# 第6讲 UDP套接字程序设计

**教学与实践目的：**学会使用UDP套接字来实现网络应用程序设计。

**UDP特点:**

（1）UDP有别于TCP，有自己独立的套接字(IP+PORT)，它们的端口号不冲突。和TCP编程相比，UDP在使用前不需要进行连接，没有流的概念。如果说TCP协议通信与电话通信类似，那么UDP通信就与邮件通信类似：不需要实时连接，只需要目的地址；

（2）UDP 通信前**不**需要建立连接，只要知道地址（ip地址和端口号）就可以给对方发送信息；

（3）基于用户数据报文（包）读写；

（4）UDP通信一般用于线路质量好的环境，如局域网内，如果是互联网，往往应用于对数据完整性不是过于苛刻的场合，例如语音传送等。

几个关键的Java类：DatagramSocket，DatagramPacket，MulticastSocket。

## 一、程序设计第一步：创建UDPClient.java程序

UDPClient.java源程序见[附录](#_1._UDPClient.java源程序)。

UDP通信，一般都是客户端必须先向被动等待连接的服务器端发UDP包，客户端不发UDP包，服务器端就根本不知道客户端的地址和端口号。一个典型的UDP客户端主要执行以下三步：

（1）创建一个DatagramSocket实例，可以选择对本地地址和端口号进行设置，但一般不用指定，程序将自动选择本地地址和可用的端口；

（2）使用DatagramSocket类来发送和接收DatagramPacket类的实例，进行通信；

（3）通信完成后，使用DatagramSocket类的close()销毁该套接字。

与Socket类不同，使用DatagramSocket实例在创建的时候并不需要指定目的地址，这也是TCP协议和UDP协议最大的不同点之一。

### 1. UDP套接字类: DatagramSocket

不像TCP通信有客户套接字Socket和服务器套接字ServerSocket两种，而UDP套接字只有一种DatagramSocket，不区分客户套接字和服务器套接字。

**（1）UDP套接字创建：**

**DatagramSocket** **datagramSocket = new DatagramSocket();**

正如前面所述，不需要指定本地的地址和端口号。

**（2）UDP套接字的几个重要方法：**

发送网络数据用：**datagramSocket.send(packet)；** //发送一个数据包到由IP和端口号指定的地址。

接收网络数据用：**datagramSocket.receive(packet)；**//接收一个数据包，没有数据，则程序会在这里**阻塞**。

（其中packet属于 DatagramPacket报文类的一个实例对象，用户数据封装在packet中）

指定超时：**datagramSocket.setSoTimeout(n)**，n是个整数，表示毫秒数。用来指定上面的receive()方法的最长阻塞时间。超过这个时长还receive方法还没得到响应，就会抛出InterruptedIOException异常，如果你的客户端设置为一个send，然后等待一个receive，可以用这个方法来设置超时，避免程序无限等待；但如果你的客户端类似TCP的设计方式，开启一个新线程来专门receive信息，那就不要用这个方法，否则在等待的过程中就超时报错了。

### 2. UDP数据报文类：DatagramPacket

TCP发送数据是基于字节流的，而UDP发送数据是基于报文DatagramPacket，网络中传递的UDP数据都要封装在这种自包含（self-contained）的报文中。

上面的UDP套接字创建的过程，会发现根本没有指定远程通信方的IP和端口，那么其send方法是发给谁呢，关键就在send方法的参数：DatagramPacket类型。数据报文除了要传输的信息外，每个数据报文实例中还附加了地址和端口信息，其具体含义取决了该数据报文是被发送还是被接收。若是要发送的数据报文，其实例中的地址则指明了目的地址和端口号；如果是要接收的数据报文，其实例中的地址则指明了所接收信息的源地址。

**（1）UDP数据报文的创建：**

DatagramPacket有多个构造方法，对于**发送信息**，需要明确远程的地址信息，采用下面的构造方法创建：

**DatagramPacket(byte[ ] data, int length, InetAddress remoteAddr, int remotePort)**

对于**接收信息**，其对象实例的创建则不用指定地址信息：

**DatagramPacket(byte[ ] data, int length)**

这样UDP套接字的send方法就能把报文发送到目的地址。上面的int length表示要读取的数据长度。byte[ ] data就是用于存储报文数据的字节数组缓存。

（2）**UDP数据报文**DatagramPacket**的几个重要方法：**

**InetAddress getAddress( )**：如果是发送的报文，返回的是目标主机的地址信息；如果是要接收的报文，返回的是发送该数据报文的主机地址信息；

**int getPort( )**：如果是发送的报文，返回的是目标主机的端口；如果是要接收的报文，返回的是发送该数据报文的主机端口；

注意：上面这两个方法主要是服务端常用，服务端可以通过这两个方法获知客户端的地址信息。

**byte[ ] getData( ):** 从报文中取数据，返回与数据报文相关联的字节数组；

## 二、程序设计第二步：创建UDPClientFX.java客户端窗体程序

用户界面如图6.1所示：

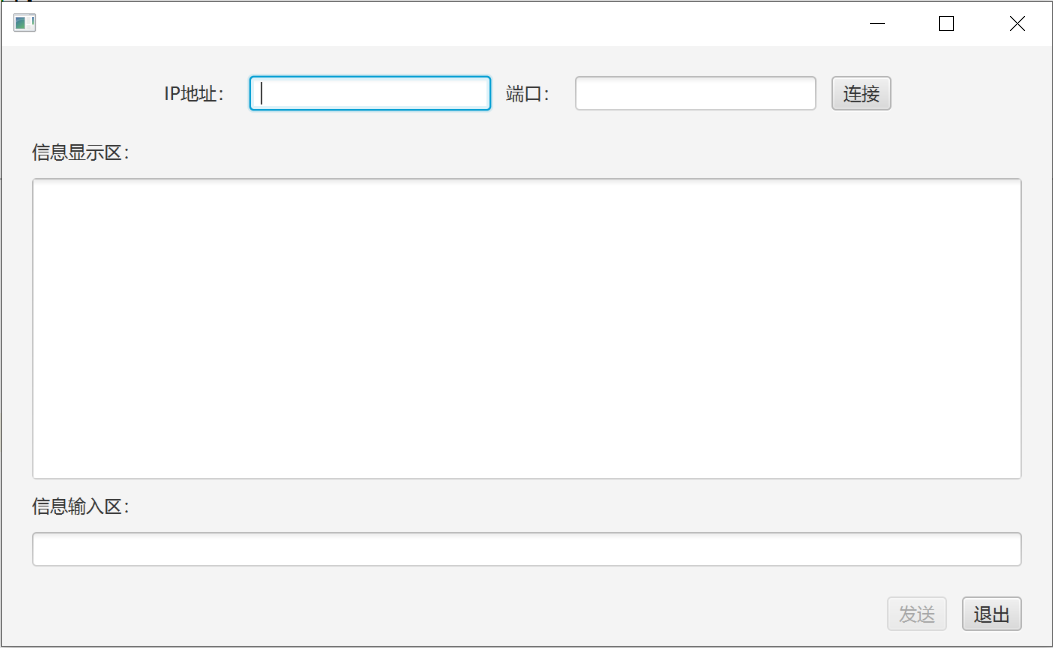


图6.1 UDP客户端界面

（1）在“连接”按钮中实例化UDPClient对象，并启动接收信息的“新线程”；

（2）在“发送”按钮中设置发送信息的代码；

（3）在“退出”按钮中设置退出程序运行的代码；

（4）用客户端向测试服务器172.16.229.253:6006发送信息，测试客户端是否工作正常。

## 三、程序设计第三步：创建UDPServer.java程序

类似TCP服务器，UDP服务器的工作也是建立一个通信终端，并被动等待客户端发起连接。但由于UDP是无连接的，并没有TCP中建立连接的那一步。UDP通信通过客户端的数据报文初始化。典型的UDP服务器需要执行以下三步：

（1）创建一个UDP套接字DatagramSocket实例，但和客户端不同，需要指定一个本地端口（端口号范围在1024-65535之间选取），这样客户端才能发起初次通信）：

**DatagramSocket datagramSocket = new DatagramSocket( port );**

此时，服务器就准备好了从**任何**客户端接收数据报文。UDP服务器默认是为所有的客户端使用同一个套接字，（这和TCP不同，TCP服务器为每个成功返回的accept方法都创建一个新的套接字；UDP套接字就像个邮箱，所有人的邮件都往这里发送）；

（2）使用UDP套接字DatagramSocket实例的receive方法来接收一个UDP报文DatagramPacket实例。当receive方法返回的时候，数据报文就包含了客户端的地址信息，这样服务器就知道该消息来自哪里，回复消息该发送到什么地方；

（3）使用套接字DatagramSocket类的send和receive方法来发送和接收报文DatagramPacket的实例，进行通信。

**注意：**因为服务端经常需要循环多次调用receive方法接收消息，如果是用同一个报文实例来接收，那么就需要在下一个receive方法之前，先调用报文实例的s**etLength(缓存数组.length)**方法。原因是报文实例包含一个内部消息的长度值，每一次receive收到报文，这个长度值就会修改为接收到的消息的实际长度值。例如这一次收到的消息是10字节，那么下一次receive到消息，超出10字节的内容都会被无提示的丢弃。所以就需要调用setLength方法来将内部消息长度值重置为构造报文实例时指定的缓存字节数组的长度。

### UDPServer功能要求：

根据UDPClient.java程序，创建对应的UDPServer服务器程序。完成如下功能：用自己的UDPClientFX客户端发送信息到自己的UDPServer服务器，UDPServer接收信息并能自动将修改后的信息返回，具体来说，就是收到客户端的任意原始信息后，返回如下内容：你学号和姓名的信息头、时间信息及你服务器接收到的原始信息，各信息用“&”符号隔开成四部分，注意顺序一定不要错，如：

20180000001&程旭元& Wed Sep 02 10:43:12 CST 2020&原始信息

上面的时间信息是用new Date( ).toString( )来生成。

**注意：**与TCP不同，小负荷的UDP服务器往往不是多线程的。由于UDP是同一个套接字对应多个客户端，对于UDP服务端，可以不需要采用多线程方式（除非为了准备响应需要做大量耗费时间的工作，这种情况确实需要多线程，避免其他客户排队等待），而是直接使用一种顺序迭代方法就可以：服务器等待客户端请求，然后读取请求，处理请求，发回响应，接着循环往复。

## 四、课堂计分

本次课堂计分需要你的UDP客户端向教师计分服务器发送信息，计分服务器根据你的信息，再访问你的UDP服务器，成功解析你服务器返回给教师计分服务器的信息后，才会记录课堂得分，具体步骤如下：

（1）关闭你机器上的防火墙，将你的UDPServer服务器在指定的端口号运行（建议端口号大于1024，例如8008）；

（2）用你的客户端给计分服务器172.16.229.253:6868发送如下指定信息：我的UDP服务器已经准备好，端口号是&\*\*\*\*&

注意，信息一定要严格按照要求，\*\*\*\*表示你的UDP服务器的端口号，两个&是信息间隔符；

（3）教师端的服务器会自动访问你的UDP服务器，如果你的服务器工作正常，反馈信息符合要求，就会提示成功信息，后台记录课堂成绩，如果长时间显示正在访问你的UDP服务器……而没有后续提示信息，那么检查你的程序，是否端口号等信息有误。

以上步骤可参看图6.2。



172.16.229.253

图6.2 计分过程

## 五、扩展练习：组播程序设计

组播是指在一群用户范围内发送和接收信息，该信息具有共享性。UDP具有组播功能，而TCP不具有。

组播地址范围为224.0.0.0--- 239.255.255.255。组播地址号唯一标示一群用户（一定网络范围内，仅限于局域网络内或有些自治系统内支持组播）。但有很多组播地址默认已经被占用，建议在225.0.0.0到238.255.255.255之间随机选择一个组播地址使用。（默认组播不能跨子网，除非设置了TTL值，且有配置组播路由器。另外同一个子网内如果也出现部分主机组播无效，可能是vmware的虚拟网卡影响，可先临时禁用这些命名为Vmnet\*的虚拟网卡，并关闭防火墙）

只要大家加入同一个组播地址，就能全体收取信息。在Java中，使用组播套接字MulticastSocket来组播数据，其是DatagramSocket的一个子类，使用方式也与DatagramSocket十分相似∶将数据放在DatagramPacket对象中，然后通过MulticastSocket收发DatagramPacket对象。

**组播套接字类MulticastSocket及其几个重要的方法：**

（1）路由器、交换机一般只转发和终端机一致IP地址和广播地址数据，终端机如何知道要接收组内信息？要先声明加入或退出某一组播组，其方法是：

MulticastSocket ms = new MulticastSocket(8900);

ms.joinGroup(groupIP); 该方法表示加入groupIP组， groupIP是InetAddress类型的组播地址。其作用是：告知自己的网络层该IP地址的包要收；转告上联的路由器这样的IP地址包要转发。

ms.leaveGroup(groupIP); 该方法表示退出groupIP组

（2）组内接收和发送信息的方法同UDP单播，也是以下两个方法，packet为DatagramPacket类型：

ms.send(packet);

ms.receive(packet);

（3）独立完成组播程序Multicast.java（源代码见[附录](#_2._Multicast.java源程序)）和窗体界面MulticastFX.java,组播套接字为： 225.0.0.1:8900，在组内发言要求以“From ip地址 学号 姓名：”为信息头。其效果如图6.3所示，要求每位同学都能看到组内其他同学的留言。

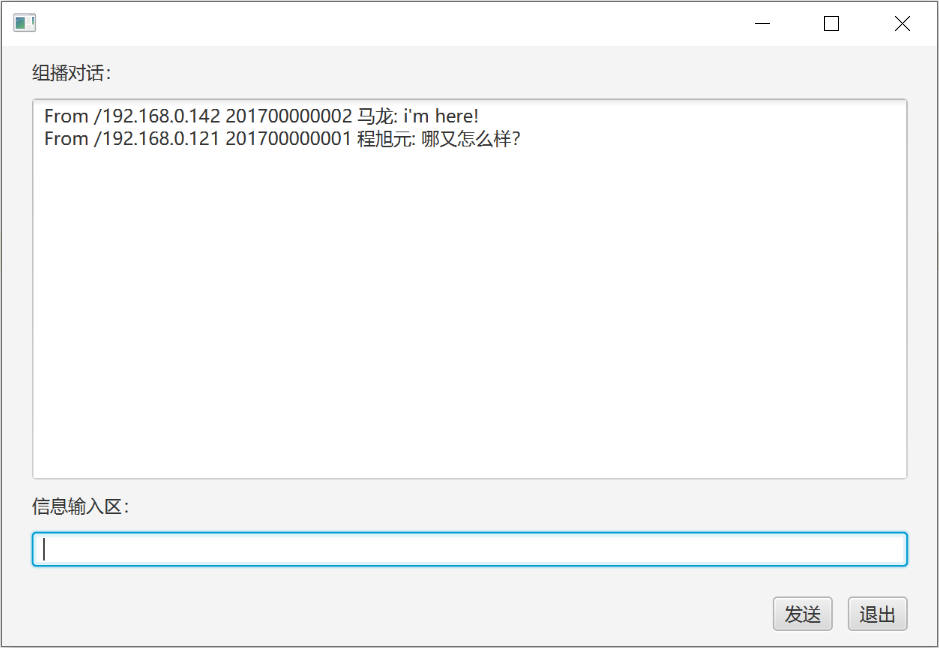


图6.3 组播对话界面

## 六、扩展练习二：UDP局域网聊天程序

与TCP不同，UDP其实不真正区分服务端和客户端，一个程序其实可以身兼二职，尝试写一个不区分服务端和客户端UDP局域网聊天程序UDPChatFX.java。在同一局域网的机器运行该程序，可以互相发消息及发送广播消息。程序应该提供一个下拉组合框来显示在线的用户ip地址，选中地址即可以给该用户发送消息；如果下拉组合框的内容为空，则给所有用户发送广播消息。发送广播消息可以简单的给广播地址"255.255.255.255"发送报文来实现。

关于在线IP地址列表的获得方法，可以给广播地址发送一个约定的探测信息，收到该特定探测信息的用户就回发一个约定的信息报文，这样就可以从该报文中取出ip地址，加入到下拉组合框中。

## 附录

### 1. UDPClient.java源程序

package chapter06.client;  
  
import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
  
public class UDPClient {  
 private int remotePort;  
 private InetAddress remoteIP;  
 private DatagramSocket socket;*//UDP套接字  
  
 //用于接收数据的报文字节数组缓存最大容量，字节为单位* private static final int *MAX\_PACKET\_SIZE* = 512;  
  
 public UDPClient(String remoteIP, String remotePort) throws IOException {  
 this.remoteIP = InetAddress.*getByName*(remoteIP);  
 this.remotePort = Integer.*parseInt*(remotePort);  
 *//创建一个UDP套接字，系统随机选定一个未使用的UDP端口绑定* socket = new DatagramSocket();  
 *//设置接收数据超时  
// socket.setSoTimeout(30000);* }  
  
 *//定义一个数据的发送方法* public void send(String msg) {  
 try {  
 *//将待发送的字符串转为字节数组* byte[] outData = msg.getBytes("utf-8");  
 *//构建用于发送的数据报文，构造方法中传入远程通信方（服务器)的ip地址和端口* DatagramPacket outPacket = new DatagramPacket(outData, outData.length, remoteIP, remotePort);  
 *//给UDPServer发送数据报* socket.send(outPacket);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//定义数据接收方法* public String receive() {  
 String msg;  
 *//先准备一个空数据报文* DatagramPacket inPacket = new DatagramPacket(  
 new byte[*MAX\_PACKET\_SIZE*], *MAX\_PACKET\_SIZE*);  
 try {  
 *//读取报文，阻塞语句，有数据就装包在inPacket报文中，以装完或装满为止。* socket.receive(inPacket);  
 *//将接收到的字节数组转为对应的字符串* msg = new String(inPacket.getData(),  
 0,inPacket.getLength(),"utf-8");  
 } catch (IOException e) {  
 e.printStackTrace();  
 msg = null;  
 }  
 return msg;  
 }  
}

### 2. Multicast.java源程序

package chapter06.multicast;  
  
import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.InetAddress;  
import java.net.MulticastSocket;  
  
public class Multicast {  
 InetAddress groupIP;  
 int port = 8900;  
 MulticastSocket ms = null; *//组播套接字* byte[] inBuff = new byte[1024];  
 byte[] outBuff = new byte[1024];  
  
 public Multicast() throws IOException {  
  
 groupIP = InetAddress.*getByName*("225.0.0.1");  
 *//开启一个组播端口(UDP端口)* ms = new MulticastSocket(port);  
 *//告诉网卡这样的IP地址数据包要接收* ms.joinGroup(groupIP);  
 }  
  
 public void send(String msg) {  
 try {  
 outBuff = ("201700000001 程旭元: " + msg).getBytes("utf-8");  
 DatagramPacket outPacket = new DatagramPacket(outBuff, outBuff.length,  
 groupIP, port);  
 ms.send(outPacket);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public String receive() {  
 try {  
 DatagramPacket inPacket = new DatagramPacket(inBuff, inBuff.length);  
 ms.receive(inPacket);  
 String msg = new String(inPacket.getData(), 0, inPacket.getLength(), "utf-8");  
 return "From " + inPacket.getAddress() + " " + msg;  
 } catch (IOException e) {  
 return null;  
 }  
 }  
  
 public void close() {  
 try {  
 ms.leaveGroup(groupIP);  
 ms.close();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}