网络编程技术

`java`

一.网络通信基础

基本知识包括网络通信中的通信协议，IP地址，端口号，TCP协议，UDP协议以及网络程序设计的基本框架。

1.基本概念

计算机网络：

> 指通过各种通信设备连接起来的，支持特定网络通信协议的，许许多多的计算机通信或计算机系统的集合

网络通信：

> 指网络中的计算机通过网络互相传递信息

通信协议：

> 是网络通信的基础，是网络中计算机之间进行通信时共同遵守的规则。不同的通信协议用不同的方法解决不同类型的通信问题。

为了实现网络上不同机器之间的通信，必须知道对方主机的地址和端口号

a.IP地址

IP地址是计算机网络中任意一台计算机地址的唯一标识符，知道了网络中某一主机的IP地址，就可以定位这一台计算机。IPv4 32位，IPv6 128位。

b.域名地址

这个就是主机的IP地址用纯数字表示的话不容易记忆，于是在IP地址上加上一个容易记忆的域名，网络中的域名解析器会将域名解析为IP地址。

c.端口号

主机上有多个进程，这些进程都可以和其他计算机进行通信，准确的说，计算机之间的通信其实就是进程与进程之间的通信。主机名和端口的组合能唯一确定网络通信的主体----进程。而端口是网络通信过程中不同进程的唯一标识。

2.通信协议

TCP和UDP是两种在网络通信中使用最多的通信协议。

a.TCP（Transfer Control Protocol)协议

是一种面向连接的，可以提供可靠传输的通信协议，意思就是TCP就是打电话，先要哦把电话拨通建立连接，然后可以肆无忌惮的谈话（传输信息），可靠性体现在对面发什么，我就接收什么。还是可以用电话类比。

b.UDP(User Datagram Protocol)协议

UDP是一种无连接的协议，传输的是一种独立的数据报。每个数据报都是独立的信息，包括完整的源地址和目的地址。UDP是有大小限制的，最大64kb.因此数据可以划分为多个数据报，每个数据报都有源地址和目的地址，假设数据报的目的地址都相同，因此每个数据报的路径可以不同。

c.两种协议的比较

1. UDP无连接，TCP必须要先建立连接
2. UDP传输数据大小有限制,TCP没有，一旦连接建立起来就按统一的格式随便跑了
3. UDP传输协议不可靠，数据报顺序不唯一，还可能丢失。TCP可靠

3.Java网络编程技术

提供了用于网络通信的java.net包。包含了多种用于各种标准网络通信协议的类和接口。大致通过以下三种方式实现网络程序设计：

a.URL编程技术

URL表示Internet上的某个资源的地址，支持ftp,http,file等多种协议，通过URL标识就可以获取远端计算机上的资源。这应该就是B/S模式。浏览器/服务器模式。通常我们抓取网页就是这个实现。

b.TCP编程技术

TCP是可靠的连接通信技术，主要使用套接字（Socket)机制。Socket是TCP/IP协议中的传输层接口，**是实现C/S模式的主要方式**

c.UDP编程技术

UDP是无连接的快速通信技术，数据报是一种在网络上传播的，独立的，自包含地址的格式化信息。**主要用于传输数据量大，非关键性数据**

> 有的时候真的要跳出来总结一下。。。狭义的理解通信，，。不是两个主机之间发短信，只要两者之间有信息流动就算是通信，即使有时候是单向的。单相思也是爱情，至少对于一方来说是，一个人的战斗。

二.URL程序设计

java.net中的URL类是对统一资源定位符（Uniform Resource Locator)的抽象，使用URL创建对象的程序称为客户端程序，一个URL对象存放着一个具体的资源引用，表明客户要访问的这个URL的资源，利用URL对象来访问URL中资源。

包含三部分信息：

1. 协议
2. 地址
3. 资源

1.URL

a.URL格式

```
传输协议名：//主机名：端口号/文件名#引用
```

例如合法的URL:

```

http://java.sun.com/index.html
http://java.sun.com/index.html#chapter1
http://192.168.0.7:7001
http://192.168.0.7:7001/port/index.html#myedu

```

b.URL类

常用的构造方法：

1. `URL(String spec)`:根据指定的字符串创建URL对象，如果字符串里指定了未知协议，则抛出MalfromedURLException异常
2. `URL（String protocol,String host,String file)`:根据参数构造URL
3. `URL（String protocol,String host,int port,String file)`：不解释了

常用的成员方法：

boolean	**equals**(Object obj) 比较此 URL 是否等于另一个对象。
String	**getAuthority**() 获取此 URL 的授权部分。
Object	**getContent**() 获取此 URL 的内容。
Object	**getContent**(Class[] classes) 获取此 URL 的内容。
int	**getDefaultPort**() 获取与此 URL 关联协议的默认端口号。
String	**getFile**() 获取此 URL 的文件名。
String	**getHost**() 获取此 URL 的主机名（如果适用）。
String	**getPath**() 获取此 URL 的路径部分。
int	**getPort**() 获取此 URL 的端口号。
String	**getProtocol**() 获取此 URL 的协议名称。
String	**getQuery**() 获取此 URL 的查询部分。
String	**getRef**() 获取此 URL 的锚点（也称为“引用”）。
String	**getUserInfo**() 获取此 URL 的 userInfo 部分。
int	**hashCode**() 创建一个适合哈希表索引的整数。
URLConnection	**openConnection**() 返回一个 URLConnection 对象，它表示到 URL 所引用的远程对象的连接。
URLConnection	**openConnection**(Proxy proxy) 与 openConnection() 类似，所不同是连接通过指定的代理建立；不支持代理方式的协议处理程序将忽略该代理参数并建立正常的连接。
InputStream	**openStream**() 打开到此 URL 的连接并返回一个用于从该连接读入的 InputStream。
boolean	**sameFile**(URL other) 比较两个 URL，不包括片段部分。
protected void	**set**(String protocol, String host, int port, String file, String ref) 设置 URL 的字段。
protected void	**set**(String protocol, String host, int port, String authority, String userInfo, String path, String query, String ref) 设置 URL 的指定的 8 个字段。
static void	**setURLStreamHandlerFactory**(URLStreamHandlerFactory fac) 设置应用程序的 URLStreamHandlerFactory。
String	**toExternalForm**() 构造此 URL 的字符串表示形式。
String	**toString**() 构造此 URL 的字符串表示形式。

实例：

```
1.
2.  /**
3.   * Created by XQF on 2016/12/3.
4.   */
```

```java
public class URLTest {
    public static final String URL_STRING="http://www.baidu.com/index.html";
    public static void main(String[] args) throws IOException {
        URL url= null;
        try {
            url = new URL(URL_STRING);
            System.out.println("协议：" + url.getProtocol());
            System.out.println("主机：" + url.getHost());
            System.out.println("端口：" + url.getPort());
            System.out.println("路径：" + url.getPath());
            System.out.println("文件：" + url.getFile());
            String string;
            StringBuffer sb = new StringBuffer();
            InputStreamReader is=new InputStreamReader(url.openStream());
            BufferedReader br=new BufferedReader(is);
            while((string=br.readLine())!=null){
                sb.append(string);
            }
            System.out.println(sb.toString());
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

c.URLConnection

主要用于应用程序和URL之间的连接，应用程序通过URLConnection可以获得URL对象的有关信息，是所有URL连接通信类的父类。该类的对象可以读写URL对象所代表的internet上的数据。

建立连接的步骤：

1. 在URL上调用openConnection()方法创建连接对象
2. 处理设置参数和一般请求属性
3. 使用connect方法建立到远程对象的实际连接
4. 远程对象变为可用，远程对象的头字段和内容变为可访问

常用字段

1. connected;表示连接状态，true表示建立了通信连接，false表示此连接对象尚未创建连接

2. url：表示此链接要在互联网上打开的远程对象

构造方法

1. URLConnection(URL url)

主要成员方法：

1. `Object getContent()`:获取此链接的内容
2. `String getContentEncoding()`：获取资源内容编码
3. `int getContentLength()`:获取资源内容长度
4. `String getContentType()`:资源内容类型
5. `URL getURL()`:返回URLConnection的url字段的值
6. `InputStream getInputStream()`:打开连接数据的输入流
7. `OutputStream getOutputStream()`:打开链接数据的输出流
8. `public void setConnectTimeout(int timeout)`:设置超时值

实例：
```java

/**
* Created by XQF on 2016/12/3.
*/
public class URLConnectionTest {
public static final String URL_STRING = "http://www.baidu.com/index.html";

```
public static void main(String[] args) {
    URL url = null;
    try {
        url = new URL(URL_STRING);
        URLConnection connection = url.openConnection();
        System.out.println("文件类型：" + connection.getContentType());
        System.out.println("文件长度：" + connection.getContentLength());
        System.out.println("文件内容：" + connection.getContent());
        System.out.println("--
----");
        ;
        String string;
        StringBuffer sb = new StringBuffer();
```
```

```
            BufferedReader br = new BufferedReader(new
    InputStreamReader(connection.getInputStream()));
            while ((string = br.readLine()) != null) {
                sb.append(string);
            }
            System.out.println(sb.toString());
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

}
]s
```

> 返回的文件内容并不是文件的真实内容。判断是否读完的条件为是否读到了“-1”，或者用“InputStreamReader.available()>0”来判断是否全部读完

## c.InetAddress

在互联网上表示一个主机的地址有两种方式，域名地址和IP地址，InetAddress类就是用来表示主机地址的。

常用成员方法：

```
1.  static InetAddress getByAddress(byte []addr)
2.  static InetAddress getByAddress(String host,byte[] addr):host可以是任何
    主机描述
3.  static InetAddress getByName(String host):给定主机名
4.  static InetAddress getLocalHost():返回本地主机
5.  String getHostName():返回主机名（域名）
6.  String getHostAddress():返回主机地址IP地址
7.  boolean isReachable(int timeout):测试是否可以到达
8.  String toString();主机地址的字符串表示（域名/IP）
9.  boolean isMulticastAddress();检测是否为多播地址
10. byte [] getAddress():返回InetAddress的原始IP
```

实例：
```

```
1.
2.  /**
3.   * Created by XQF on 2016/12/3.
4.   */
5.  public class InetAddressTest {
6.      public static void main(String[] args) {
7.          try {
8.              //获取给定域名的地址
9.              InetAddress inetAddress1 = null;
10.             inetAddress1 = InetAddress.getByName("www.baidu.com");
11.             System.out.println("----------------------------------------
    ------------");
12.
13.             System.out.println(inetAddress1.getHostName());//显示主机名
14.             System.out.println(inetAddress1.getHostAddress());//显示IP地
    址
15.             System.out.println(inetAddress1);//显示地址的字符串描述
16.             System.out.println("----------------------------------------
    ------------");
17.
18.             //获取本机的地址
19.             InetAddress inetAddress2 = InetAddress.getLocalHost();
20.             System.out.println(inetAddress2.getHostName());
21.             System.out.println(inetAddress2.getHostAddress());
22.             System.out.println(inetAddress2);
23.             System.out.println("----------------------------------------
    ------------");
24.
25.             //获取给定IP的主机的地址（72.5.124.55
26.             byte[] bytes = new byte[]{(byte) 72, (byte) 5, (byte) 124,
    (byte) 55};
27.             InetAddress inetAddress3 = InetAddress.getByAddress(bytes);
28.             InetAddress inetAddress4 = InetAddress.getByAddress("Sun 官
    方网站 21(java.sun.com)", bytes);
29.             System.out.println(inetAddress3);
30.             System.out.println(inetAddress4);
31.         } catch (UnknownHostException e) {
32.             e.printStackTrace();
33.         }
34.
35.
36.     }
37. }
```

三.TCP程序设计

TCP/IP套机字用于在主机和Internet之间建立可靠的，双向的，持续的，点对点的流式连接。

1.网络套接字

套接字是一个用于端点连接和数据交换的对象，一个套接字由IP地址和端口号唯一确定。网络的每一个端点都可以通过和连接绑定的套接字对象交换数据。在C/S模式下，按照套接字在网络中的作用不同，分为客户机端套接字和服务器端套接字

服务器端套接字SeverSocket

> 始终在监听是否有连接请求，有请求并且被接受，SeverSocket向客户机发回“接收”消息，两个socket之间的连接就建立了

客户端套接字Socket

> 建立一个和服务器的连接，需要知道服务器端提供的主机名和提供服务的端口号。

所以套接字关键部分就是建立连接，建立连接的过程为客户机发送请求，服务器监听到请求，接收请求后向客户机返回“接收”的消息，这个连接就建立好了，相当于是电话线扯好了大，接下来就是传送了

数据传送工作过程的具体操作：

1. 创建Socket对象
2. 打开连接到Socket对象的输入输出流
3. 按照一定的协议对Socket对象进行读和写操作
4. 关闭Socket对象（即关闭Socket对象绑定的连接）
5. 注意在这个过程中出现的各种异常，最最常见的就是在未启动服务器的情况下就打开客户机请求数据。

2.Socket

常用构造方法：

```
Socket(InetAddress address,int port)
Scoket(String host,int port)
//实例
Socket mySocket=new Socket("218.198.118.112",2010)
```

> 每一个端口提供一个特殊服务，只有正确给出端口，才能获得相应的服务。http->80,ftp->23.**0-1023是系统保留端口，自己设置端口号时，尽量大于1023的端口号**

常用的成员方法：

```

InetAddress getInetAddress():返回套接字连接的地址
InetAddress getLocalAddress():获取套接字绑定的本地地址
int getLocalPort();获取套接字绑定的本地端口
SocketAddress getLocalSocketAddress():返回套接字绑定的端点的地址，若没有绑定返回null
InputStream getInputStream():返回套接字的输入流
OutputStream getOutputStream():返回套接字的输出liu
int getPort():返回连接到的远程端口
boolean isBound():返回绑定状态
boolean isClosed():返回关闭状态
boolean isConnected():返回套接字的连接状态
void connect(SocketAddress endpoint,int timeout):链接到服务器并设置超时值
```

实例：

```

/**
 * Created by XQF on 2016/12/3.
 */
public class SocketTest {

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("218.198.118.103", 80);
            System.out.println("是否绑定连接：" + socket.isBound());
            System.out.println("本地端口:" + socket.getLocalPort());
            System.out.println("连接服务器的端口："+socket.getPort());
```

```
                System.out.println("连接服务器的地址:" + socket.getInetAddress
    ());
                System.out.println("连接远程服务的套接字：" + socket.getRemoteS
    ocketAddress());
                System.out.println("是否处于连接状态：" + socket.isConnected()
    );
                System.out.println("客户套接详情：" + socket.toString());
            } catch (Exception e) {
                System.out.println("服务器没有启动！");
            }
        }
    }
```

3.SocketSever

常用构造方法：

1. SeverSocket()
2. SeverSocket(int port)
3. SeverSocket(int port,int backlog)
4. SeverSocket(int port,int backlog,InetAddress bindAddr)

其中port可以为0，表示任何空闲端口
backlog指定了服务器所能支持的最长连接队列，如果队列满了则拒绝该连接
bindAddr是要将服务器绑定到的InetAddress，bindAddr参数可以在SeverSocket的多宿主主机上使用，SeverSocket仅接收对其地址之一的连接请求，如果bindAddr为null,则默认接受任何所有本地地址上的连接

```
SeverSocket severSocket=new SeverSocket(2010);
SeverSocket severSocket=new SeverSocket(2010,10);//端口为2010，队列最长为10
```

常用成员方法

Socket	**accept**() 侦听并接受到此套接字的连接。
void	**bind**(SocketAddress endpoint) 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。
void	**bind**(SocketAddress endpoint, int backlog) 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。
void	**close**() 关闭此套接字。

实例：

```
1.
2.  /**
3.   * Created by XQF on 2016/12/3.
4.   */
5.  public class serverSocketTest {
6.      public static void main(String[] args) {
7.          ServerSocket serverSocket = null;
8.          try {
9.              serverSocket = new ServerSocket(2010);
10.             System.out.println("服务器端口：" + serverSocket.getLocalPort
    ());
11.             System.out.println("服务器地址：" +
    serverSocket.getInetAddress());
12.             System.out.println("服务器套接字：" +
    serverSocket.getLocalSocketAddress());
13.             System.out.println("是否绑定连接：" +
    serverSocket.isBound());
14.             System.out.println("是否关闭：" + serverSocket.isClosed());
15.             System.out.println("服务器套接字详情：" + serverSocket.toStrin
    g());
16.         } catch (IOException e) {
17.             e.printStackTrace();
18.         }
19.     }
20. }
```

4.TCP编程实例

BusinessPrococal.java

```
1.
2.  /**
3.   * Created by XQF on 2016/12/3.
4.   */
5.  public interface BusinessProtocal {
6.      public static final int PAY_BILL = 1;
7.      public static final int ROAMING_SERVICE = 2;
8.
9.      public void paybill();
10.
```

```
    public void roamingService();
}
```

> **接口中不仅可以放方法，还可以放字符串常量**

MobileSever.java

```

/**
 * Created by XQF on 2016/12/3.
 */
public class MobileServer implements BusinessProtocal {
    private ServerSocket serverSocket;
    private DataInputStream in;
    private DataOutputStream out;
    private int serviceId;
    private Socket socket;
    private int fee;
    private String str;

    public MobileServer() {
        try {
            serverSocket = new ServerSocket(2010);
            while (true) {
                System.out.println("服务器准备就绪，等待客户请求。。。。");
                socket = serverSocket.accept();//堵塞状态，除非有客户呼叫
                in = new DataInputStream(socket.getInputStream());
                out = new DataOutputStream(socket.getOutputStream());
                while (true) {
                    try {
                        serviceId = in.readInt();//读取放入"线路里的信息"
                        switch (serviceId) {
                            case PAY_BILL:
                                paybill();//支付话费
                                break;
                            case ROAMING_SERVICE:
                                roamingService();//办理漫游
                            default:
                                break;
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
```

```
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("客户" + socket.getInetAddress().getHostName() + "业务办理完毕。已经离开了。。。。");
        }
    }

    @Override
    public void paybill() {
        try {
            fee = in.readInt();
            System.out.println("正在处理用户" + socket.getInetAddress().getHostName() + "预交饿的" + fee + "元话费请求！");
            System.out.println("交费处理完毕！");
            out.writeUTF("尊敬的客户，你已经成功预交了 " + fee + " 元话费");
            Thread.sleep(1000);
            out.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void roamingService() {
        try {
            str = in.readUTF();
            System.out.println("正在处理用户" + socket.getInetAddress().getHostName() + "将手机漫游到 " + str + "的请求");
            out.writeUTF("尊敬的用户，你的手机已经漫游到" + str + "了");
            Thread.sleep(1000);
            System.out.println("漫游处理完毕");
            out.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new MobileServer();
    }
}
```

> **考虑为什么要使用DataInputStream和DataOutputStream？因为我们的选择模式中有数字，根据数字来判断是选择哪一项服务，只有这个流有readInt()方法，而且还能输出输入字符串。。。总结一下整个过程，首先在服务器端建立一个ServerSocket,然后设置无限循环,调用accept()方法来获取可能的请求，，accept()方法的返回结果是一个Socket。用socket打开一个输入流，打开一个输出流。socket已经建立了一个链接，我们所有的传送数据操作都应该和socket有关。从输入流读取的数据就是客户机传来的请求或者数据，往输出流写入数据就是打算向客户机返回的信息。因此输入流中点出要什么信息，就往输出流中写什么信息。一旦有请求被接收就进入处理阶段。处理的时候，先从输入流中读出一个数字判断是办理什么业务，服务器办理业务，再将业务办理的结果返回到输出流中，返回给客户机。**

MobileClient.java

```
/**
 * Created by XQF on 2016/12/3.
 */
public class MobileClient  implements BusinessProtocal{
    private String string ;
    private DataInputStream in;
    private DataOutputStream out;
    private Socket socket;
    public MobileClient(){
        try {
            socket=new Socket("192.168.177.2",2010);
            in=new DataInputStream(socket.getInputStream());
            out=new DataOutputStream(socket.getOutputStream());
            paybill();
            Thread.sleep(500);//假装很耗时
            roamingService();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    @Override
    public void paybill() {
        try {
            out.writeInt(PAY_BILL);
            out.writeInt(200);
            out.flush();//
```

```
29.
30.
31.                 //得到的返回
32.                 string=in.readUTF();
33.                 System.out.println("来自服务员："+string);
34.             } catch (IOException e) {
35.                 e.printStackTrace();
36.             }
37.         }
38.
39.         @Override
40.         public void roamingService() {
41.             try {
42.                 out.writeInt(ROAMING_SERVICE);
43.                 out.writeUTF("香港");
44.                 out.flush();
45.     // 得到的返回
46.                 string=in.readUTF();
47.                 System.out.println("来自服务员："+string);
48.
49.             } catch (IOException e) {
50.                 e.printStackTrace();
51.             }
52.         }
53.         public static void main(String[] args) {
54.             new MobileClient();
55.         }
56.     }
```

> **要总结的就是客户机和服务器都是要一个输出流一个输入流的，这里的代码设计很是喜欢。还有就是见识到了接口的魅力。不仅充当了Config的角色（提供常量字符串），还增加了代码的紧凑性，看上去是环环相扣的，也是多态的魅力。接口的不同实现，使得代码变得如此的好懂，联系也强，佩服。还有就是输出流要刷新flush一下。再来解释一下DataInputStream，感觉也只是适合数据量少且类型多样的数据传送。服务器的监听一定要处于无限循环中**

四.UDP程序设计

UDP是一种无连接的网络通信机制，更像是邮件和发短信的通信方式

1.数据报通信

数据报是指起始点和目的地都能使用无连接网络服务的网络层的信息单元。UDP不可靠但是快速，因此在海量数据但是不精密的传送服务中占有重要地位，比如看个视频，视频在路上丢了几帧好像并没有什么大碍。
Java通过两个类实现UDP协议顶层的数据报，一个是**DatsgramPackets**,对象是数据容器，**DatagramSocket**是用来发送和接收DatagramPackets的套接字。采用UDP通信时，首先将要传输的数据打包，将打包好的数据传送给目的地。目的地接收数据包，然后查看数据包中的内容。

2.DatagramPacket

要发送或者接收数据包，需要用DatagramPacket将数据内容打包，即用DatagramPacket创建一个对象，称为数据包。

a.构造方法

构造方法摘要
DatagramPacket(byte[] buf, int length) 构造 DatagramPacket，用来接收长度为 length 的数据包。
DatagramPacket(byte[] buf, int length, InetAddress address, int port) 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号。
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 offset, int length, InetAddress address, int port) 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号。
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号。
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号。

b.常用方法

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是从该机器接收到的。
byte[]	**getData**() 返回数据缓冲区。
int	**getLength**() 返回将要发送或接收到的数据的长度。
int	**getOffset**() 返回将要发送或接收到的数据的偏移量。
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是从该主机接收到的。
SocketAddress	**getSocketAddress**() 获取要将此包发送到的或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 iaddr) 设置要将此数据报发往的那台机器的 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 iport) 设置要将此数据报发往的远程主机上的端口号。
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）。

3.DatagramSocket

DatagramSocket是用来发送和接收数据包的套接字，负责将数据包发送到目的地，或从目的地接收数据包。

a.构造方法

构造方法摘要	
	DatagramSocket() 构造数据报套接字并将其绑定到本地主机上任何可用的端口。
protected	**DatagramSocket**(DatagramSocketImpl impl) 创建带有指定 DatagramSocketImpl 的未绑定数据报套接字。
	DatagramSocket(int port) 创建数据报套接字并将其绑定到本地主机上的指定端口。
	DatagramSocket(int port, InetAddress laddr) 创建数据报套接字，将其绑定到指定的本地地址。
	DatagramSocket(SocketAddress bindaddr) 创建数据报套接字，将其绑定到指定的本地套接字地址。

b.常用方法

c.总结

简单实例（将“你好”打包并发送）：

```
1.  byte[] buff="你好".getBytes();
2.  InetAddress inetAddress=InetAddress.getByName("192.168.0.107");//I地址和域名都可以
3.  DatagramPocket dataPacket=new DatagramPocket(buff,buff.length,inetAddress,2018);
```

```
4.  DatagramSocket sendSocket=new DatagramSocket();
5.  sendSocket.send(dataPacket);
```

简单实例（取出外界发送给2018端口的数据包中的内容）：

```
1.
2.  byte []buff=new byte[8192];
3.  DatagramPocket receivePacket=new DatagramPocket(buff,buff.length);
4.  DatagramSocket receiveSocket=new DatagramSocket(2018);
5.  receiveSocket.receive(receivePacket);
6.  int length=receivePacket.getLength();
7.  String message=new String(receivePacket.getData(),0,length);
8.  System.out.println(message);
```

4.UDP编程实例

a.UDPCollection.java

两个主机的界面几乎是一样的，设置为公共代码复用。这就是核心代码了

```
1.
2.  /**
3.   * Created by XQF on 2016/12/4.
4.   */
5.  public class UDPCollection extends JFrame implements Runnable, ActionLi
    stener {
6.      private JTextField sendMsg;//信息内容文本框
7.      private JTextArea receivedMsg;//接收消息显示区
8.      private JButton sendBtn;
9.      private Container container;
10.     private Model model;
11.
12.
13.     private String titleString;
14.     private int inPort;
15.     private int outPort;
16.
17.     public UDPCollection(Model model) {
18.         this.model = model;
19.         container = this.getContentPane();
20.         this.setSize(400, 300);
21.         this.setVisible(true);
```

```
22.         this.setTitle(model.getTitleString());
23.         container.setLayout(new BorderLayout());//给顶层容器的默认布局换成
    BorderLayout布局,不过顶层容器的默认布局就是这个呀，简直多此一举
24.
25.
26.     // 添加滑动面板，也就是中间的消息界面
27.         JScrollPane centerPanel = new JScrollPane();//新建滑动面板对象
28.         receivedMsg = new JTextArea();
29.         centerPanel.setViewportView(receivedMsg);//把文本编辑区放进滚动面
    板
30.         // JScrollPanel centerPanel=new JScrollPanel(receivedMsg)
31.         container.add(centerPanel, BorderLayout.CENTER);//把滚动面板放在
    窗口最中间
32.
33.     // 添加底部面板，也就是编辑消息和发送消息按钮的面板
34.         JPanel bottomPanel = new JPanel();
35.         JLabel label = new JLabel("编辑信息");//创建一个标签提示栏对象
36.         sendMsg = new JTextField(20);//创建一个文本编辑框用于编辑消息，大小为
    20为列数，。大概也是指的宽度了。
37.         sendBtn = new JButton("发送消息");
38.         bottomPanel.add(label);
39.         bottomPanel.add(sendMsg);
40.         bottomPanel.add(sendBtn);
41.         container.add(bottomPanel, BorderLayout.SOUTH);
42.
43.     //下面对事件进行处理
44.
45.         sendBtn.addActionListener(this);//注册点击按钮事件
46.         sendMsg.addActionListener(this);//注册聊天栏动作事件
47.
48.     // 新开一个线程用于接收数据
49.         Thread t = new Thread(this);//因为实现了runable接口
50.         t.start();
51.     //点击退出就退出
52.         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
53.     }
54.
55.
56.     //点击事件处理
57.     public void actionPerformed(ActionEvent event) {
58.         byte[] buffer = sendMsg.getText().trim().getBytes();
59.         InetAddress destAddress = null;//先拿到一个主机地址，创建一个
    InetAddress对象，这个对象在后续创建DatagramPacket有用
60.         try {
61.             destAddress = InetAddress.getByName("127.0.0.1");
```

```
62.             } catch (UnknownHostException e) {
63.                 System.out.println("找不到主机！" + e.getMessage());
64.             }
65.
66.             //构造数据包一直都在用这个方法，相对来说简单，把端口，大小什么的都说明了，
67.             //将buff.length()大小的数据buff发送到目的地址为destAddress的2012端口
    处
68.             DatagramPacket dataPacket = new DatagramPacket(buffer, buffer.l
    ength, destAddress, model.getOutPort());
69.
70.             //创建套接字对象，准备打开管子放水了,这个类的构造方法还是比价简单，此时是
    发送就不需要绑定端口，所以使用这种构造方法
71.             DatagramSocket sendSocket = null;
72.             try {
73.                 sendSocket = new DatagramSocket();
74.             } catch (SocketException e) {
75.                 e.printStackTrace();
76.                 System.out.println("套接字创建错误！");
77.             }
78.
79.     //          receivedMsg.append("==================本地消息
    ==================\n");
80.     //          receivedMsg.append("数据报目标主机地址：" +
    dataPacket.getAddress() + "\n");//为了显摆一下在数据包里进行操
    作，InetAddress对象也可以获得。
81.     //          receivedMsg.append("数据报目标端口：" + dataPacket.getPort() +
    "\n");
82.     //          receivedMsg.append("数据报长度：" + dataPacket.getLength() +
    "\n");
83.
84.             receivedMsg.append(model.getTitleString()+"发出："+sendMsg.getTe
    xt().trim()+"\n");
85.             //发送数据报
86.             try {
87.                 sendSocket.send(dataPacket);
88.             } catch (IOException e) {
89.                 System.out.println("数据报发送错误！");
90.             }
91.
92.
93.             //将消息编辑界面置空，看上去是不见了
94.             sendMsg.setText("");
95.         }
96.
97.
```

```
98.      //子线程要干的事情,接收数据
99.      public void run() {
100.         DatagramPacket receivedPacket = null;
101.         DatagramSocket receivedSocket = null;
102.         byte[] buffer = new byte[8192];
103.
104.         try {
105.
106.             //发送的时候使用四个参数的构造方法，接收的时候使用两个参数的构造方法
107.             receivedPacket = new DatagramPacket(buffer, buffer.length);
108.             //接收的Socket与端口绑定，就是为了接收这个端口的数据
109.             receivedSocket = new DatagramSocket(model.getInPort());
110.         } catch (SocketException e) {
111.             e.printStackTrace();
112.         }
113.         while (true) {
114.             //如果套接字为空就跳出死循环,这里很奇怪，既然套接字里面都不为空了为什么在后面才来获取数据包
115.             if (receivedSocket == null) {
116.                 break;
117.             } else {
118.                 try {
119.                     receivedSocket.receive(receivedPacket);//接收数据，这句过后receivedPacket就不是空的了
120.                 } catch (IOException e) {
121.                     e.printStackTrace();
122.                 }
123.                 int length = receivedPacket.getLength();//获取内容长度只能在数据包里获取，不能在套接字，。，。套接字只是一条路。
124.                 InetAddress address = receivedPacket.getAddress();//拿到此套接字连接的地址，即获取发送人的地址
125.                 int port = receivedSocket.getPort();//获取发送者发送数据的端口号
126.
127.                 //将获取到的数据转换为字符串
128.                 String message = new String(receivedPacket.getData(), 0, length);//查了一下API文档，里面的String真的有这个构造方法
129. //                receivedMsg.append("==================异地消息==================");
130. //                receivedMsg.append("收到数据长度：" + length + "\n");
131. //                receivedMsg.append("收到数据来自：" + address + " 端口：" + port + "\n");
132. //                receivedMsg.append("收到数据是：" + message + "\n");
133.                 receivedMsg.append(model.getTitleString() + "收到：" + message+"\n");
```

```
                }
            }
        }
    }
```