 地址和端口。

下面程序使用 DatagramSocket 实现了 Server/Client 结构的网络通信。本程序的服务器端使用循环 1000 次来读取 DatagramSocket 中的数据报，每当读取到内容之后便向该数据报的发送者送回一条信息。服务器端程序代码如下。

程序清单：codes\17\17.4\UdpServer.java

```
public class UdpServer
{
    public static final int PORT = 30000;
    // 定义每个数据报的最大大小为 4KB
    private static final int DATA_LEN = 4096;
    // 定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    // 以指定字节数组创建准备接收数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff , inBuff.length);
    // 定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket;
    // 定义一个字符串数组，服务器端发送该数组的元素
    String[] books = new String[]
    {
        "疯狂 Java 讲义",
        "轻量级 Java EE 企业应用实战",
        "疯狂 Android 讲义",
        "疯狂 Ajax 讲义"
    };
    public void init()throws IOException
    {
        try(
            // 创建 DatagramSocket 对象
            DatagramSocket socket = new DatagramSocket(PORT))
        {
            // 采用循环接收数据
            for (int i = 0; i < 1000 ; i++ )
            {
                // 读取 Socket 中的数据，读到的数据放入 inPacket 封装的数组里
                socket.receive(inPacket);
                // 判断 inPacket.getData()和 inBuff 是否是同一个数组
                System.out.println(inBuff == inPacket.getData());
                // 将接收到的内容转换成字符串后输出
                System.out.println(new String(inBuff
                    , 0 , inPacket.getLength()));
                // 从字符串数组中取出一个元素作为发送数据
                byte[] sendData = books[i % 4].getBytes();
                // 以指定的字节数组作为发送数据，以刚接收到的 DatagramPacket 的
                // 源 SocketAddress 作为目标 SocketAddress 创建 DatagramPacket
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
                    outPacket = new DatagramPacket(sendData
                        , sendData.length , inPacket.getSocketAddress());
                    // 发送数据
                    socket.send(outPacket);
                }
            }
        }
        public static void main(String[] args)
            throws IOException
        {
            new UdpServer().init();
        }
    }
```

上面程序中的粗体字代码就是使用 DatagramSocket 发送、接收 DatagramPacket 的关键代码，该程序可以接收 1000 个客户端发送过来的数据。

客户端程序代码也与此类似，客户端采用循环不断地读取用户键盘输入，每当读取到用户输入的内容后就将该内容封装成 DatagramPacket 数据报，再将该数据报发送出去；接着把 DatagramSocket 中的数据读入接收用的 DatagramPacket 中（实际上是读入该 DatagramPacket 所封装的字节数组中）。客户端程序代码如下。

程序清单：codes\17\17.4\UdpClient.java

```
public class UdpClient
{
    // 定义发送数据报的目的地
    public static final int DEST_PORT = 30000;
    public static final String DEST_IP = "127.0.0.1";
    // 定义每个数据报的最大大小为 4KB
    private static final int DATA_LEN = 4096;
    // 定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    // 以指定的字节数组创建准备接收数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff , inBuff.length);
    // 定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket = null;
    public void init()throws IOException
    {
        try(
            // 创建一个客户端 DatagramSocket，使用随机端口
            DatagramSocket socket = new DatagramSocket())
        {
            // 初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
            outPacket = new DatagramPacket(new byte[0] , 0
                , InetAddress.getByName(DEST_IP) , DEST_PORT);
            // 创建键盘输入流
            Scanner scan = new Scanner(System.in);
            // 不断地读取键盘输入
            while(scan.hasNextLine())
            {
                // 将键盘输入的一行字符串转换成字节数组
                byte[] buff = scan.nextLine().getBytes();
                // 设置发送用的 DatagramPacket 中的字节数据
                outPacket.setData(buff);
                // 发送数据报
                socket.send(outPacket);
                // 读取 Socket 中的数据，读到的数据放在 inPacket 所封装的字节数组中
                socket.receive(inPacket);
                System.out.println(new String(inBuff , 0
                    , inPacket.getLength()));
            }
        }
    }
    public static void main(String[] args)
        throws IOException
    {
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            new UdpClient().init();
        }
    }
```

上面程序中的粗体字代码同样也是使用 DatagramSocket 发送、接收 DatagramPacket 的关键代码，这些代码与服务器端代码基本相似。而客户端与服务器端的唯一区别在于：服务器端的 IP 地址、端口是固定的，所以客户端可以直接将该数据报发送给服务器端，而服务器端则需要根据接收到的数据报来决定“反馈”数据报的目的地。

读者可能会发现，使用 DatagramSocket 进行网络通信时，服务器端无须也无法保存每个客户端的状态，客户端把数据报发送到服务器端后，完全有可能立即退出。但不管客户端是否退出，服务器端都无法知道客户端的状态。

当使用 UDP 协议时，如果想让一个客户端发送的聊天信息被转发到其他所有的客户端则比较困难，可以考虑在服务器端使用 Set 集合来保存所有的客户端信息，每当接收到一个客户端的数据报之后，程序检查该数据报的源 SocketAddress 是否在 Set 集合中，如果不在就将该 SocketAddress 添加到该 Set 集合中。这样又涉及一个问题：可能有些客户端发送一个数据报之后永久性地退出了程序，但服务器端还将该客户端的 SocketAddress 保存在 Set 集合中……总之，这种方式需要处理的问题比较多，编程比较烦琐。幸好 Java 为 UDP 协议提供了 MulticastSocket 类，通过该类可以轻松地实现多点广播。

17.4.3 使用 MulticastSocket 实现多点广播

DatagramSocket 只允许数据报发送给指定的目标地址，而 MulticastSocket 可以将数据报以广播方式发送到多个客户端。

若要使用多点广播，则需要让一个数据报标有一组目标主机地址，当数据报发出后，整个组的所有主机都能收到该数据报。IP 多点广播（或多点发送）实现了将单一信息发送到多个接收者的广播，其思想是设置一组特殊网络地址作为多点广播地址，每一个多点广播地址都被看做一个组，当客户端需要发送、接收广播信息时，加入到该组即可。

IP 协议为多点广播提供了这批特殊的 IP 地址，这些 IP 地址的范围是 224.0.0.0 至 239.255.255.255。多点广播示意图如图 17.8 所示。

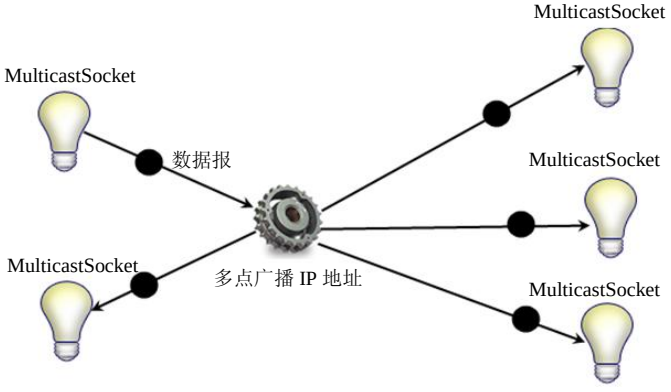

图 17.8 多点广播示意图

从图 17.8 中可以看出，MulticastSocket 类是实现多点广播的关键，当 MulticastSocket 把一个 DatagramPacket 发送到多点广播 IP 地址时，该数据报将被自动广播到加入该地址的所有 MulticastSocket。MulticastSocket 既可以将数据报发送到多点广播地址，也可以接收其他主机的广播信息。

MulticastSocket 有点像 DatagramSocket，事实上 MulticastSocket 是 DatagramSocket 的一个子类，也就是说，MulticastSocket 是特殊的 DatagramSocket。当要发送一个数据报时，可以使用随机端口创建 MulticastSocket，也可以在指定端口创建 MulticastSocket。MulticastSocket 提供了如下 3 个构造器。

- public MulticastSocket()：使用本机默认地址、随机端口来创建 MulticastSocket 对象。
- public MulticastSocket(int portNumber)：使用本机默认地址、指定端口来创建 MulticastSocket 对象。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- public MulticastSocket(SocketAddress bindaddr)：使用本机指定 IP 地址、指定端口来创建 MulticastSocket 对象。

创建 MulticastSocket 对象后，还需要将该 MulticastSocket 加入到指定的多点广播地址，MulticastSocket 使用 joinGroup()方法加入指定组；使用 leaveGroup()方法脱离一个组。

- joinGroup(InetAddress multicastAddr)：将该 MulticastSocket 加入指定的多点广播地址。
- leaveGroup(InetAddress multicastAddr)：让该 MulticastSocket 离开指定的多点广播地址。

在某些系统中，可能有多个网络接口。这可能会给多点广播带来问题，这时候程序需要在一个指定的网络接口上监听，通过调用 setInterface()方法可以强制 MulticastSocket 使用指定的网络接口；也可以使用 getInterface()方法查询 MulticastSocket 监听的网络接口。

提示：

如果创建仅用于发送数据报的 MulticastSocket 对象，则使用默认地址、随机端口即可。但如果创建接收用的 MulticastSocket 对象，则该 MulticastSocket 对象必须具有指定端口，否则发送方无法确定发送数据报的目标端口。

MulticastSocket 用于发送、接收数据报的方法与 DatagramSocket 完全一样。但 MulticastSocket 比 DatagramSocket 多了一个 setTimeToLive(int ttl)方法，该 ttl 参数用于设置数据报最多可以跨过多少个网络，当 ttl 的值为 0 时，指定数据报应停留在本地主机；当 ttl 的值为 1 时，指定数据报发送到本地局域网；当 ttl 的值为 32 时，意味着只能发送到本站点的网络上；当 ttl 的值为 64 时，意味着数据报应保留在本地区；当 ttl 的值为 128 时，意味着数据报应保留在本大洲；当 ttl 的值为 255 时，意味着数据报可发送到所有地方；在默认情况下，该 ttl 的值为 1。

从图 17.8 中可以看出，使用 MulticastSocket 进行多点广播时所有的通信实体都是平等的，它们都将自己的数据报发送到多点广播 IP 地址，并使用 MulticastSocket 接收其他人发送的广播数据报。下面程序使用 MulticastSocket 实现了一个基于广播的多人聊天室。程序只需要一个 MulticastSocket，两个线程，其中 MulticastSocket 既用于发送，也用于接收；一个线程负责接收用户键盘输入，并向 MulticastSocket 发送数据，另一个线程则负责从 MulticastSocket 中读取数据。

程序清单：codes\17\17.4\MulticastSocketTest.java

```
// 让该类实现 Runnable 接口，该类的实例可作为线程的 target
public class MulticastSocketTest implements Runnable
{
    // 使用常量作为本程序的多点广播 IP 地址
    private static final String BROADCAST_IP
        = "230.0.0.1";
    // 使用常量作为本程序的多点广播目的地端口
    public static final int BROADCAST_PORT = 30000;
    // 定义每个数据报的最大大小为 4KB
    private static final int DATA_LEN = 4096;
    // 定义本程序的 MulticastSocket 实例
    private MulticastSocket socket = null;
    private InetAddress broadcastAddress = null;
    private Scanner scan = null;
    // 定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    // 以指定字节数组创建准备接收数据的 DatagramPacket 对象
    private DatagramPacket inPacket
        = new DatagramPacket(inBuff , inBuff.length);
    // 定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket = null;
    public void init()throws IOException
    {
        try(
            // 创建键盘输入流
            Scanner scan = new Scanner(System.in))
        {
            // 创建用于发送、接收数据的 MulticastSocket 对象
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
                // 由于该 MulticastSocket 对象需要接收数据，所以有指定端口
                socket = new MulticastSocket(BROADCAST_PORT);
                broadcastAddress = InetAddress.getByName(BROADCAST_IP);
                // 将该 socket 加入指定的多点广播地址
                socket.joinGroup(broadcastAddress);
                // 设置本 MulticastSocket 发送的数据报会被回送到自身
                socket.setLoopbackMode(false);
                // 初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
                outPacket = new DatagramPacket(new byte[0]
                      , 0 , broadcastAddress , BROADCAST_PORT);
                // 启动以本实例的 run()方法作为线程执行体的线程
                new Thread(this).start();
                // 不断地读取键盘输入
                while(scan.hasNextLine())
                {
                      // 将键盘输入的一行字符串转换成字节数组
                      byte[] buff = scan.nextLine().getBytes();
                      // 设置发送用的 DatagramPacket 里的字节数据
                      outPacket.setData(buff);
                      // 发送数据报
                      socket.send(outPacket);
                }
          }
          finally
          {
                socket.close();
          }
    }
    public void run()
    {
          try
          {
                while(true)
                {
                      // 读取 Socket 中的数据，读到的数据放在 inPacket 所封装的字节数组里
                      socket.receive(inPacket);
                      // 打印输出从 socket 中读取的内容
                      System.out.println("聊天信息: " + new String(inBuff
                            , 0 , inPacket.getLength()));
                }
          }
          // 捕获异常
          catch (IOException ex)
          {
                ex.printStackTrace();
                try
                {
                      if (socket != null)
                      {
                            // 让该 Socket 离开该多点 IP 广播地址
                            socket.leaveGroup(broadcastAddress);
                            // 关闭该 Socket 对象
                            socket.close();
                      }
                      System.exit(1);
                }
                catch (IOException e)
                {
                      e.printStackTrace();
                }
          }
    }
    public static void main(String[] args)
          throws IOException
    {
          new MulticastSocketTest().init();
    }
}
```

上面程序中 init()方法里的第一行粗体字代码先创建了一个 MulticastSocket 对象，由于需要使用该

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

对象接收数据报，所以为该 Socket 对象设置使用固定端口；第二行粗体字代码将该 Socket 对象添加到指定的多点广播 IP 地址；第三行粗体字代码设置该 Socket 发送的数据报会被回送到自身（即该 Socket 可以接收到自己发送的数据报）。至于程序中使用 MulticastSocket 发送、接收数据报的代码，与使用 DatagramSocket 并没有区别，故此处不再赘述。

下面将结合 MulticastSocket 和 DatagramSocket 开发一个简单的局域网即时通信工具，局域网内每个用户启动该工具后，就可以看到该局域网内所有的在线用户，该用户也会被其他用户看到，即看到如图 17.9 所示的窗口。

在图 17.9 所示的用户列表中双击任意一个用户，即可启动一个如图 17.10 所示的交谈窗口。

如果双击图 17.10 所示用户列表窗口中的“所有人”列表项，即可启动一个与图 17.10 相似的交谈窗口，不同的是通过该窗口发送的消息将会被所有人看到。

图 17.9　局域网聊天工具

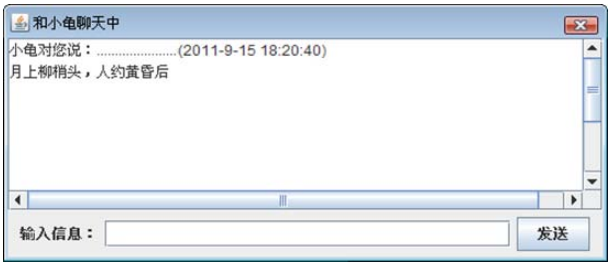

图 17.10　与特定用户交谈

该程序的实现思路是，每个用户都启动两个 Socket，即一个 MulticastSocket，一个 DatagramSocket。其中 MulticastSocket 会周期性地向 230.0.0.1 发送在线信息，且所有用户的 MulticastSocket 都会加入到 230.0.0.1 这个多点广播 IP 地址中，这样每个用户都可以收到其他用户广播的在线信息，如果系统经过一段时间没有收到某个用户广播的在线信息，则从用户列表中删除该用户。除此之外，该 MulticastSocket 还用于向所有用户发送广播信息。

DatagramSocket 主要用于发送私聊信息，当用户收到其他用户广播来的 DatagramPacket 时，即可获取该用户 MulticastSocket 对应的 SocketAddress，这个 SocketAddress 将作为发送私聊信息的重要依据——本程序让 MulticastSocket 在 30000 端口监听，而 DatagramSocket 在 30001 端口监听，这样程序就可以根据其他用户广播来的 DatagramPacket 得到他的 DatagramSocket 所在的地址。

本系统提供了一个 UserInfo 类，该类封装了用户名、图标、对应的 SocketAddress 以及该用户对应的交谈窗口、失去联系的次数等信息。该类的代码片段如下。

程序清单：codes\17\17.4\LanTalk\UserInfo.java

```
public class UserInfo
{
    // 该用户的图标
    private String icon;
    // 该用户的名字
    private String name;
    // 该用户的 MulitcastSocket 所在的 IP 地址和端口
    private SocketAddress address;
    // 该用户失去联系的次数
    private int lost;
    // 该用户对应的交谈窗口
    private ChatFrame chatFrame;
    public UserInfo(){}
    // 有参数的构造器
    public UserInfo(String icon , String name
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            , SocketAddress address , int lost)
        {
            this.icon = icon;
            this.name = name;
            this.address = address;
            this.lost = lost;
        }
        // 省略所有 field 的 setter 和 getter 方法
        ...
        // 使用 address 作为该用户的标识，所以根据 address
        // 重写 hashCode()和 equals()方法
        public int hashCode()
        {
            return address.hashCode();
        }
        public boolean equals(Object obj)
        {
            if (obj != null && obj.getClass() == UserInfo.class)
            {
                UserInfo target = (UserInfo)obj;
                if (address != null)
                {
                    return address.equals(target.getAddress());
                }
            }
            return false;
        }
    }
```

通过 UserInfo 类的封装，所有客户端只需要维护该 UserInfo 类的列表，程序就可以实现广播、发送私聊信息等功能。本程序底层通信的工具类则需要一个 MulticastSocket 和一个 DatagramSocket，该工具类的代码如下。

程序清单：codes\17\17.4\LanTalk\ComUtil.java

```
// 聊天交换信息的工具类
public class ComUtil
{
    // 使用常量作为本程序的多点广播 IP 地址
    private static final String BROADCAST_IP
        = "230.0.0.1";
    // 使用常量作为本程序的多点广播目的地端口
    // DatagramSocket 所用的端口为该端口号-1
    public static final int BROADCAST_PORT = 30000;
    // 定义每个数据报的最大大小为 4KB
    private static final int DATA_LEN = 4096;
    // 定义本程序的 MulticastSocket 实例
    private MulticastSocket socket = null;
    // 定义本程序私聊的 Socket 实例
    private DatagramSocket singleSocket = null;
    // 定义广播的 IP 地址
    private InetAddress broadcastAddress = null;
    // 定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    // 以指定字节数组创建准备接收数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff , inBuff.length);
    // 定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket = null;
    // 聊天的主界面程序
    private LanTalk lanTalk;
    // 构造器，初始化资源
    public ComUtil(LanTalk lanTalk) throws Exception
    {
        this.lanTalk = lanTalk;
        // 创建用于发送、接收数据的 MulticastSocket 对象
        // 因为该 MulticastSocket 对象需要接收数据，所以有指定端口
        socket = new MulticastSocket(BROADCAST_PORT);
        // 创建私聊用的 DatagramSocket 对象
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            singleSocket = new DatagramSocket(BROADCAST_PORT + 1);
            broadcastAddress = InetAddress.getByName(BROADCAST_IP);
            // 将该 socket 加入指定的多点广播地址
            socket.joinGroup(broadcastAddress);
            // 设置本 MulticastSocket 发送的数据报被回送到自身
            socket.setLoopbackMode(false);
            // 初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
            outPacket = new DatagramPacket(new byte[0]
                  , 0 , broadcastAddress , BROADCAST_PORT);
            // 启动两个读取网络数据的线程
            new ReadBroad().start();
            Thread.sleep(1);
            new ReadSingle().start();
      }
      // 广播消息的工具方法
      public void broadCast(String msg)
      {
            try
            {
                  // 将 msg 字符串转换成字节数组
                  byte[] buff = msg.getBytes();
                  // 设置发送用的 DatagramPacket 里的字节数据
                  outPacket.setData(buff);
                  // 发送数据报
                  socket.send(outPacket);
            }
            // 捕获异常
            catch (IOException ex)
            {
                  ex.printStackTrace();
                  if (socket != null)
                  {
                        // 关闭该 Socket 对象
                        socket.close();
                  }
                  JOptionPane.showMessageDialog(null
                        , "发送信息异常，请确认 30000 端口空闲，且网络连接正常！"
                        , "网络异常", JOptionPane.ERROR_MESSAGE);
                  System.exit(1);
            }
      }
      // 定义向单独用户发送消息的方法
      public void sendSingle(String msg , SocketAddress dest)
      {
            try
            {
                  // 将 msg 字符串转换成字节数组
                  byte[] buff = msg.getBytes();
                  DatagramPacket packet = new DatagramPacket(buff
                        , buff.length , dest);
                  singleSocket.send(packet);
            }
            // 捕获异常
            catch (IOException ex)
            {
                  ex.printStackTrace();
                  if (singleSocket != null)
                  {
                        // 关闭该 Socket 对象
                        singleSocket.close();
                  }
                  JOptionPane.showMessageDialog(null
                        , "发送信息异常，请确认 30001 端口空闲，且网络连接正常！"
                        , "网络异常", JOptionPane.ERROR_MESSAGE);
                  System.exit(1);
            }
      }
      // 不断地从 DatagramSocket 中读取数据的线程
      class ReadSingle extends Thread
      {
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
        // 定义接收网络数据的字节数组
        byte[] singleBuff = new byte[DATA_LEN];
        private DatagramPacket singlePacket =
              new DatagramPacket(singleBuff , singleBuff.length);
        public void run()
        {
              while (true)
              {
                    try
                    {
                          // 读取 Socket 中的数据
                          singleSocket.receive(singlePacket);
                          // 处理读到的信息
                          lanTalk.processMsg(singlePacket , true);
                    }
                    // 捕获异常
                    catch (IOException ex)
                    {
                          ex.printStackTrace();
                          if (singleSocket != null)
                          {
                                // 关闭该 Socket 对象
                                singleSocket.close();
                          }
                          JOptionPane.showMessageDialog(null
                                , "接收信息异常，请确认 30001 端口空闲，且网络连接正常！"
                                , "网络异常", JOptionPane.ERROR_MESSAGE);
                          System.exit(1);
                    }
              }
        }
  }
  // 持续读取 MulticastSocket 的线程
  class ReadBroad extends Thread
  {
        public void run()
        {
              while (true)
              {
                    try
                    {
                          // 读取 Socket 中的数据
                          socket.receive(inPacket);
                          // 打印输出从 Socket 中读取的内容
                          String msg = new String(inBuff , 0
                                , inPacket.getLength());
                          // 读到的内容是在线信息
                          if (msg.startsWith(YeekuProtocol.PRESENCE)
                                && msg.endsWith(YeekuProtocol.PRESENCE))
                          {
                                String userMsg = msg.substring(2
                                      , msg.length() - 2);
                                String[] userInfo = userMsg.split(YeekuProtocol
                                      .SPLITTER);
                                UserInfo user = new UserInfo(userInfo[1]
                                      , userInfo[0] , inPacket.getSocketAddress(), 0);
                                // 控制是否需要添加该用户的旗标
                                boolean addFlag = true;
                                ArrayList<Integer> delList = new ArrayList<>();
                                // 遍历系统中已有的所有用户，该循环必须循环完成
                                for (int i = 1 ; i < lanTalk.getUserNum() ; i++ )
                                {
                                      UserInfo current = lanTalk.getUser(i);
                                      // 将所有用户失去联系的次数加 1
                                      current.setLost(current.getLost() + 1);
                                      // 如果该信息由指定用户发送
                                      if (current.equals(user))
                                      {
                                       current.setLost(0);
                                       // 设置该用户无须添加
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
                                    addFlag = false;
                                    }
                                    if (current.getLost() > 2)
                                    {
                                    delList.add(i);
                                    }
                              }
                              // 删除 delList 中的所有索引对应的用户
                              for (int i = 0; i < delList.size() ; i++)
                              {
                                    lanTalk.removeUser(delList.get(i));
                              }
                              if (addFlag)
                              {
                                    // 添加新用户
                                    lanTalk.addUser(user);
                              }
                        }
                        // 读到的内容是公聊信息
                        else
                        {
                              // 处理读到的信息
                              lanTalk.processMsg(inPacket , false);
                        }
                  }
                  // 捕获异常
                  catch (IOException ex)
                  {
                        ex.printStackTrace();
                        if (socket != null)
                        {
                              // 关闭该 Socket 对象
                              socket.close();
                        }
                        JOptionPane.showMessageDialog(null
                              , "接收信息异常，请确认 30000 端口空闲，且网络连接正常！"
                              , "网络异常", JOptionPane.ERROR_MESSAGE);
                        System.exit(1);
                  }
            }
      }
  }
}
```

该类主要实现底层的网络通信功能，在该类中提供了一个 broadCast()方法，该方法使用 Multicast Socket 将指定字符串广播到所有客户端；还提供了 sendSingle()方法，该方法使用 DatagramSocket 将指定字符串发送到指定 SocketAddress，如程序中前两行粗体字代码所示。除此之外，该类还提供了两个内部线程类：ReadSingle 和 ReadBroad，这两个线程类采用循环不断地读取 DatagramSocket 和 Multicast Socket 中的数据，如果读到的信息是广播来的在线信息，则保持该用户在线；如果读到的是用户的聊天信息，则直接将该信息显示出来。

在该类中用到了本程序的一个主类：LanTalk，该类使用 DefaultListModel 来维护用户列表，该类里的每个列表项就是一个 UserInfo。该类还提供了一个 ImageCellRenderer，该类用于将列表项绘制出用户图标和用户名字。

程序清单：codes\17\17.4\LanTalk\LanTalk.java

```
public class LanTalk extends JFrame
{
    private DefaultListModel<UserInfo> listModel
          = new DefaultListModel<>();
    // 定义一个 JList 对象
    private JList<UserInfo> friendsList = new JList<>(listModel);
    // 定义一个用于格式化日期的格式器
    private DateFormat formatter = DateFormat.getDateTimeInstance();
    public LanTalk()
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
        {
            super("局域网聊天");
            // 设置该 JList 使用 ImageCellRenderer 作为单元格绘制器
            friendsList.setCellRenderer(new ImageCellRenderer());
            listModel.addElement(new UserInfo("all" , "所有人"
                  , null , -2000));
            friendsList.addMouseListener(new ChangeMusicListener());
            add(new JScrollPane(friendsList));
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setBounds(2, 2, 160 , 600);
        }
        // 根据地址来查询用户
        public UserInfo getUserBySocketAddress(SocketAddress address)
        {
            for (int i = 1 ; i < getUserNum() ; i++)
            {
                UserInfo user = getUser(i);
                if (user.getAddress() != null
                      && user.getAddress().equals(address))
                {
                      return user;
                }
            }
            return null;
        }
        // ------下面四个方法是对 ListModel 的包装------
        // 向用户列表中添加用户
        public void addUser(UserInfo user)
        {
            listModel.addElement(user);
        }
        // 从用户列表中删除用户
        public void removeUser(int pos)
        {
            listModel.removeElementAt(pos);
        }
        // 获取该聊天窗口的用户数量
        public int getUserNum()
        {
            return listModel.size();
        }
        // 获取指定位置的用户
        public UserInfo getUser(int pos)
        {
            return listModel.elementAt(pos);
        }
        // 实现 JList 上的鼠标双击事件监听器
        class ChangeMusicListener extends MouseAdapter
        {
            public void mouseClicked(MouseEvent e)
            {
                // 如果鼠标的击键次数大于 2
                if (e.getClickCount() >= 2)
                {
                    // 取出鼠标双击时选中的列表项
                    UserInfo user = (UserInfo)friendsList.getSelectedValue();
                    // 如果该列表项对应用户的交谈窗口为 null
                    if (user.getChatFrame() == null)
                    {
                        // 为该用户创建一个交谈窗口，并让该用户引用该窗口
                        user.setChatFrame(new ChatFrame(null , user));
                    }
                    // 如果该用户的窗口没有显示，则让该用户的窗口显示出来
                    if (!user.getChatFrame().isShowing())
                    {
                        user.getChatFrame().setVisible(true);
                    }
                }
            }
        }
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
        /**
         * 处理网络数据报，该方法将根据聊天信息得到聊天者
         * 并将信息显示在聊天对话框中
         * @param packet 需要处理的数据报
         * @param single 该信息是否为私聊信息
         */
        public void processMsg(DatagramPacket packet , boolean single)
        {
            // 获取发送该数据报的 SocketAddress
            InetSocketAddress srcAddress = (InetSocketAddress)
                packet.getSocketAddress();
            // 如果是私聊信息，则该 Packet 获取的是 DatagramSocket 的地址
            // 将端口号减 1 才是对应的 MulticastSocket 的地址
            if (single)
            {
                srcAddress = new InetSocketAddress(srcAddress.getHostName()
                    , srcAddress.getPort() - 1);
            }
            UserInfo srcUser = getUserBySocketAddress(srcAddress);
            if (srcUser != null)
            {
                // 确定消息将要显示到哪个用户对应的窗口中
                UserInfo alertUser = single ? srcUser : getUser(0);
                // 如果该用户对应的窗口为空，则显示该窗口
                if (alertUser.getChatFrame() == null)
                {
                    alertUser.setChatFrame(new ChatFrame(null , alertUser));
                }
                // 定义添加的提示信息
                String tipMsg = single ? "对您说：" : "对大家说：";
                // 显示提示信息
                alertUser.getChatFrame().addString(srcUser.getName()
                    + tipMsg + ".....................("
                    + formatter.format(new Date()) + ")\n"
                    + new String(packet.getData() , 0 , packet.getLength())
                    + "\n");
                if (!alertUser.getChatFrame().isShowing())
                {
                    alertUser.getChatFrame().setVisible(true);
                }
            }
        }
        // 主方法，程序的入口
        public static void main(String[] args)
        {
            LanTalk lanTalk = new LanTalk();
            new LoginFrame(lanTalk , "请输入用户名、头像后登录");
        }
    }
    // 定义用于改变 JList 列表项外观的类
    class ImageCellRenderer extends JPanel
        implements ListCellRenderer<UserInfo>
    {
        private ImageIcon icon;
        private String name;
        // 定义绘制单元格时的背景色
        private Color background;
        // 定义绘制单元格时的前景色
        private Color foreground;
        @Override
        public Component getListCellRendererComponent(JList list
            , UserInfo userInfo , int index
            , boolean isSelected , boolean cellHasFocus)
        {
            // 设置图标
            icon = new ImageIcon("ico/" + userInfo.getIcon() + ".gif");
            name = userInfo.getName();
            // 设置背景色、前景色
            background = isSelected ? list.getSelectionBackground()
                : list.getBackground();
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            foreground = isSelected ? list.getSelectionForeground()
                  : list.getForeground();
            // 返回该 JPanel 对象作为单元格绘制器
            return this;
      }
      // 重写 paintComponent 方法，改变 JPanel 的外观
      public void paintComponent(Graphics g)
      {
            int imageWidth = icon.getImage().getWidth(null);
            int imageHeight = icon.getImage().getHeight(null);
            g.setColor(background);
            g.fillRect(0, 0, getWidth(), getHeight());
            g.setColor(foreground);
            // 绘制好友图标
            g.drawImage(icon.getImage() , getWidth() / 2 - imageWidth / 2
                  , 10 , null);
            g.setFont(new Font("SansSerif" , Font.BOLD , 18));
            // 绘制好友用户名
            g.drawString(name, getWidth() / 2 - name.length() * 10
                  , imageHeight + 30 );
      }
      // 通过该方法来设置该 ImageCellRenderer 的最佳大小
      public Dimension getPreferredSize()
      {
            return new Dimension(60, 80);
      }
}
```

上面类中提供的 addUser()和 removeUser()方法暴露给通信类 ComUtil 使用，用于向用户列表中添加、删除用户。除此之外，该类还提供了一个 processMsg()方法，该方法用于处理网络中读取的数据报，将数据报中的内容取出，并显示在特定的窗口中。

提示：

上面讲解的只是本程序的关键类，本程序还涉及 YeekuProtocol、ChatFrame、LoginFrame 等类，由于篇幅关系，此处不再给出这些类的源代码，读者可以参考 codes\17\17.4\LanTalk 路径下的源代码。