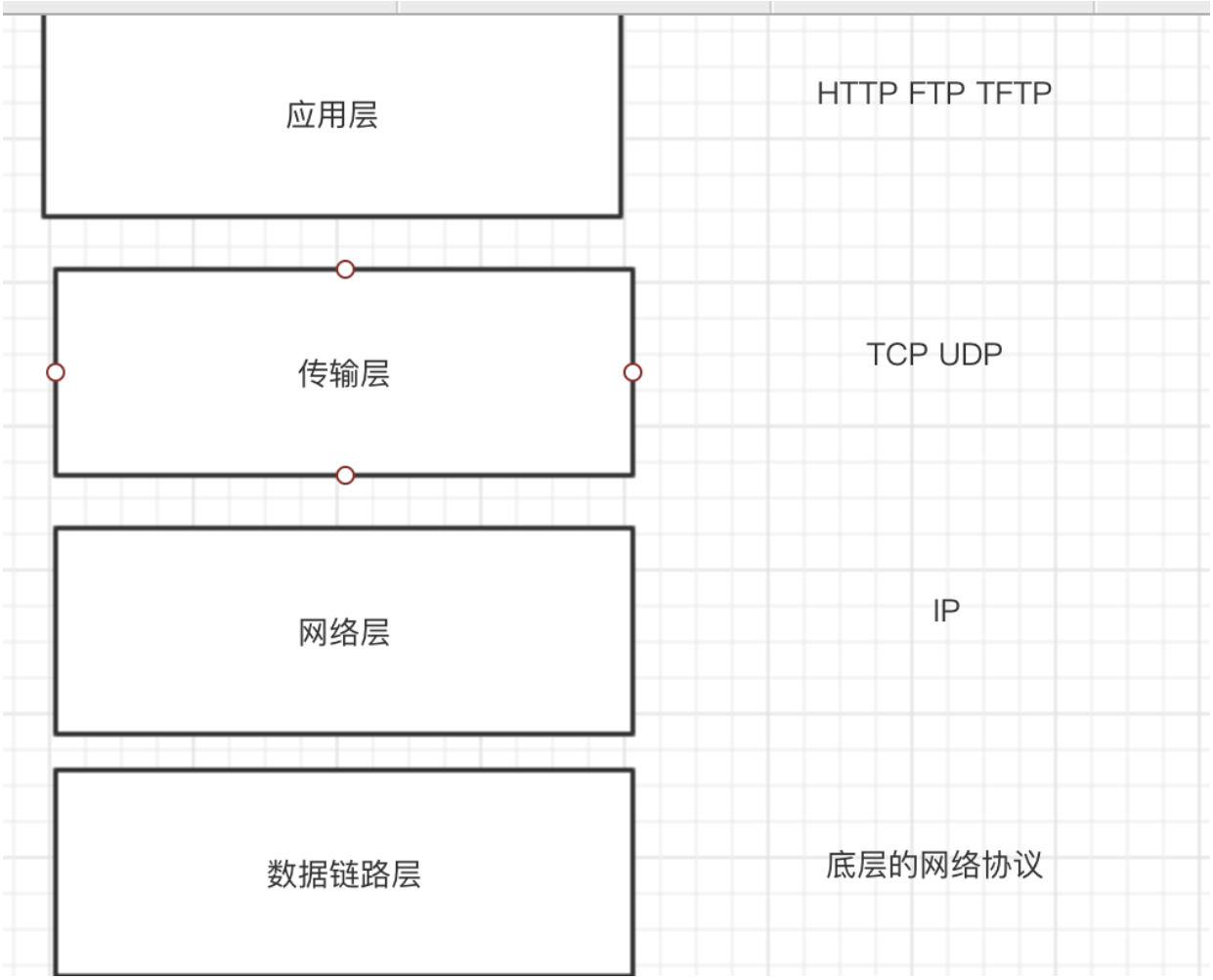
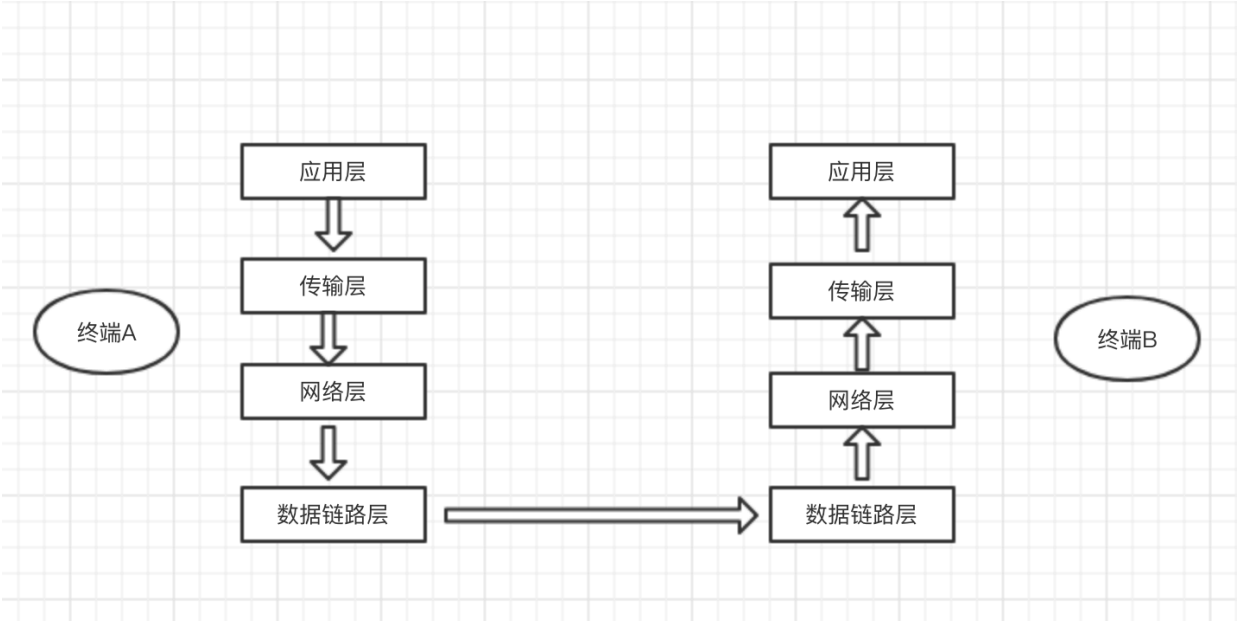
# 网络编程

计算机网络就是通过硬件设施，传输媒介把分散在不同地区的计算机进行连接，形成了一个资源共享和数据传输的网络系统。两台终端通过网络进行连接时，需要遵守一定的规则，这个规则就是网络通信协议，网络通信协议有TCP/IP协议，IPX/SPX协议，NetBEUI协议等，我们常用的是TCP/IP协议，同时TCP/IP协议是分层的。

TCP/IP协议可以分为四层，分别是应用层，传输层，网络层，数据链路层/物理层。




比如终端A正在和终端B通过网络进行通信，整个数据的传输流程是终端A—应用层—传输层—网络层—数据链路层—网络层—传输层—应用层—终端B。




# IP

互联网上的每台计算机都有一个唯一标识，网络中的请求可以根据这个标识找到具体的计算机。这个唯一标识就是IP地址（Internet Protocol），用户可以通过操作系统的设置来查看本机IP地址，windows下命令行：ipconfig，

mac下命令行：ifconfig。IP地址使用32位的二进制数据表示，我们查到的IP地址已经转为10进制。

IP地址 = 网络地址+主机地址

网络地址：用来识别主机所在的网络

主机地址：用来识别网络中的主机

IP地址分为5类，A类是政府机构使用，B类是大中型公司企业，C类是个人使用，D类用于组播，E类用于实验，各类的IP地址可容纳的地址数量不同。

A类 1.0.0.1 ~ 126.255.255.254

B类 128.0.0.1 ～ 191.255.255.254

C类 192.0.0.1 ~ 223.255.255.254

D类 224.0.0.1 ~ 239.255.255.254

E类 240.0.0.1 ~ 255.255.255.254

需要注意的是127.0.0.1用来测试，表示的是本机IP，同时localhost也表示本机。

IP地址分为IPv4和IPv6两种，其中IPv4是使用最广泛的版本。

Java中有专门的类来表示IP地址，这个类是java.net.InetAddress。

常用的方法

public static InetAddress getLocalHost() 获取本机的InetAddress对象

public static InetAddress getByName(String host) 通过主机名称获取对应IP的InetAddress对象

String getHostName() 获取主机名称

public String getHostAddress() 获取主机的IP地址

```
package com.southwind.test;

import java.net.InetAddress;
import java.net.UnknownHostException;

public class Test {
    public static void main(String[] args) {
        try {
            //获取本机InetAddress(IP地址和计算机名称)
            InetAddress inetAddress = InetAddress.getLocalHost();
            System.out.println(inetAddress);
            //获取主机名
            System.out.println(inetAddress.getHostName());
            //获取IP地址
            System.out.println(inetAddress.getHostAddress());
            //获取127.0.0.1的InetAddress
            InetAddress inetAddress2 = InetAddress.getByName("127.0.0.1");
            System.out.println(inetAddress2);
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

## 端口

如果把IP地址比作一栋大厦的地址，那么端口（port）就是不同的房间的门牌号，IP地址需要和端口结合起来使用，IP地址负责找到终端，一台终端可以同时运行多个服务，通过端口来确定具体的服务。

## URL

URL（Uniform Resource Locator）统一资源定位符，可以直接通过URL找到互联网中的资源，网页html，文字，图片，视频，音频。

URL类常用的方法：

public URL(String protocol,String host,int port,String file) 根据协议，IP地址，端口号，资源名称获取URL对象

public final InputStream openStream() 获取输入流

```
package com.southwind.test;
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.MalformedURLException;
import java.net.URL;

public class Test2 {
    public static void main(String[] args) {
        InputStream inputStream = null;
        Reader reader = null;
        BufferedReader bufferedReader = null;
        try {
            URL url = new
URL("http","127.0.0.1",8080,"/libmanagesys/login.jsp");
            inputStream = url.openStream();
            reader = new InputStreamReader(inputStream);
            bufferedReader = new BufferedReader(reader);
            String str = null;
            while((str = bufferedReader.readLine())!=null) {
                System.out.println(str);
            }
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                bufferedReader.close();
                reader.close();
                inputStream.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

## URLConnection

URLConnection是封装访问远程网络资源一般方法的类，通过它可以建立与远程服务器的连接，检查远程资源的一些属性。

public int getContentLength() 获取内容的长度，int类型

public long getContentLengthLong() 获取内容的长度，long类型

public String getContentType() 获取内容的类型

```
package com.southwind.test;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

public class Test2 {
    public static void main(String[] args) {
        try {
            URL url = new
URL("http","127.0.0.1",8080,"/libmanagesys/images/img.jpeg");
            URLConnection urlConnection = url.openConnection();
            long length = urlConnection.getContentLengthLong();
            System.out.println("资源的长度："+length);
            String type = urlConnection.getContentType();
            System.out.println("资源的类型："+type);
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

## URLDecoder和URLEncoder

在使用URL访问服务时，通常会看到地址后面会追加很多其他的附带信息，将name=张三&age=22作为参数传给Java后台服务，上图中可以看到英文和数字可以正常显示，但是中文会变成%数字字母组成的另外一种编码形式，服务器获取到该请求之后如果想拿到正确的中文信息，一定需要进行解码操作，Java中提供了java.net.URLDecoder类完成解码，将“%E5%BC%A0%E4%B8%89”转为“张三”，同时还提供了java.net.URLEncoder类，可以将汉字转换为HTTP请求的编码方式，即“张三”转为“%E5%BC%A0%E4%B8%89”。

```
package com.southwind.test;

import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.net.URLEncoder;

public class Test3 {
    public static void main(String[] args) {
        String str = "张三";
        try {
            String encode = URLEncoder.encode(str, "UTF-8");
            System.out.println("编码："+encode);
            String decode = URLDecoder.decode(encode, "UTF-8");
            System.out.println("解码："+decode);
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

## TCP和UDP

TCP：可靠的连接，握手，就是看两台终端是否联通了，联通之后才能传输数据。

1.先建立连接，确定联通之后，再进行数据传输。

2.一方每传输一次数据，需要对方确认响应，再进行下一次数据传输，如果没有响应，就一直等待。

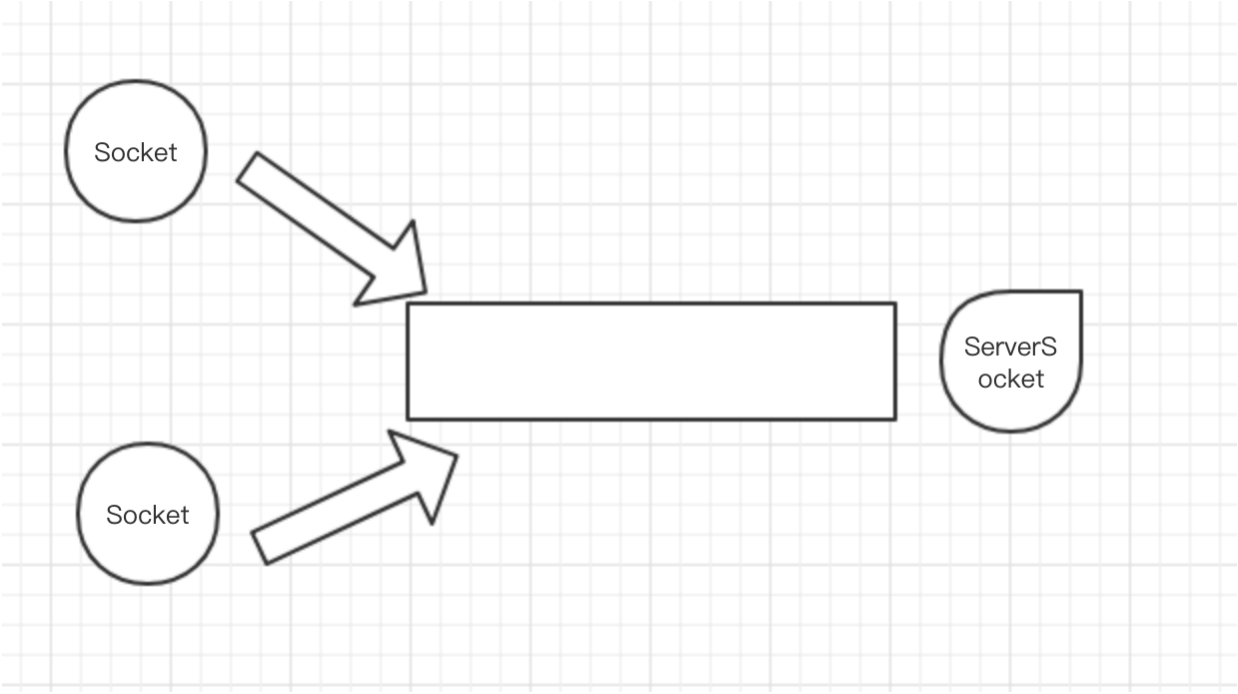
UDP：不可靠的连接，只管发送，不管对方是否能够接收到。

TCP虽然可靠，但是效率低，UDP虽然不可靠，但是效率高。

需要根据具体的业务场景来决定选择TCP还是UDP，金融系统一定是TCP，语音通话，视频通话都是UDP。

## Socket

TCP：在Socket程序开发中，服务端使用ServerSocket等待客户端的请求，对于Java网络编程来说，每一个客户端都使用一个Socket对象来表示。




ServerSocket类常用的方法

public ServerSocket(int port) 通过端口创建ServerSocket实例对象。

public InetAddress getInetAddress() 获取服务端的IP地址。

public Socket accpet() 等待客户端请求，并返回Socket对象。

public void close() 关闭ServerSocket对象。

public boolean isClosed() 判断ServerSocket是否关闭。

Socket类常用的方法

public Socket(String host,int port) 根据服务器IP，端口创建要连接的Socket对象

public InputStream getInputStream() 获取Socket的输入流。

public synchronized void close() 关闭Socket。

public boolean isClosed() 判断Sokcet是否关闭。

服务端

```
package com.southwind.net;

import java.io.DataInputStream;
import java.io.DataOutputStream;
```

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class TCPServer {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket socket = null;
        InputStream inputStream = null;
        DataInputStream dataInputStream = null;
        OutputStream outputStream = null;
        DataOutputStream dataOutputStream = null;
        try {
            //建立ServerSocket监听，端口为8080
            serverSocket = new ServerSocket(8080);
            System.out.println("服务已启动，等待接收客户端请求...");
            socket = serverSocket.accept();
            //接收客户端请求
            inputStream = socket.getInputStream();
            dataInputStream = new DataInputStream(inputStream);
            String request = dataInputStream.readUTF();
            System.out.println("接收到了客户端请求："+request);
            //给客户端做出响应
            outputStream = socket.getOutputStream();
            dataOutputStream = new DataOutputStream(outputStream);
            String response = "Hello World";
            dataOutputStream.writeUTF(response);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                dataOutputStream.close();
                outputStream.close();
                dataInputStream.close();
                inputStream.close();
                socket.close();
                serverSocket.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

客户端

```
package com.southwind.net;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class TCPClient {
    public static void main(String[] args) {
        Socket socket = null;
        OutputStream outputStream = null;
        DataOutputStream dataOutputStream = null;
        InputStream inputStream = null;
        DataInputStream dataInputStream = null;
        try {
            //建立客户端Socket，连接端口为8080的本地服务
            socket = new Socket("127.0.0.1", 8080);
            //给服务器发消息
            String message = "服务器你好，我是客户端";
            System.out.println("客户端说："+message);
            outputStream = socket.getOutputStream();
            dataOutputStream = new DataOutputStream(outputStream);
            dataOutputStream.writeUTF(message);
            //接收服务器的响应
            inputStream = socket.getInputStream();
            dataInputStream = new DataInputStream(inputStream);
            String response = dataInputStream.readUTF();
            System.out.println("我是客户端，服务器响应为："+response);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                dataInputStream.close();
                inputStream.close();
                dataOutputStream.close();
                outputStream.close();
                socket.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

Java中使用DatagramSocket类和DatagramPacket类完成UDP程序的开发。

DatagramSocket的常用方法

public DatagramSocket(int port) 根据端口创建DatagramSocket实例对象

public void send(DatagramPacket p) 发送数据

public synchronized void recevie(DatagramPacket p) 接收数据

DatagramPacket的常用方法

public DatagramPacket(byte buf[],int length,InetAddress address,int port) 根据发送的数据，数据长度，IP地址，端口号创建DatagramPacket对象

public synchronized byte[] getData() 获取接收到的数据

public synchronized int getLength() 获取数据的长度

public synchronized int getPort() 获取发送数据的Socket端口

public synchronized SocketAddress getSokcetAddress() 获取发送数据的Socket信息

```
package com.southwind.net;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketAddress;
import java.net.SocketException;

public class Receive {
    public static void main(String[] args) {
        //接收数据包
        byte[] buff = new byte[1024];
        DatagramPacket datagramPacket = new
DatagramPacket(buff,buff.length);
        DatagramSocket datagramSocket = null;
        try {
            datagramSocket = new DatagramSocket(8080);
            datagramSocket.receive(datagramPacket);
            String mess = new
String(datagramPacket.getData(),0,datagramPacket.getLength());
            System.out.println("我是Receive，接收到
了"+datagramPacket.getPort()+"传来的数据："+mess);

            //发送数据包
            String reply = "我已接收到了你的信息";
```

```
            SocketAddress socketAddress =
datagramPacket.getSocketAddress();
            DatagramPacket datagramPacket2 = new
DatagramPacket(reply.getBytes(), reply.getBytes().length, socketAddress);
            datagramSocket.send(datagramPacket2);
        } catch (SocketException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            datagramSocket.close();
        }

    }
}
```

```
package com.southwind.net;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

public class Send {
    public static void main(String[] args) {
        String mess = "我是一个请求";
        DatagramSocket datagramSocket = null;
        //数据打包
        try {
            InetAddress inetAddress = InetAddress.getByName("localhost");
            DatagramPacket datagramPacket = new
DatagramPacket(mess.getBytes(), mess.getBytes().length, inetAddress, 8080);
            datagramSocket = new DatagramSocket(8181);
            datagramSocket.send(datagramPacket);

            //接收返回的数据
            byte[] buff = new byte[1024];
            DatagramPacket datagramPacket2 = new DatagramPacket(buff,
buff.length);
            datagramSocket.receive(datagramPacket2);
            String reply = new
String(datagramPacket2.getData(),0,datagramPacket2.getLength());
```

```
            System.out.println("我是Send，接收到
了"+datagramPacket2.getPort()+"返回的数据："+reply);
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SocketException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            datagramSocket.close();
        }

    }
}
```

多线程网络编程

```
package com.southwind.net.thread;

import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerThread {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(8080);
            System.out.println("服务器已启动");
            while(true) {
                Socket socket = serverSocket.accept();
                new Thread(new ServerRunnable(socket)).start();
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

class ServerRunnable implements Runnable{
    private Socket socket;
```

```
    public ServerRunnable(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        // TODO Auto-generated method stub
        InputStream inputStream = null;
        DataInputStream dataInputStream = null;
        try {
            inputStream = this.socket.getInputStream();
            dataInputStream = new DataInputStream(inputStream);
            String message = dataInputStream.readUTF();
            System.out.println(message);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                dataInputStream.close();
                inputStream.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }

}
```

```
package com.southwind.net.thread;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

public class ClientThread {
    public static void main(String[] args) {
        for(int i = 0; i < 100; i++) {
            new Thread(new ClientRunnable(i)).start();
        }
    }
}

class ClientRunnable implements Runnable{
```

```
    private int num;
    public ClientRunnable(int num) {
        this.num = num;
    }
    @Override
    public void run() {
        // TODO Auto-generated method stub
        Socket socket = null;
        OutputStream outputStream = null;
        DataOutputStream dataOutputStream = null;
        try {
            socket = new Socket("localhost", 8080);
            String mess = "我是客户端"+this.num;
            outputStream = socket.getOutputStream();
            dataOutputStream = new DataOutputStream(outputStream);
            dataOutputStream.writeUTF(mess);
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                dataOutputStream.close();
                outputStream.close();
                socket.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }

}
```