1.网络编程入门

1.1 网络编程概述【理解】

- 计算机网络

  是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统，网络管理软件及网络通信协议的管理和协调下，实现资源共享和信息传递的计算机系统

- 网络编程

  在网络通信协议下，实现网络互连的不同计算机上运行的程序间可以进行数据交换

1.2 网络编程三要素【理解】

- IP地址

  要想让网络中的计算机能够互相通信，必须为每台计算机指定一个标识号，通过这个标识号来指定要接收数据的计算机和识别发送的计算机，而IP地址就是这个标识号。也就是设备的标识

- 端口

  网络的通信，本质上是两个应用程序的通信。每台计算机都有很多的应用程序，那么在网络通信时，如何区分这些应用程序呢？如果说IP地址可以唯一标识网络中的设备，那么端口号就可以唯一标识设备中的应用程序了。也就是应用程序的标识

- 协议

  通过计算机网络可以使多台计算机实现连接，位于同一个网络中的计算机在进行连接和通信时需要遵守一定的规则，这就好比在道路中行驶的汽车一定要遵守交通规则一样。在计算机网络中，这些连接和通信的规则被称为网络通信协议，它对数据的传输格式、传输速率、传输步骤等做了统一规定，通信双方必须同时遵守才能完成数据交换。常见的协议有UDP协议和TCP协议

1.3 IP地址【理解】

IP地址：是网络中设备的唯一标识

- IP地址分为两大类
  - IPv4：是给每个连接在网络上的主机分配一个32bit地址。按照TCP/IP规定，IP地址用二进制来表示，每个IP地址长32bit，也就是4个字节。例如一个采用二进制形式的IP地址是“11000000 10101000 00000001 01000010”，这么长的地址，处理起来也太费劲了。为了方便使用，IP地址经常被写成十进制的形式，中间使用符号“.”分隔不同的字节。于是，上面的IP地址可以表示为“192.168.1.66”。IP地址的这种表示法叫做“点分十进制表示法”，这显然比1和0容易记忆得多
  - IPv6：由于互联网的蓬勃发展，IP地址的需求量愈来愈大，但是网络地址资源有限，使得IP的分配越发紧张。为了扩大地址空间，通过IPv6重新定义地址空间，采用128位地址长度，每16个字节一组，分成8组十六进制数，这样就解决了网络地址资源数量不够的问题
- DOS常用命令：
  - ipconfig：查看本机IP地址
  - ping IP地址：检查网络是否连通
- 特殊IP地址：
  - 127.0.0.1：是回送地址，可以代表本机地址，一般用来测试使用

1.4InetAddress【应用】

InetAddress：此类表示Internet协议（IP）地址

- 相关方法

方法名	说明
static InetAddress getByName(String host)	确定主机名称的IP地址。主机名称可以是机器名称，也可以是IP地址
String getHostName()	获取此IP地址的主机名
String getHostAddress()	返回文本显示中的IP地址字符串

- 代码演示

```
public class InetAddressDemo {
    public static void main(String[] args) throws UnknownHostException {
        //InetAddress address = InetAddress.getByName("itheima");
        InetAddress address = InetAddress.getByName("192.168.1.66");

        //public String getHostName(): 获取此IP地址的主机名
        String name = address.getHostName();
        //public String getHostAddress(): 返回文本显示中的IP地址字符串
        String ip = address.getHostAddress();

        System.out.println("主机名: " + name);
        System.out.println("IP地址: " + ip);
    }
}
```

1.5端口和协议【理解】

- 端口
  - 设备上应用程序的唯一标识
- 端口号
  - 用两个字节表示的整数，它的取值范围是0~65535。其中，0~1023之间的端口号用于一些知名的网络服务和应用，普通的应用程序需要使用1024以上的端口号。如果端口号被另外一个服务或应用所占用，会导致当前程序启动失败
- 协议
  - 计算机网络中，连接和通信的规则被称为网络通信协议
- UDP协议
  - 用户数据报协议(User Datagram Protocol)
  - UDP是无连接通信协议，即在数据传输时，数据的发送端和接收端不建立逻辑连接。简单来说，当一台计算机向另外一台计算机发送数据时，发送端不会确认接收端是否存在，就会发出数据，同样接收端在收到数据时，也不会向发送端反馈是否收到数据。
  - 由于使用UDP协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输

    - 例如视频会议通常采用UDP协议，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。但是在使用UDP协议传送数据时，由于UDP的面向无连接性，不能保证数据的完整性，因此在传输重要数据时不建议使用UDP协议
- TCP协议
    - 传输控制协议 (Transmission Control Protocol)
    - TCP协议是面向连接的通信协议，即传输数据之前，在发送端和接收端建立逻辑连接，然后再传输数据，它提供了两台计算机之间可靠无差错的数据传输。在TCP连接中必须要明确客户端与服务器端，由客户端向服务端发出连接请求，每次连接的创建都需要经过“三次握手”
    - 三次握手：TCP协议中，在发送数据的准备阶段，客户端与服务器之间的三次交互，以保证连接的可靠

      第一次握手，客户端向服务器端发出连接请求，等待服务器确认

      第二次握手，服务器端向客户端回送一个响应，通知客户端收到了连接请求

      第三次握手，客户端再次向服务器端发送确认信息，确认连接
    - 完成三次握手，连接建立后，客户端和服务器就可以开始进行数据传输了。由于这种面向连接的特性，TCP协议可以保证传输数据的安全，所以应用十分广泛。例如上传文件、下载文件、浏览网页等

2.UDP通信程序

2.1 UDP发送数据【应用】

- Java中的UDP通信
    - UDP协议是一种不可靠的网络协议，它在通信的两端各建立一个Socket对象，但是这两个Socket只是发送，接收数据的对象，因此对于基于UDP协议的通信双方而言，没有所谓的客户端和服务器的概念
    - Java提供了DatagramSocket类作为基于UDP协议的Socket
- 构造方法

方法名	说明
DatagramSocket()	创建数据报套接字并将其绑定到本机地址上的任何可用端口
DatagramPacket(byte[] buf,int len,InetAddress add,int port)	创建数据包,发送长度为len的数据包到指定主机的指定端口

- 相关方法

方法名	说明
void send(DatagramPacket p)	发送数据报包
void close()	关闭数据报套接字
void receive(DatagramPacket p)	从此套接字接受数据报包

- 发送数据的步骤
    - 创建发送端的Socket对象(DatagramSocket)
    - 创建数据，并把数据打包

    - 调用DatagramSocket对象的方法发送数据
    - 关闭发送端
- 代码演示

```
public class SendDemo {
    public static void main(String[] args) throws IOException {
        //创建发送端的Socket对象(DatagramSocket)
        // DatagramSocket() 构造数据报套接字并将其绑定到本地主机上的任何可用端口
        DatagramSocket ds = new DatagramSocket();

        //创建数据，并把数据打包
        //DatagramPacket(byte[] buf, int length, InetAddress address, int port)
        //构造一个数据包，发送长度为 length的数据包到指定主机上的指定端口号。
        byte[] bys = "hello,udp,我来了".getBytes();

        DatagramPacket dp = new
DatagramPacket(bys,bys.length,InetAddress.getByName("192.168.1.66"),10086);

        //调用DatagramSocket对象的方法发送数据
        //void send(DatagramPacket p) 从此套接字发送数据报包
        ds.send(dp);

        //关闭发送端
        //void close() 关闭此数据报套接字
        ds.close();
    }
}
```

2.2UDP接收数据【应用】

- 接收数据的步骤
    - 创建接收端的Socket对象(DatagramSocket)
    - 创建一个数据包，用于接收数据
    - 调用DatagramSocket对象的方法接收数据
    - 解析数据包，并把数据在控制台显示
    - 关闭接收端
- 构造方法

方法名	说明
DatagramPacket(byte[] buf, int len)	创建一个DatagramPacket用于接收长度为len的数据包

- 相关方法

方法名	说明
byte[] getData()	返回数据缓冲区
int getLength()	返回要发送的数据的长度或接收的数据的长度

- 示例代码

```
public class ReceiveDemo {
    public static void main(String[] args) throws IOException {
        //创建接收端的Socket对象(DatagramSocket)
        DatagramSocket ds = new DatagramSocket(12345);

        while (true) {
            //创建一个数据包，用于接收数据
            byte[] bys = new byte[1024];
            DatagramPacket dp = new DatagramPacket(bys, bys.length);

            //调用DatagramSocket对象的方法接收数据
            ds.receive(dp);

            //解析数据包，并把数据在控制台显示
            System.out.println("数据是: " + new String(dp.getData(), 0,
                               dp.getLength()));
        }
    }
}
```

2.3UDP通信程序练习【应用】

- 案例需求

  UDP发送数据：数据来自于键盘录入，直到输入的数据是886，发送数据结束

  UDP接收数据：因为接收端不知道发送端什么时候停止发送，故采用死循环接收

- 代码实现

```
/*
    UDP发送数据：
        数据来自于键盘录入，直到输入的数据是886，发送数据结束
 */
public class SendDemo {
    public static void main(String[] args) throws IOException {
        //创建发送端的Socket对象(DatagramSocket)
        DatagramSocket ds = new DatagramSocket();
        //自己封装键盘录入数据
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String line;
        while ((line = br.readLine()) != null) {
            //输入的数据是886，发送数据结束
            if ("886".equals(line)) {
                break;
            }
            //创建数据，并把数据打包
            byte[] bys = line.getBytes();
            DatagramPacket dp = new DatagramPacket(bys, bys.length,
InetAddress.getByName("192.168.1.66"), 12345);

            //调用DatagramSocket对象的方法发送数据
```

```
            ds.send(dp);
        }
        //关闭发送端
        ds.close();
    }
}

/*
    UDP接收数据：
        因为接收端不知道发送端什么时候停止发送，故采用死循环接收
 */
public class ReceiveDemo {
    public static void main(String[] args) throws IOException {
        //创建接收端的Socket对象(DatagramSocket)
        DatagramSocket ds = new DatagramSocket(12345);
        while (true) {
            //创建一个数据包，用于接收数据
            byte[] bys = new byte[1024];
            DatagramPacket dp = new DatagramPacket(bys, bys.length);
            //调用DatagramSocket对象的方法接收数据
            ds.receive(dp);
            //解析数据包，并把数据在控制台显示
            System.out.println("数据是：" + new String(dp.getData(), 0,
dp.getLength()));
        }
        //关闭接收端
//        ds.close();
    }
}
```

3.TCP通信程序

3.1TCP发送数据【应用】

- Java中的TCP通信
  - Java对基于TCP协议的的网络提供了良好的封装，使用Socket对象来代表两端的通信端口，并通过Socket产生IO流来进行网络通信。
  - Java为客户端提供了Socket类，为服务器端提供了ServerSocket类
- 构造方法

方法名	说明
Socket(InetAddress address,int port)	创建流套接字并将其连接到指定IP指定端口号
Socket(String host, int port)	创建流套接字并将其连接到指定主机上的指定端口号

- 相关方法

方法名	说明
InputStream getInputStream()	返回此套接字的输入流
OutputStream getOutputStream()	返回此套接字的输出流

- 示例代码

```
public class ClientDemo {
    public static void main(String[] args) throws IOException {
        //创建客户端的Socket对象(Socket)
        //Socket(String host, int port) 创建流套接字并将其连接到指定主机上的指定端口号
        Socket s = new Socket("192.168.1.66",10000);

        //获取输出流，写数据
        //OutputStream getOutputStream() 返回此套接字的输出流
        OutputStream os = s.getOutputStream();
        os.write("hello,tcp,我来了".getBytes());

        //释放资源
        s.close();
    }
}
```

3.2TCP接收数据【应用】

- 构造方法

方法名	说明
ServletSocket(int port)	创建绑定到指定端口的服务器套接字

- 相关方法

方法名	说明
Socket accept()	监听要连接到此的套接字并接受它

- 示例代码

```
public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器端的Socket对象(ServerSocket)
        //ServerSocket(int port) 创建绑定到指定端口的服务器套接字
        ServerSocket ss = new ServerSocket(10000);

        //Socket accept() 侦听要连接到此套接字并接受它
        Socket s = ss.accept();

        //获取输入流，读数据，并把数据显示在控制台
        InputStream is = s.getInputStream();
```

```
        byte[] bys = new byte[1024];
        int len = is.read(bys);
        String data = new String(bys,0,len);
        System.out.println("数据是: " + data);

        //释放资源
        s.close();
        ss.close();
    }
}
```

3.3TCP通信程序练习【应用】

- 案例需求

  客户端：发送数据，接受服务器反馈

  服务器：收到消息后给出反馈

- 案例分析
  - 客户端创建对象，使用输出流输出数据
  - 服务端创建对象，使用输入流接受数据
  - 服务端使用输出流给出反馈数据
  - 客户端使用输入流接受反馈数据
- 代码实现

```
public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器端的Socket对象(ServerSocket)
        ServerSocket ss = new ServerSocket(10000);

        //监听客户端连接，返回一个Socket对象
        Socket s = ss.accept();

        //获取输入流，读数据，并把数据显示在控制台
        InputStream is = s.getInputStream();
        byte[] bys = new byte[1024];
        int len = is.read(bys);
        String data = new String(bys, 0, len);
        System.out.println("服务器: " + data);

        //给出反馈
        OutputStream os = s.getOutputStream();
        os.write("数据已经收到".getBytes());

        //释放资源
//        s.close();
        ss.close();
    }
}

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

```
        //创建客户端的Socket对象(Socket)
        Socket s = new Socket("192.168.1.66", 10000);

        //获取输出流，写数据
        OutputStream os = s.getOutputStream();
        os.write("hello,tcp,我来了".getBytes());

        //接收服务器反馈
        InputStream is = s.getInputStream();
        byte[] bys = new byte[1024];
        int len = is.read(bys);
        String data = new String(bys, 0, len);
        System.out.println("客户端: " + data);

        //释放资源
//        is.close();
//        os.close();
        s.close();
    }
}
```

3.4TCP通信程序练习【应用】

- 案例需求

  客户端：数据来自于键盘录入，直到输入的数据是886，发送数据结束

  服务端：接收到数据在控制台输出

- 案例分析
  - 客户端创建对象，使用键盘录入循环接受数据，接受一行发送一行，直到键盘录入886为止
  - 服务端创建对象，使用输入流按行循环接受数据，直到接受到null为止
- 代码实现

```
public class ClientDemo {
    public static void main(String[] args) throws IOException {
        //创建客户端Socket对象
        Socket s = new Socket("192.168.1.66",10000);

        //数据来自于键盘录入，直到输入的数据是886，发送数据结束
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        //封装输出流对象
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));
        String line;
        while ((line=br.readLine())!=null) {
            if("886".equals(line)) {
                break;
            }

            //获取输出流对象
            bw.write(line);
            bw.newLine();
```

```
            bw.flush();
        }

        //释放资源
        s.close();
    }
}

public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器Socket对象
        ServerSocket ss = new ServerSocket(10000);

        //监听客户端的连接，返回一个对应的Socket对象
        Socket s = ss.accept();

        //获取输入流
        BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }

        //释放资源
        ss.close();
    }
}
```

3.5TCP通信程序练习【应用】

- 案例需求

  客户端：数据来自于键盘录入，直到输入的数据是886,发送数据结束

  服务端：接受到的数据写入文本文件中

- 案例分析
  - 客户端创建对象，使用键盘录入循环接受数据，接受一行发送一行，直到键盘录入886为止
  - 服务端创建对象，创建输出流对象指向文件，每接受一行数据后使用输出流输出到文件中，直到接受到null为止
- 代码实现

```
ublic class ClientDemo {
    public static void main(String[] args) throws IOException {
        //创建客户端Socket对象
        Socket s = new Socket("192.168.1.66",10000);
        //数据来自于键盘录入，直到输入的数据是886，发送数据结束
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        //封装输出流对象
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));
        String line;
```

```
        while ((line=br.readLine())!=null) {
            if("886".equals(line)) {
                break;
            }
            bw.write(line);
            bw.newLine();
            bw.flush();
        }
        //释放资源
        s.close();
    }
}

public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器Socket对象
        ServerSocket ss = new ServerSocket(10000);
        //监听客户端连接，返回一个对应的Socket对象
        Socket s = ss.accept();
        //接收数据
        BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        //把数据写入文本文件
        BufferedWriter bw = new BufferedWriter(new FileWriter("myNet\\s.txt"));

        String line;
        while ((line=br.readLine())!=null) {
            bw.write(line);
            bw.newLine();
            bw.flush();
        }

        //释放资源
        bw.close();
        ss.close();
    }
}
```

3.6TCP通信程序练习【应用】

- 案例需求

  客户端：数据来自于文本文件

  服务器：接收到的数据写入文本文件

- 案例分析
  - 创建客户端，创建输入流对象指向文件，从文件循环读取数据，每读取一行就使用输出流给服务器输出一行
  - 创建服务端，创建输出流对象指向文件，从客户端接受数据，每接受一行就给文件中输出一行
- 代码实现

```
public class ClientDemo {
```

```
    public static void main(String[] args) throws IOException {
        //创建客户端Socket对象
        Socket s = new Socket("192.168.1.66",10000);

        //封装文本文件的数据
        BufferedReader br = new BufferedReader(new
FileReader("myNet\\InetAddressDemo.java"));
        //封装输出流写数据
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));

        String line;
        while ((line=br.readLine())!=null) {
            bw.write(line);
            bw.newLine();
            bw.flush();
        }

        //释放资源
        br.close();
        s.close();
    }
}

public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器Socket对象
        ServerSocket ss = new ServerSocket(10000);

        //监听客户端连接，返回一个对应的Socket对象
        Socket s = ss.accept();

        //接收数据
        BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        //把数据写入文本文件
        BufferedWriter bw = new BufferedWriter(new FileWriter("myNet\\Copy.java"));

        String line;
        while ((line=br.readLine())!=null) {
            bw.write(line);
            bw.newLine();
            bw.flush();
        }

        //释放资源
        bw.close();
        ss.close();
    }
}
```

3.7TCP通信程序练习【应用】

- 案例需求

  客户端：数据来自于文本文件，接收服务器反馈

  服务器：接收到的数据写入文本文件，给出反馈

- 案例分析
  - 创建客户端对象，创建输入流对象指向文件，每读入一行数据就给服务器输出一行数据，输出结束后使用shutdownOutput()方法告知服务端传输结束
  - 创建服务器对象，创建输出流对象指向文件，每接受一行数据就使用输出流输出到文件中，传输结束后。使用输出流给客户端反馈信息
  - 客户端接受服务端的回馈信息
- 相关方法

方法名	说明
void shutdownInput()	将此套接字的输入流放置在“流的末尾”
void shutdownOutput()	禁止用此套接字的输出流

- 代码实现

```
public class ClientDemo {
    public static void main(String[] args) throws IOException {
        //创建客户端Socket对象
        Socket s = new Socket("192.168.1.66",10000);

        //封装文本文件的数据
        BufferedReader br = new BufferedReader(new
FileReader("myNet\\InetAddressDemo.java"));
        //封装输出流写数据
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));

        String line;
        while ((line=br.readLine())!=null) {
            bw.write(line);
            bw.newLine();
            bw.flush();
        }

        //public void shutdownOutput()
        s.shutdownOutput();

        //接收反馈
        BufferedReader brClient = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        String data = brClient.readLine(); //等待读取数据
        System.out.println("服务器的反馈: " + data);

        //释放资源
        br.close();
        s.close();
```

```
    }
}

public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器Socket对象
        ServerSocket ss = new ServerSocket(10000);

        //监听客户端连接，返回一个对应的Socket对象
        Socket s = ss.accept();

        //接收数据
        BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        //把数据写入文本文件
        BufferedWriter bw = new BufferedWriter(new FileWriter("myNet\\Copy.java"));

        String line;
        while ((line=br.readLine())!=null) { //等待读取数据
            bw.write(line);
            bw.newLine();
            bw.flush();
        }
        //给出反馈
        BufferedWriter bwServer = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));
        bwServer.write("文件上传成功");
        bwServer.newLine();
        bwServer.flush();

        //释放资源
        bw.close();
        ss.close();
    }
}
```

3.8TCP通信程序练习【应用】

- 案例需求

  客户端：数据来自于文本文件，接收服务器反馈

  服务器：接收到的数据写入文本文件，给出反馈，代码用线程进行封装，为每一个客户端开启一个线程

- 案例分析
  - 创建客户端对象，创建输入流对象指向文件，每读入一行数据就给服务器输出一行数据，输出结束后使用shutdownOutput()方法告知服务端传输结束
  - 创建多线程类，在run()方法中读取客户端发送的数据，为了防止文件重名，使用计数器给文件名编号，接受结束后使用输出流给客户端发送反馈信息。
  - 创建服务端对象，每监听到一个客户端则开启一个新的线程接受数据。
  - 客户端接受服务端的回馈信息
- 代码实现

```
public class ClientDemo {
    public static void main(String[] args) throws IOException {
        //创建客户端Socket对象
        Socket s = new Socket("192.168.1.66",10000);

        //封装文本文件的数据
        BufferedReader br = new BufferedReader(new
FileReader("myNet\\InetAddressDemo.java"));
        //封装输出流写数据
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));

        String line;
        while ((line=br.readLine())!=null) {
            bw.write(line);
            bw.newLine();
            bw.flush();
        }

        s.shutdownOutput();

        //接收反馈
        BufferedReader brClient = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        String data = brClient.readLine(); //等待读取数据
        System.out.println("服务器的反馈: " + data);

        //释放资源
        br.close();
        s.close();
    }
}

public class ServerThread implements Runnable {
    private Socket s;

    public ServerThread(Socket s) {
        this.s = s;
    }

    @Override
    public void run() {
        try {
            //接收数据写到文本文件
            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
            //解决名称冲突问题
            int count = 0;
            File file = new File("myNet\\Copy["+count+"].java");
            while (file.exists()) {
                count++;
                file = new File("myNet\\Copy["+count+"].java");
            }
```

```
            BufferedWriter bw = new BufferedWriter(new FileWriter(file));

            String line;
            while ((line=br.readLine())!=null) {
                bw.write(line);
                bw.newLine();
                bw.flush();
            }

            //给出反馈
            BufferedWriter bwServer = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));
            bwServer.write("文件上传成功");
            bwServer.newLine();
            bwServer.flush();

            //释放资源
            s.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public class ServerDemo {
    public static void main(String[] args) throws IOException {
        //创建服务器Socket对象
        ServerSocket ss = new ServerSocket(10000);

        while (true) {
            //监听客户端连接，返回一个对应的Socket对象
            Socket s = ss.accept();
            //为每一个客户端开启一个线程
            new Thread(new ServerThread(s)).start();
        }

    }
}
```