UDP的通信方式

1. **UDP：非连接的、速度快、但不安全。**
2. UDP通信用到的类：

（1）**DatagramPacket类**：用于封装数据，即所说的**数据报**。数据报数据的长度不要超过8192K。

（2）**DatagramSocket类**：用于**发送和接收数据报**。

1. **发送数据报**：

**利用DatagramPacket类和DatagramSocket类：**

**首先建立一个DatagramSocket对象，然后把数据、目标地址、目标端口号封装成DatagramPacket对象。利用send方法把数据报发送出去，最后释放资源即可。**

1. **接收数据报**：

首先是利用**DatagramSocket类的带参数int port的构造方法，创建一个数据报套接字对象，端口号必须是待接收的数据报的端口号；然后创建一个空的DatagramPacket对象，**

**其次利用receive（DatagramPacket p）方法把收到数据封装到数据报对象p中。receive方法可能会堵塞，直到收到数据报为止。最后释放资源即可。**

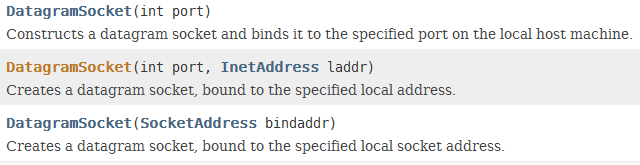


1. **DatagramSocket类**：位于java.net包中，直接继承于java.lang.Object类，用于UDP数据通信。



**（1）DatagramSocket类的**构造方法：有多个，其中有个空参数的构造方法。





**（2）成员方法：**

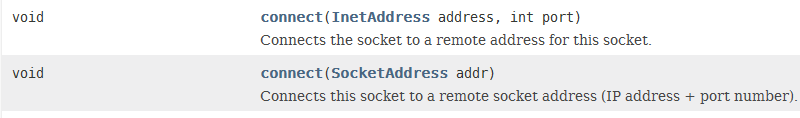
①利用send（DatagramPacket p）方法把数据报发送出去。



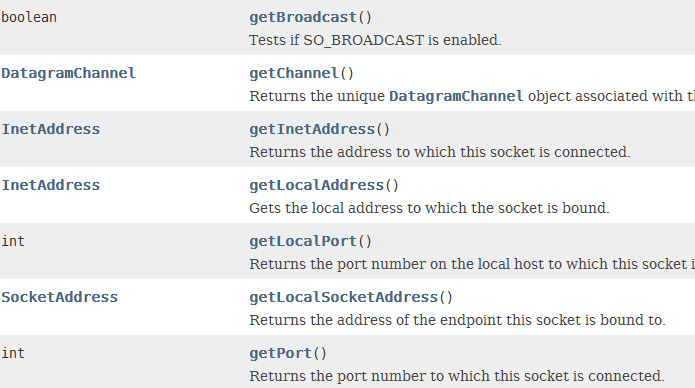
②利用**receive方法**接收**数据报**。



③connect方法：



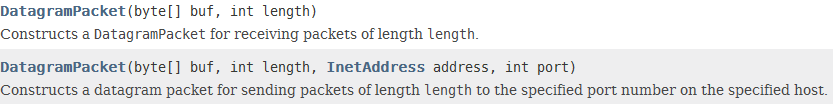
④getXxx方法:



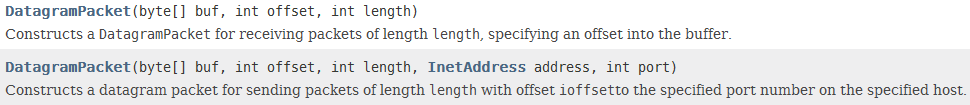
1. **DatagramPacket类介绍：**

**存在于java.net包中，直接继承于java.lang.Object类，是final的，不允许继承。**

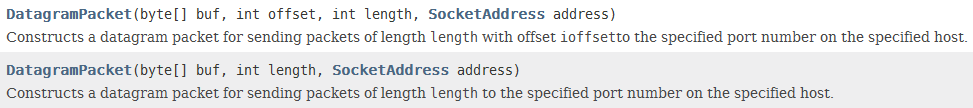
1. **构造方法：**
   1. **指定数据（字节数组）及其长度，可以指定数据目标地址和目标端口号。**



* + 1. 指定字节数组的起始和长度；

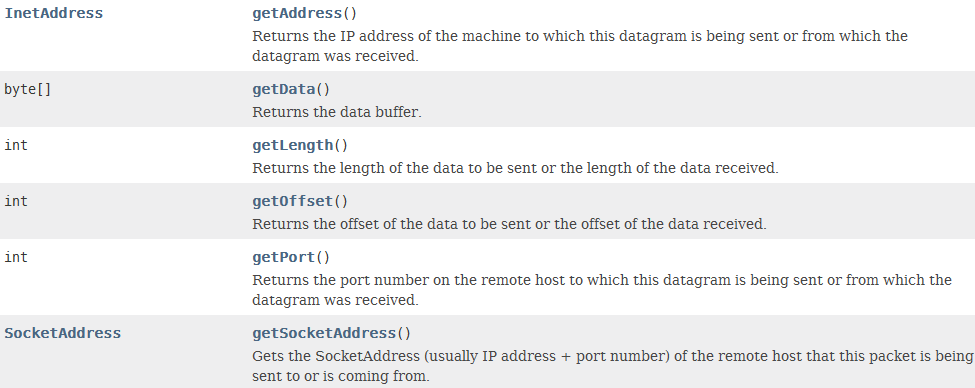


* + 1. 把网络地址和端口号封装成**SocketAddress对象**。

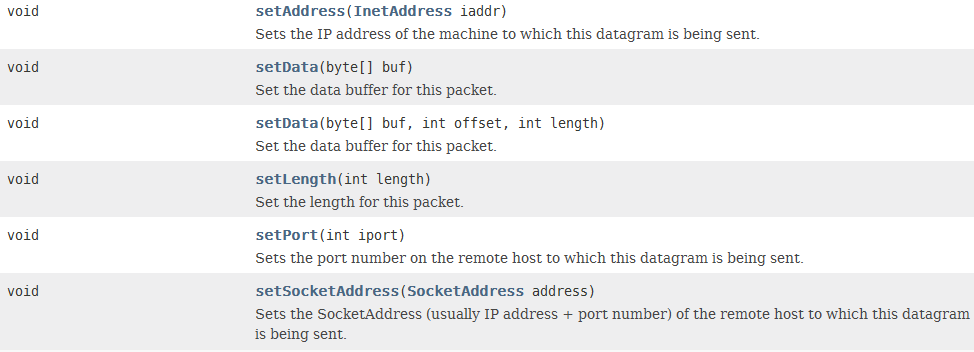


1. 成员方法：
2. getXxx方法

获取的数据的**目标地址和端口号**。



1. setXxx方法：



1. 既然UDP通信是面向非连接的，所以客户端和服务器端不需要建立连接，即可相互发送数据。

客户端：

1. 创建客户端：即创建一个DatagramSocket对象(指定一个端口号)；
2. 准备数据，并打包：封装成数据报，需要指明目的地址和端口号。DatagramPacket + 目的地址及端口号；
3. 发送；
4. 释放资源。

服务器端：

1. 创建服务器端：创建一个DatagramSocket对象(指定一个端口号)；
2. 准备容器接收：DatagramPacket；
3. 包接收数据；
4. 分析数据；
5. 释放资源。
6. C/S示例：

客户端：

public class MyClient {

public static void main(String[] args) throws IOException {

// 1. 创建客户端，由于UDP不需要建立连接，所以创建客户端时，不需要指定服务器的地址。

DatagramSocket client = new DatagramSocket(6666);

// 2.对需要发送的数据进行封装，封装成数据报对象，数据报需要指明目的地址和端口号。

byte[] data = "我爱Java北京邮电大学信息与通信工程学院".getBytes();

DatagramPacket packet = new DatagramPacket(data, data.length,new InetSocketAddress("localhost", 8888));

// 3. 直接发送

client.send(packet );

// 4. 释放资源

client.close();

}

}

服务器：

**public** **class** MyServer {

**public** **static** **void** main(String[] args) **throws** IOException {

// 1. 创建服务器，并指明端口号

DatagramSocket server = **new** DatagramSocket(8888);

// 2. 准备好接受容器，并将接受容器封装成数据报

**byte**[] container = **new** **byte**[1024];

DatagramPacket packet = **new** DatagramPacket(container, container.length);

// 3. 接受数据）阻塞式方法

server.receive(packet);

// 4。分析数据报

**byte**[] result = packet.getData();

String str = **new** String(result,0,result.length);

System.***out***.println(str);

// 5.释放资源

server.close();

}

}

分析： 由于UDP传输是非面向连接的，所以在创建客户端服务器不需要建立连接，不需要指明目的地址，只需要在数据报中封装上目的地址和端口号即可。服务器的receive方法是阻塞式方法，直到接收到数据才会继续往下进行。需要首先打开服务器，再利用客户顿发送数据，否则数据就会丢失。但是客户端不会知道数据是否被接收到，因为是非面向连接的。

1. 上面的例子是发送了字符串，方法是先把字符串转换成了字节数据，然后直接对字节数据进行封装发送；服务器端是对字节数据进行处理，在转换成字符串。

现在如果是想要发送一个double类型的数据，怎么办？

思路:先把double类型数据转换成字节数据，然后封装发送，服务器端，接收字节数据，然后在解析为double类型即可。下面主要是添加了两个convert方法，实现**double—>byte[]，和byte[]🡪double**。

当然重点是如何进行转换的问题？

在这里通过字节数组流和数据流进行完成：

ByteArrayOutputStream—>DataOutputStream

ByteArrayInputStream—>DataInputStream

示例：

客户端：

**public** **class** MyClient2 {

**public** **static** **void** main(String[] args) **throws** IOException {

DatagramSocket client = **new** DatagramSocket(6666);

**double** d = 3.1415926;

**byte**[] data = *convert*(d);

DatagramPacket packet = **new** DatagramPacket(data, data.length,**new** InetSocketAddress("localhost", 8888));

client.send(packet );

client.close();

}

**private** **static** **byte**[] convert(**double** data) **throws** IOException {

ByteArrayOutputStream bos = **new** ByteArrayOutputStream();

DataOutputStream dos = **new** DataOutputStream(bos);

dos.writeDouble(data);

bos.flush();

**byte**[] result = bos.toByteArray();

**return** result;

}

}

服务器端：

**public** **class** MyServer2 {

**public** **static** **void** main(String[] args) **throws** IOException {

DatagramSocket server = **new** DatagramSocket(8888);

**byte**[] container = **new** **byte**[1024];

DatagramPacket packet = **new** DatagramPacket(container, container.length);

server.receive(packet);

**byte**[] data = packet.getData();

**double** result = *convert*(data);

System.***out***.println(result);

server.close();

}

**private** **static** **double** convert(**byte**[] data) **throws** IOException {

ByteArrayInputStream bis = **new** ByteArrayInputStream(data);

DataInputStream dis = **new** DataInputStream(bis);

**double** result = dis.readDouble();

**return** result;

}

}