007.使用MulticastSocket实现多点广播

基于UDP协议的网络编程

TCP 协议之外，还有一个 UDP 协议，UDP 协议是用户数据报协议的简称，也用于网络数据的传输。

虽然 UDP 协议是一种不太可靠的协议，但有时在需要较快地接收数据并且可以忍受较小错误的情况下，

UDP 就会表现出更大的优势。

UDP通信两个核心类：DatagramSocket类和DatagramPacket类

下面是在 Java 中使用 UDP 协议发送数据的步骤。

- 使用 DatagramSocket() 创建一个数据包套接字。
- 使用 DatagramPacket() 创建要发送的数据包。
- 使用 DatagramSocket 类的 send() 方法发送数据包。

接收 UDP 数据包的步骤如下：

- 使用 DatagramSocket 创建数据包套接字，并将其绑定到指定的端口。
- 使用 DatagramPacket 创建字节数组来接收数据包。
- 使用 DatagramSocket 类的 receive() 方法接收 UDP 包。

DatagramPacket 类

java.net 包中的 DatagramPacket 类用来表示数据报包，数据报包用来实现无连接包投递服务。

每条报文仅根据该包中包含的信息从一台机器路由到另一台机器。

从一台机器发送到另一台机器的多个包可能选择不同的路由，也可能按不同的顺序到达。

DatagramPacket的构造方法

构造方法	说明
DatagramPacket(byte[] buf,int length)	构造 DatagramPacket，用来接收长度为 length 的数据包。
DatagramPacket(byte[] buf,int offset, int length)	构造 DatagramPacket，用来接收长度为 length 的包，在缓冲区中指定了偏移量。
DatagramPacket(byte[] buf,int length, InetAddress address,int port)	构造 DatagramPacket，用来将长度为 length 的包发送到指定主机上的指定端口。
DatagramPacket(byte[] buf,int length, SocketAddress address)	构造数据报包，用来将长度为 length 的包发送到指定主机上的指定端口。
DatagramPacket(byte[] buf,int offset, int length,InetAddress address,int port)	构造 DatagramPacket，用来将长度为 length 偏移量为 offset 的包发送到指定主机上的指定端口。
DatagramPacket(byte[] buf,int offset, int length,SocketAddress address)	构造数据报包，用来将长度为 length、偏移量为 offset 的包发送到指定主机上的指定端口。

DatagramPacket的常用方法

方法	说明
InetAddress getAddress()	返回某台机器的 IP 地址，此数据报将要发往该机器或者从该机器接收。
byte[] getData()	返回数据缓冲区。
int getLength()	返回将要发送或者接收的数据的长度。
int getOffset()	返回将要发送或者接收的数据的偏移量。
int getPort()	返回某台远程主机的端口号，此数据报将要发往该主机或者从该主机接收。
getSocketAddress()	获取要将此包发送或者发出此数据报的远程主机的 SocketAddress（通常为 IP地址+端口号）。
void setAddress(InetAddress addr)	设置要将此数据报发往的目的机器的IP地址。
void setData(byte[] buf)	为此包设置数据缓冲区。
void setData(byte[] buf,int offset, int length)	为此包设置数据缓冲区。
void setLength(int length)	为此包设置长度。
void setPort(int port)	设置要将此数据报发往的远程主机的端口号。
void setSocketAddress(SocketAddress address)	设置要将此数据报发往的远程主机的 SocketAddress（通常为 IP地址+端口号）。

DatagramSocket 类

DatagramSocket 类用于表示发送和接收数据报包的套接字。

数据报包套接字是包投递服务的发送或接收点。

每个在数据报包套接字上发送或接收的包都是单独编址和路由的。

从一台机器发送到另一台机器的多个包可能选择不同的路由，也可能按不同的顺序到达。

DatagramSocket 类的常用构造方法

构造方法	说明
DatagramSocket()	构造数据报包套接字并将其绑定到本地主机上任何可用的端口。
DatagramSocket(int port)	创建数据报包套接字并将其绑定到本地主机上的指定端口。
DatagramSocket(int portJnetAddress addr)	创建数据报包套接字，将其绑定到指定的本地地址。
DatagramSocket(SocketAddress bindaddr)	创建数据报包套接字，将其绑定到指定的本地套接字地址。

DatagramSocket 类的常用方法

方法	说明
void bind(SocketAddress addr)	将此 DatagramSocket 绑定到特定的地址和端口。
void close()	关闭此数据报包套接字。
void connect(InetAddress address,int port)	将套接字连接到此套接字的远程地址。
void connect(SocketAddress addr)	将此套接子连接到远程套接子地址（IP地址+端口号）。
void disconnect()	断开套接字的连接。
InetAddress getInetAddress()	返回此套接字连接的地址。
InetAddress getLocalAddress()	获取套接字绑定的本地地址。
int getLocalPort()	返回此套接字绑定的本地主机上的端口号。
int getPort()	返回此套接字的端口。

编写 UDP 例子程序，客户端与服务器端之间交互数据：

```
public class UdbClient {
    public static void main(String[] args) {
        //套接字
        try(DatagramSocket socket = new DatagramSocket();) {
             //不传参数就是随机使用本地端口
            //定义两个用来收和发的数据的集装箱
            //在收数据的集装箱上面指定的长度，是每次收数据的报包大小上限,4K
            byte[] inbuffer = new byte[4096];
            DatagramPacket inPacket = new DatagramPacket(inbuffer,inbuffer.length);
            //预定义一个发数据的集装箱
            DatagramPacket outPacket = new DatagramPacket(new byte[0],0,
                    new InetSocketAddress("127.0.0.1",30000));

            //发送数据，是键盘上输入的数据
            Scanner sc = new Scanner(System.in);
            while(sc.hasNextLine()){
                byte[] datas = sc.nextLine().getBytes();
                outPacket.setData(datas);
                socket.send(outPacket);
                //服务器端收到我发送的数据，服务器应该可以沿着去的路，给我回应一个信息过来
                socket.receive(inPacket);
                System.out.println("服务器发来的信息："
                        + new String(inbuffer,0,inPacket.getLength()));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
public class UdpServer {
    public static void main(String[] args) {
        //首先拿套接字
        try(DatagramSocket socket =
                    new DatagramSocket(new InetSocketAddress("127.0.0.1",30000));) {
            //定义接收和发送数据的集装箱
            byte[] inbuffer = new byte[4096];
            DatagramPacket inPacket = new DatagramPacket(inbuffer,inbuffer.length);
            //定义发数据的集装箱
            DatagramPacket outPacket;
            while(socket.isClosed()==false){
                //收数据
                socket.receive(inPacket);
                System.out.println(new String(inbuffer,0,inPacket.getLength()));
                SocketAddress clientAddr = inPacket.getSocketAddress();
                //原路把一些数据返回去
                byte[] sendData = "服务器收到".getBytes();
                outPacket = new DatagramPacket(sendData,sendData.length,clientAddr);
                socket.send(outPacket);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

NIO对UDP协议的支持：

NIO同样支持udp协议，不过相关的方法调用是有区别的

java的udp端口类是DatagramSocket

因此对应的nio channel类就是DatagramChannel

由于udp是一个无连接的协议，因此服务器端和客户端的代码基本相同。

实际上服务器和客户端之间并没有太大区分。所以不存在ServerDatagramChannel这种玩意了，

服务器端和客户端都是创建一个DatagramChannel。然后bind一个端口，注册Selector之后就可以打开监听了。

注意和SocketChannel的区别有以下几点

接收数据：SocketChannel的方法是

```
1. ByteBuffer buffer = ByteBuffer.allocate(500);
2. int readCount = socketChannel.read(buffer);
```

虽然DatagramChannel也有read(ByteBuffer buffer)这个方法，但是直接调用只会抛出异常。

接收数据包的正确姿势如下：

```
1. ByteBuffer buffer = ByteBuffer.allocate(500);
2. InetSocketAddress =
(InetSocketAddress)datagramChannel.receive(buffer);
```

此方法直接返回一个SocketAddress对象，包含了数据来源的地址和端口，在反馈数据的时候就有大用处了。
于此同时，datagramChannel.getRemoteAddress()方法自然是不可能有正确结果的，没有连接是不会有Remote Address的

同理，发送数据包的姿势也变了
原来的方式是：

```
1. socketChannel.write(ByteBuffer.wrap("test data".getBytes()));
```

由于udp的无连接属性，此方法会让系统一头雾水的，我们需要加上发送的目标地址：

```
1. datagramChannel.send(ByteBuffer.wrap("test data".getBytes()),new
   InetAddressSocket("localhost",12345));
```

链接特定机器地址
上面说不能用read和write方法，你说我非要用呢？可以！
DatagramChannel实际上是可以指定到网络中的特定地址的。

```
1. channel.connect(new InetSocketAddress("localhost"), 80));
```

当连接上后，可以向使用传统的通道那样调用read()和Writer()方法。区别是数据的读写情况得不到保证。

写一个小demo：

```
public class UdpNIOServer {
    //定义两个缓冲器，发数据 和 收数据
    private ByteBuffer inbuffer = ByteBuffer.allocate(1024);
    private ByteBuffer outbuffer = ByteBuffer.allocate(1024);

    public void init() {
        //TCP：ServerSocketChannel
        //UDP：DataProgramChannel
        try {
            DatagramChannel datagramChannel = DatagramChannel.open();
            datagramChannel.configureBlocking(false);
            //主要接收数据，要绑定一个固定端口
            datagramChannel.bind(new InetSocketAddress("127.0.0.1", 40000));
            //选择器
            Selector selector = Selector.open();
            datagramChannel.register(selector, SelectionKey.OP_READ);
            //通过循环不断地接收和发送数据
            while (true) {
                int count = selector.select(5000);
                if (count == 0) continue;
                Iterator<SelectionKey> it = selector.selectedKeys().iterator();
                while (it.hasNext()) {
                    SelectionKey key = it.next();
                    if (key.isReadable()) {
                        handlerRead(key);
                    }
                    selector.selectedKeys().remove(key);
                }
```

```
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    public void handlerRead(SelectionKey key) {
        try {
            DatagramChannel datagramChannel = (DatagramChannel) key.channel();
            inbuffer.clear();
            InetSocketAddress sendAddr = (InetSocketAddress)
datagramChannel.receive(inbuffer);
            inbuffer.flip();
            System.out.println("客户端发来的信息："
                    + StandardCharsets.UTF_8.decode(inbuffer).toString());
            //回应
            handlerWrite(datagramChannel, sendAddr);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void handlerWrite(DatagramChannel datagramChannel, InetSocketAddress sendAddr) {
        try {
            String content = "服务器收到！";
            outbuffer.clear();
            outbuffer.put(content.getBytes("UTF-8"));
            outbuffer.flip();
            datagramChannel.send(outbuffer, new
InetSocketAddress(sendAddr.getHostName(),sendAddr.getPort()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    public static void main(String[] args) {
        UdpNIOServer server = new UdpNIOServer();
        server.init();
    }

}

public class UdpNIOClient {
    public static void main(String[] args) {
        //定义两个缓冲器，发数据 和 收数据
        ByteBuffer inbuffer = ByteBuffer.allocate(1024);
        ByteBuffer outbuffer = ByteBuffer.allocate(1024);
        try {
            Scanner scanner = new Scanner(System.in);
            DatagramChannel datagramChannel = DatagramChannel.open();
            datagramChannel.configureBlocking(false);
            while (scanner.hasNextLine()) {
                String content = scanner.nextLine();

                //发送数据
                outbuffer.clear();
                outbuffer.put(content.getBytes("UTF-8"));
                outbuffer.flip();
                datagramChannel.send(outbuffer,new InetSocketAddress("127.0.0.1", 40000));
```

```
                inbuffer.clear();
                datagramChannel.receive(inbuffer);
                inbuffer.flip();
                System.out.println("服务器发送来的信息："
                        + StandardCharsets.UTF_8.decode(inbuffer).toString());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

使用MulticastSocket实现多点广播

DatagramSocket只允许数据报发送给指定的一个目标客户端

而MulticastSocket可以将数据报以广播的方式发送到多个客户端。

组播地址：称为组播组的一组主机所共享的地址。

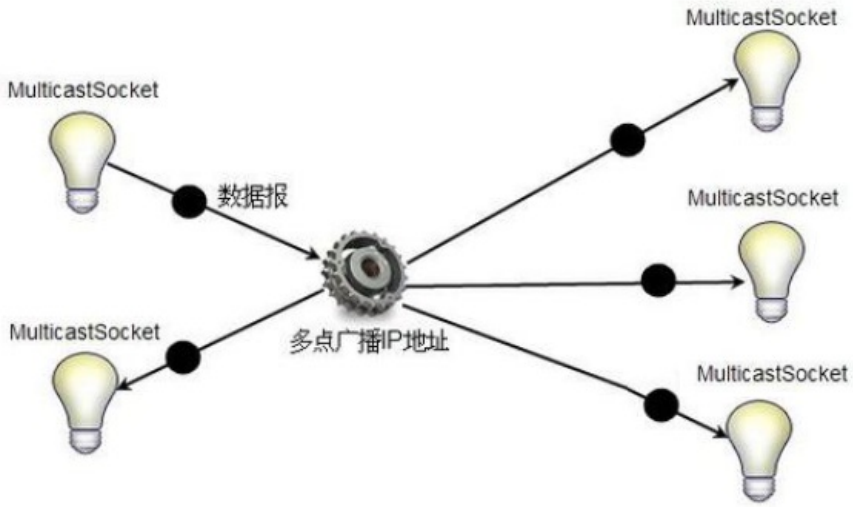
组播地址的范围在224.0.0.0--- 239.255.255.255之间（都为D类地址 1110开头）。

备注：如果现在有三台机器A、B、C，三台机器IP地址都不一样，A\B为server监听广播消息，

C为客户端发送广播消息，将A、B两台机器的MulticastSocket对象绑定在组播地址中的其中一个，

然后C客户端发送消息的组播地址一致，则A、B就能够接收C发送的消息。

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MulitcastSocket是DatagramSocket的一个子类，当要发送一个数据报时，

可以使用随机端口创建一个MulticastSocket,也可以在指定端口创建MulticastSocket。

MulticastSocket提供了如下3个构造器

1、MulticastSocket():使用本机默认地址、随机端口来创建MulticastSocket对象

2、MulticastSocket(int portNumber)使用本机默认地址、指定端口来创建对象

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

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

1、joinGroup(InetAddress multicastAddr)：将该MulticastSocket加入指定的多点广播地址。

2、leaveGroup(InetAddress multicastAddr)：让该MulticastSocket离开指定的多点广播地址。

应用程序只将数据报包发送给组播地址，路由器将确保包被发送到改组播组中的所有主机。

注意：如果MulticastSocket仅用于发送信息则使用默认地址和随机端口即可，
但是如果用来接收信息，则必须要指定端口，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。

最后玩儿一个综合案例：结合MulticastSocket和DatagramSocket实现一个局域网即时通讯工具。