使用DatagramSocket发送、接收数据(Socket之UDP套接字)

From jiangxinyu

使用DatagramSocket发送、接收数据（1）

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。

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。

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);
```

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。

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。

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。

使用DatagramSocket发送、接收数据（2）

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。

```
1 //程序清单：codes\17\17.4\UdpServer.java
2 public class UdpServer
3 {
4   public static final int PORT = 30000;
5   // 定义每个数据报的最大大小为4KB
6   private static final int DATA_LEN = 4096;
7   // 定义接收网络数据的字节数组
8   byte[] inBuff = new byte[DATA_LEN];
9   // 以指定字节数组创建准备接收数据的DatagramPacket对象
10  private DatagramPacket inPacket =
11    new DatagramPacket(inBuff , inBuff.length);
12  // 定义一个用于发送的DatagramPacket对象
13  private DatagramPacket outPacket;
14  // 定义一个字符串数组，服务器端发送该数组的元素
15  String[] books = new String[]
16    {
17    "疯狂Java讲义",
18    "轻量级Java EE企业应用实战",
19    "疯狂Android讲义",
20    "疯狂Ajax讲义"
21    };
22  public void init()throws IOException
23    {
24    try(
25      // 创建DatagramSocket对象
26      DatagramSocket socket = new DatagramSocket(PORT))
27        {
28      // 采用循环接收数据
29      for (int i = 0; i < 1000 ; i++ )
30            {
31        // 读取Socket中的数据，读到的数据放入inPacket封装的数组里
32              socket.receive(inPacket);
33        // 判断inPacket.getData()和inBuff是否是同一个数组
34        System.out.println(inBuff == inPacket.getData());
35        // 将接收到的内容转换成字符串后输出
36        System.out.println(new String(inBuff
37          , 0 , inPacket.getLength()));
38        // 从字符串数组中取出一个元素作为发送数据
39        byte[] sendData = books[i % 4].getBytes();
40        // 以指定的字节数组作为发送数据，以刚接收到的DatagramPacket的
41        // 源SocketAddress作为目标SocketAddress创建DatagramPacket
42        outPacket = new DatagramPacket(sendData
43                    , sendData.length , inPacket.getSocketAddress());
44        // 发送数据
```

```
45                  socket.send(outPacket);
46              }
47          }
48      }
49    public static void main(String[] args)
50      throws IOException
51      {
52      new UdpServer().init();
53      }
54 }
```

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。

使用DatagramSocket发送、接收数据（3）

```
1 //程序清单：codes\17\17.4\UdpClient.java
2
3 public class UdpClient
4 {
5   // 定义发送数据报的目的地
6   public static final int DEST_PORT = 30000;
7   public static final String DEST_IP = "127.0.0.1";
8   // 定义每个数据报的最大大小为4KB
9   private static final int DATA_LEN = 4096;
10  // 定义接收网络数据的字节数组
11  byte[] inBuff = new byte[DATA_LEN];
12  // 以指定的字节数组创建准备接收数据的DatagramPacket对象
13  private DatagramPacket inPacket =
14    new DatagramPacket(inBuff , inBuff.length);
15  // 定义一个用于发送的DatagramPacket对象
16  private DatagramPacket outPacket = null;
17  public void init()throws IOException
18    {
19    try(
20      // 创建一个客户端DatagramSocket，使用随机端口
21      DatagramSocket socket = new DatagramSocket())
22        {
23      // 初始化发送用的DatagramSocket，它包含一个长度为0的字节数组
24      outPacket = new DatagramPacket(new byte[0] , 0
25                , InetAddress.getByName(DEST_IP) , DEST_PORT);
26      // 创建键盘输入流
27      Scanner scan = new Scanner(System.in);
28      // 不断地读取键盘输入
29      while(scan.hasNextLine())
```

```
30              {
31          // 将键盘输入的一行字符串转换成字节数组
32          byte[] buff = scan.nextLine().getBytes();
33          // 设置发送用的DatagramPacket中的字节数据
34                  outPacket.setData(buff);
35          // 发送数据报
36                  socket.send(outPacket);
37          // 读取Socket中的数据，读到的数据放在inPacket所封装的字节数组中
38                  socket.receive(inPacket);
39          System.out.println(new String(inBuff , 0
40                      , inPacket.getLength()));
41              }
42          }
43      }
44    public static void main(String[] args)
45      throws IOException
46      {
47      new UdpClient().init();
48      }
49  }
```

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。

Socket之UDP套接字

UDP套接字：UDP套接字的使用是通过DatagramPacket类和DatagramSocket类，客户端和服务器端都是用DatagramPacket类来接收数据，使用DatagramSocket类来发送数据。

UDP客户端：也是主要执行三个步骤。

1.创建DatagramSocket实例；

2.使用DatagramSocket类的send()和receive()方法发送和接收DatagramPacket实例；

3.最后使用DatagramSocket类的close()方法销毁该套接字。

下面是例子，它主要执行三个步骤，

1.向服务器发送信息；

2.在receive()方法上最多阻塞等待3秒钟，在超时前若没有收到响应，则重发请求（最多重发5次）；

3.关闭客户端。

```
1 //UDPEchoClientTimeout.java
2
3 import java.net.DatagramSocket;
4 import java.net.DatagramPacket;
5 import java.net.InetAddress;
6 import java.io.IOException;
7 import java.io.InterruptedIOException;
8
9 public class UDPEchoClientTimeout {
10
11   private static final int TIMEOUT = 3000;  // 设置超时为3秒
12   private static final int MAXTRIES = 5;    // 最大重发次数5次
13
14   public static void main(String[] args) throws IOException {
15
16     if ((args.length < 2) || (args.length > 3)) { // Test for correct # of args
17       throw new IllegalArgumentException("Parameter(s): <Server> <Word> [<Port>]");
18         }
19     InetAddress serverAddress = InetAddress.getByName(args[0]); // 服务器地址
20     // Convert the argument String to bytes using the default encoding
21     //发送的信息
22     byte[] bytesToSend = args[1].getBytes();
23
24     int servPort = (args.length == 3) ? Integer.parseInt(args[2]) : 7;
25
26     DatagramSocket socket = new DatagramSocket();
27
28     socket.setSoTimeout(TIMEOUT); // 设置阻塞时间
29
30     DatagramPacket sendPacket = new DatagramPacket(bytesToSend, // 相当于将发送的信息打包
31             bytesToSend.length, serverAddress, servPort);
32
33     DatagramPacket receivePacket =                    // 相当于空的接收包
34       new DatagramPacket(new byte[bytesToSend.length], bytesToSend.length);
35
36     int tries = 0;    // Packets may be lost, so we have to keep trying
37     boolean receivedResponse = false;
38     do {
39       socket.send(sendPacket);        // 发送信息
40       try {
41         socket.receive(receivePacket); // 接收信息
42
43         if (!receivePacket.getAddress().equals(serverAddress)) {// Check source
```

```
44          throw new IOException("Received packet from an unknown source");
45                }
46          receivedResponse = true;
47        } catch (InterruptedIOException e) { // 当receive不到信息或者receive时间超过3秒时，就向服务器重发请求
48          tries += 1;
49          System.out.println("Timed out, " + (MAXTRIES - tries) + " more tries...");
50              }
51      } while ((!receivedResponse) && (tries < MAXTRIES));
52
53      if (receivedResponse) {
54        System.out.println("Received: " + new String(receivePacket.getData()));
55      } else {
56        System.out.println("No response -- giving up.");
57          }
58          socket.close();
59      }
60 }
```

例子只是简单的向指定的服务器发送信息，并将发送的信息由服务器返回给指定客户端。

UDP服务器端：典型的UDP服务器要执行三个步骤，

1.创建一个指定了本地端口的DatagramSocket实例；

2.使用DatagramSocket的receive()方法接收一个来自客户端的DatagramPacket实例，而这个DatagramPacket实例在客户端创建时就包含了客户端的地址，这样我们就知道回复信息要发送到哪里了；

3.使用DatagramSocket类的send()和receive()方法来发送和接收DatagramPacket实例。

下面是例子

```
1 //UDPEchoServer.java
2 import java.io.IOException;
3 import java.net.DatagramPacket;
4 import java.net.DatagramSocket;
5
6 public class UDPEchoServer {
7
8   private static final int ECHOMAX = 255; // 发送或接收的信息最大字节数
9
10  public static void main(String[] args) throws IOException {
11
12    if (args.length != 1) { // Test for correct argument list
13      throw new IllegalArgumentException("Parameter(s): <Port>");
14          }
15
16    int servPort = Integer.parseInt(args[0]);
17
18    DatagramSocket socket = new DatagramSocket(servPort);
19    DatagramPacket packet = new DatagramPacket(new byte[ECHOMAX], ECHOMAX);
```

```
20
21      while (true) { // 不断接收来自客户端的信息及作出相应的相应
22        socket.receive(packet); // Receive packet from client
23        System.out.println("Handling client at " + packet.getAddress().getHostAddress() + " on port " +
packet.getPort());
24        socket.send(packet); // 将客户端发送来的信息返回给客户端
25              packet.setLength(ECHOMAX);
26       // 重置packet的内部长度，因为处理了接收到的信息后，数据包的内部长度将被
27       //设置为刚处理过的信息的长度，而这个长度可能比缓冲区的原始长度还要短，
28       //如果不重置，而且接收到的新信息长于这个内部长度，则超出长度的部分将会被截断，所以这点必须注意到。
29           }
30      /* NOT REACHED */
31      }
32 }
```

例子只是简单地将客户端发送过来的信息再回复给客户端，服务器端会不断地receive来自客户端的信息，如果receive不到任何客户端请求，则将会进入阻塞状态，直到receive到有客户端请求位置。

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。

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。

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：