网络编程概述

1.1、概述

Java是Internet上的语言，它从语言级上提供了对网络应用程序的支持，程序员能够很容易开发常见的网络应用程序。Java提供的网络类库，可以实现无痛的网络连接，联网的底层细节被隐藏在 Java 的本机安装系统里，由JVM进行控制。并且Java实现了一个跨平台的网络库，**程序员面对的是一个统一的网络编程环境。**

计算机网络：

把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大，功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件，软件，数据信息等资源。 【百度：地球村】在几个世纪之前，你都不敢想你在北京说话，美国都能听到！

网络编程的目的：

直接或间接地通过网络协议与其他计算机实现数据交换，进行通讯。

网络编程中有两个主要的问题：

1. 如何准确的定位网络上的一台或多台主机，定位主机上的特定的应用
2. 找到主机后如何可靠高效地进行数据传输

网络编程！ = 网页编程（web开发）

B/S 架构 和 C/S 架构

1.2、网络通信两个要素

如何实现网络中的主机互相通信?

通信双方的地址：

IP

端口号

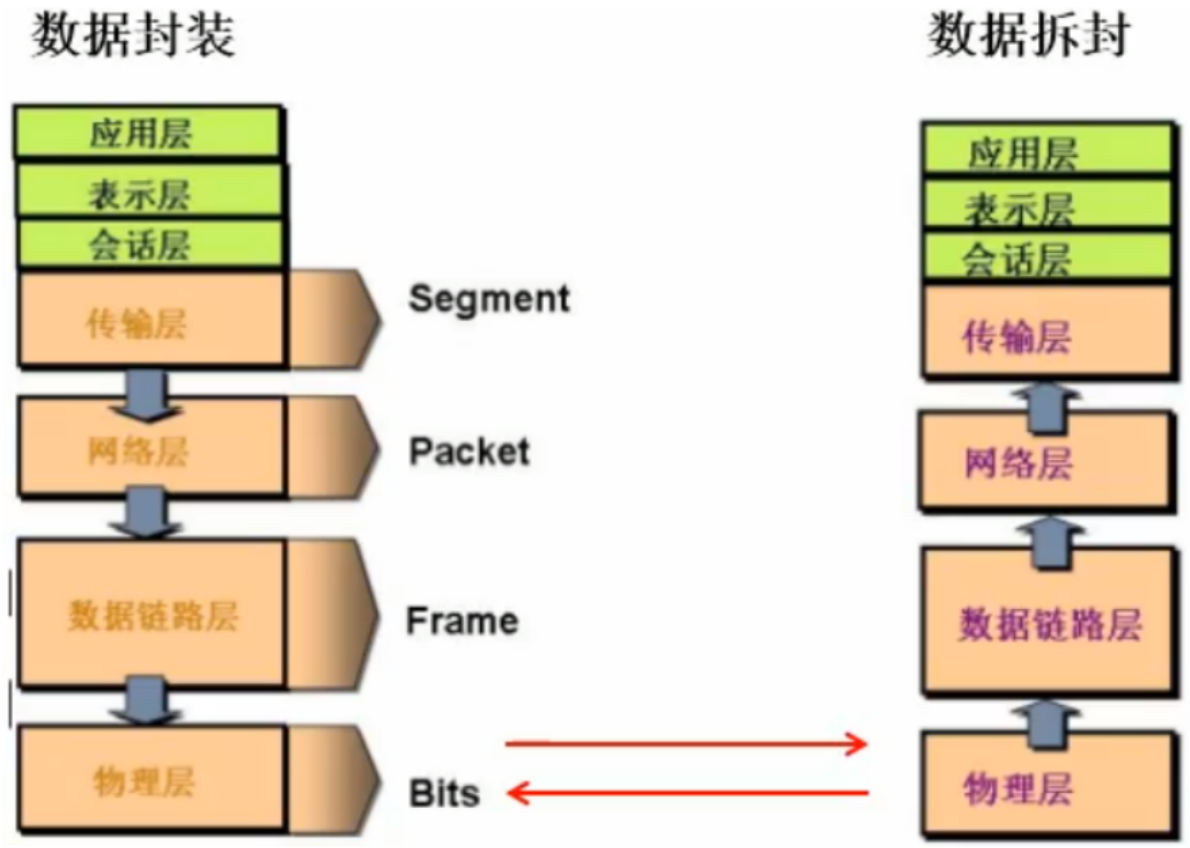
【比如说我们这里上课使用的就是局域网，你们连接到我的电脑，就能查看到我电脑的画面了】

一定的规则：（即，网络通信协议，有两套参考模型）

OSI 参考模型：模型过于理想化，未能在因特网上进行广泛推广！

TCP/IP 参考模型：TCP/IP协议，事实上的国际标准。

OSI参考模型	TCP/IP参考模型	TCP/IP参考模型各层对应协议
应用层	应用层	HTTP、FTP、Telnet、DNS…
表示层		
会话层		
传输层	传输层	TCP、UDP、…
网络层	网络层	IP、ICMP、ARP…
数据链路层	物理+数据链路层	Link
物理层		



小总结:

1. 网络编程中有两个主要的问题:

   如何准确的定位网络上一台或多台主机;

   定位主机上的特定的应用 找到主机后如何可靠高效的进行数据传输

   2. 网络编程中的两个要素:

      ip 和 端口号

      提供网络通信协议。TCP/IP参考模型（应用层，传输层，网络层，物理+数据链路层），

1.3、Inet Adderss

ip地址：Inet Adderss

- 唯一的标识 internet 上的计算机 (通信实体)
- 本地回环地址（hostAddress）：127.0.0.1 主机名 (hostName)：localhost
- IP地址分类方式一： IPV4 IPV6
- IPV4：4个字节组成，4个0~255。大概42亿个， 30亿都在北美，亚洲4亿。2011年初已经用 尽。以点分十进制表示，如192.168.0.1
- IPV6：128位（16个字节），写成8个无符号整数，每个整数用四个十六进制位表示，数之间 用冒号 隔开，如：2001:0db8:3c4d:0015:0000:0000:1a2f:1a2b
- IP地址分类方式2：公网地址（万维网使用） 和 私有地址（局域网使用）。
- 192.168.开头的就是私有地址，范围即为 192.168.0.0 ~ 192.168.255.255，专门为组织机构 内部使用
- 【查看 JDK 帮助文档=> InetAddress类，代表IP】
- 特点：不便于记忆，我们常使用域名来访问：www.baidu.com
- https://blog.kuangstudy.com/ => DNS 域名解析（150.109.117.44）=> 现在本机的hosts文件，判断是否有输入的域名地址，没有的话，再通过DNS服务器，找主机。
- hosts文件地址： c：\windows\system32\drivers\etc\hosts

上代码：InetAddressTest

```
package com.kuang.lesson1;
import java.net.InetAddress;
import java.net.UnknownHostException;
//IP 这个东西，怎么用Java对象表示
public class InetAddressTest {
    public static void main(String[] args) {
        try {
            //获得IP地址
            InetAddress inetAddresses1 =
            InetAddress.getByName("192.168.8.123");
            System.out.println(inetAddresses1);
            InetAddress inetAddresses2 =
            InetAddress.getByName("www.baidu.com");
            System.out.println(inetAddresses2);
            //获取本地IP
            InetAddress inetAddresses3 = InetAddress.getByName("127.0.0.1");
            System.out.println(inetAddresses3);
            InetAddress inetAddresses4 = InetAddress.getByName("localhost");
            System.out.println(inetAddresses4);
            InetAddress inetAddresses5 = InetAddress.getLocalHost();
            System.out.println(inetAddresses5);
            //getHostName
            System.out.println(inetAddresses2.getHostName());
            //getHostAddress
            System.out.println(inetAddresses2.getHostAddress());
            //Canonical : 规范的
            System.out.println(inetAddresses2.getCanonicalHostName());
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

1.4、端口号

端口号标识正在计算机上运行的进程（程序）

不同的进程有不同的端口号，用来区分软件

被规定为一个16位的整数 0~65535

TCP 和 UDP 各有 65535个端口，单个协议下端口不能冲突

端口分类：

公认端口： 0~1023。被预先定义的服务通信占用端口。

- HTTP 默认端口 ： 80
- HTTPS 默认端口：443
- FTP 默认端口： 21
- Telnet 默认端口：23

注册端口：1024~49151、分配给用户进程或应用程序。

- tomcat 默认端口：8080
- Mysql 默认端口：3306
- Oracle 默认端口：1521

动态、私有端口：49152~65535

```
netstat -ano #查看所有端口
netstat -ano|findstr "6732" #查看指定端口
tasklist|findstr "6732" #查看指定进程
# 使用任务管理器查看PID
```

端口号与IP地址的组合，得出一个网络套接字：Socket，所以说一些网络编程也被称为Socket编程

```
package com.kuang.lesson1;
import java.net.InetSocketAddress;
public class InetSocketAddressTest {
    public static void main(String[] args) {
        InetSocketAddress socketAddress = new
        InetSocketAddress("127.0.0.1",8080);
        InetSocketAddress socketAddress2 = new
        InetSocketAddress("localhost",9000);
        System.out.println(socketAddress.getHostName());
        System.out.println(socketAddress.getAddress());
        System.out.println(socketAddress.getPort());
        System.out.println(socketAddress2.getHostName());
        System.out.println(socketAddress2.getAddress()); //返回地址
        System.out.println(socketAddress2.getPort()); //返回端口
    }
}
```

1.5、网络通信协议

协议：就好比我们都说的普通话，大家才能听懂我讲的，但是我们还有自己的方言！

网络通信协议：

计算机网络中实现通信必须有一些约定，即通信协议，对速率，传输代码，代码结构，传输控制步骤，出错控制等制定标准。

问题：网路协议太复杂?

计算机网络通信涉及内容很多，比如指定源地址和目标地址，加密解密，压缩解压缩，差错控制，流量控制，路由控制，如何实现如此复杂的网络协议呢?

通信协议分层的思想

在制定协议时，把复杂成份分解成一些简单的成份，再将他们复合起来。最常用的复合方式是层次方式，即同层间可以通信，上一层调用下一层，而与再下一层不发生关系。各层互不影响，利于系统的开发和扩展。

TCP/IP协议簇

传输层协议中有两个非常重要的协议：

- 用户传输协议 TCP （Transmission Control Protocol）
- 用户数据报协议（User Datagram Protocol）
- Tcp/IP 以其两个主要协议： 传输控制协议：TCP，和网络互联协议：IP，而得名，实际上是一组协议，包括多个具有不同功能且互为关联的协议。
- IP（Internet Protocol）协议是网络层的主要协议，支持网间互联的数据通信。
- TCP/IP协议模型从更实用的角度出发，形成了高效的四层体系结构，即物理链路层，IP层，传输层和应用层

TCP 和 UDP对比

TCP协议

- 使用TCP协议前，必须建立TCP连接，形成传输数据通道；
- 传输前，采用‘三次握手’方式，点对点通信，是可靠的。
- TCP协议进行通信的两个应用进程：客户端，服务端。
- 在连接中可进行大数据量的传输
- 传输完毕，需要释放已建立的连接，效率低
- 举例：打电话

UDP协议

- 将数据，源，目的封装成数据包，不需要建立连接
- 每个数据报的大小限制在64K内
- 发送方不管对方是否准备好，接收方收到也不确认，故事不可靠的
- 可以广播发送
- 发送数据结束时，无需释放资源，开销小，速度快。
- 举例：发短信，导弹（饱和攻击）

2、TCP网络编程

2.1、案例一

需求：客户端发送信息给服务端，服务端将数据显示在控制台上。

```
package com.kuang.lesson2;
import java.io.IOException;
```

```
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
//客户端
public class TcpClientDemo01 {
    public static void main(String[] args) {
        Socket socket = null;
        OutputStream os = null;
        try {
            //1. 连接服务器的地址
            InetAddress serverIP = InetAddress.getByName("127.0.0.1");
            int port = 8899;
            //2. 创建一个Socket
            socket = new Socket(serverIP,port);
            //3. 创建一个输出流，向外写东西
            os = socket.getOutputStream();
            os.write("你好,欢迎学习狂神说Java".getBytes());
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            //4. 关闭资源
            try {
                if (os!=null){
                    os.close();
                }
                if (socket!=null){
                    socket.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

服务端

```
package com.kuang.lesson2;
import javax.sound.midi.Soundbank;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Base64;
//服务端
public class TcpServerDemo01 {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket accept = null;
        InputStream is = null;
        ByteArrayOutputStream baos = null;
        try {
            //1. 开放服务器端口，创建ServerSocket
```

```
            serverSocket = new ServerSocket(8899);
            //2. 等待客户端的连接
            accept = serverSocket.accept();
            //3. 读入客户端的消息,
            is = accept.getInputStream();
            /*
            回忆之前的IO流方案，弊端：存在中文，可能存在乱码。
            byte[] buffer = new byte[1024];
            int len;
            while ((len=is.read(buffer))!=-1){
            String str = new String(buffer,0,len);
            System.out.println(str);
            }
            **/
            baos = new ByteArrayOutputStream();
            byte[] buffer = new byte[1024];
            int len;
            while ((len=is.read(buffer))!=-1){
                baos.write(buffer,0,len);
            }
            System.out.println(baos.toString());
            System.out.println("数据来源地
址："+accept.getInetAddress().getHostName());
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            //4. 关闭资源
            try {
                if (baos!=null){
                    baos.close();
                }
                if (is!=null){
                    is.close();
                }
                if (accept!=null){
                    accept.close();
                }
                if (serverSocket!=null){
                    serverSocket.close();
                }
            }catch (Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

2.2、案例二

需求：客户端发送文件给服务器，服务端将文件保存在本地。

```
package com.kuang.lesson3;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;
```

```
//客户端
public class TcpClientDemo02 {
    public static void main(String[] args) throws Exception{
        //1. 创建socket连接
        Socket socket = new Socket(InetAddress.getByName("127.0.0.1"),9090);
        //2. 创建一个输出流
        OutputStream os = socket.getOutputStream();
        //3. 读取文件
        FileInputStream fis = new FileInputStream(new File("qinjiang.jpg"));
        //4. 写出文件
        byte[] buffer = new byte[1024];
        int len;
        while ((len=fis.read(buffer))!=-1){
            os.write(buffer,0,len);
        }
        //5. 资源关闭,应该使用 try-catch-finally
        fis.close();
        os.close();
        socket.close();
    }
}
```

服务端：

```
package com.kuang.lesson3;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
//服务端
public class TcpServerDemo02 {
    public static void main(String[] args) throws Exception {
        //1. 开启 ServerSocket
        ServerSocket serverSocket = new ServerSocket(9090);
        //2. 侦听 客户端 Socket
        Socket socket = serverSocket.accept();
        //3. 获取输入流
        InputStream is = socket.getInputStream();
        //4. 读取接收的文件并保存
        FileOutputStream fos = new FileOutputStream(new File("receive.jpg"));
        byte[] buffer = new byte[1024];
        int len;
        while ((len=is.read(buffer))!=-1){
            fos.write(buffer,0,len);
        }
        //5.关闭资源,应该使用 try-catch-finally
        fos.close();
        is.close();
        socket.close();
        serverSocket.close();
    }
}
```

2.3、案例三

需求：我们需要在案例二的基础上，接收成功后，返回给客户端，接收成功！然后客户端才关闭连接！

```java
package com.kuang.lesson4;
import java.io.*;
import java.net.InetAddress;
import java.net.Socket;
//客户端
public class TcpClientDemo03 {
public static void main(String[] args) throws Exception{
//1. 创建socket连接
Socket socket = new Socket(InetAddress.getByName("127.0.0.1"),
9090);
//2. 创建一个输出流
OutputStream os = socket.getOutputStream();
//3. 读取文件
FileInputStream fis = new FileInputStream(new File("qinjiang.jpg"));
//4. 写出文件
byte[] buffer = new byte[1024];
int len;
while ((len=fis.read(buffer))!=-1){
os.write(buffer,0,len);
}
//告诉服务器，我传输完了,关闭数据的输出，不然就会一直阻塞!
socket.shutdownOutput();
//先别着急关，等待服务器响应，响应到控制台，注意重复的变量问题!
InputStream inputStream = socket.getInputStream();
ByteArrayOutputStream baos = new ByteArrayOutputStream();
byte[] buffer2 = new byte[1024];
int len2;
while ((len2=inputStream.read(buffer2))!=-1){
baos.write(buffer2,0,len2);
}
System.out.println(baos.toString());
//5. 资源关闭,应该使用 try-catch-finally
baos.close();
inputStream.close();
fis.close();
os.close();
socket.close();
}
}
```

```java
package com.kuang.lesson4;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
//服务端
public class TcpServerDemo03 {
public static void main(String[] args) throws Exception {
//1. 开启 ServerSocket
ServerSocket serverSocket = new ServerSocket(9090);
//2. 侦听 客户端 Socket
Socket socket = serverSocket.accept();
//3. 获取输入流
```

```
InputStream is = socket.getInputStream();
//4. 读取接收的文件并保存
FileOutputStream fos = new FileOutputStream(new
File("receive2.jpg"));
byte[] buffer = new byte[1024];
int len;
while ((len=is.read(buffer))!=-1){
fos.write(buffer,0,len);
}
//通知客户端接收成功
OutputStream outputStream = socket.getOutputStream();
outputStream.write("文件已经成功收到，OK".getBytes());
//5.关闭资源,应该使用 try-catch-finally
outputStream.close();
fos.close();
is.close();
socket.close();
serverSocket.close();
}
}
```

3、UDP网络编程

3.1、说明

DatagramSocket 和 DatagramPacket 两个类实现了基于UDP协议的网络程序。

UDP 数据报通过数据报套接字 DatagramSocket 发送和接收，系统不保证UDP数据报一定能够安 全送到目的地，也不确定什么时候可以抵达。

DatagramPacket 对象封装了UDP数据报，在数据报中包含了发送端的IP地址和端口号以及接收端 的IP地址和端口号。

UDP协议中每个数据报都给出了完整的地址信息，因此无需建立发送方和接收方的连接。如同发快 递包裹一样。

3.2、案例一

```
package com.kuang.lesson5;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class UDPSenderDemo01 {
    public static void main(String[] args) throws Exception {
        //1. 建立DatagramSocket
        DatagramSocket socket = new DatagramSocket();
        //2. 封装数据包
        String msg = "UDPSender==>";
        byte[] data = msg.getBytes();
        InetAddress inet = InetAddress.getByName("127.0.0.1");
        int port = 9090;
        DatagramPacket packet = new
        DatagramPacket(data,0,data.length,inet,port);
        //3. 通过 Socket 发送 packet
        socket.send(packet);
        //4. 关闭socket
```

```
        socket.close();
    }
}
```

接收方

```
package com.kuang.lesson5;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class UDPReceiverDemo01 {
    public static void main(String[] args) throws Exception {
        //1. 建立DatagramSocket,开放端口
        DatagramSocket socket = new DatagramSocket(9090);
        //2. 接收数据
        byte[] buffer = new byte[1024];
        DatagramPacket packet = new DatagramPacket(buffer,0,buffer.length);
        socket.receive(packet);
        //3. 输出数据
        // packet.getData() : 获取packet中的数据
        System.out.println(new String(packet.getData(), 0,packet.getLength()));
        //4. 关闭socket
        socket.close();
    }
}
```

测试： 启动接收方，启动发送方，查看是否能够接收到！

3.3、案例二：在线咨询

```
package com.kuang.udpchat;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetSocketAddress;
public class UdpTalkClient {
    public static void main(String[] args) throws Exception {
        System.out.println("发送方启动中....");
        //1. 使用DatagramSocket 指定端口，创建发送端
        DatagramSocket socket = new DatagramSocket(8888);
        //2. 准备数据，转成字节数组
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
        while (true){
            String data = reader.readLine();
            byte[] datas = data.getBytes();
            //3. 封装成DatagramPacket包裹，需要指定目的地
            DatagramPacket packet = new DatagramPacket(datas,0,datas.length,
            new InetSocketAddress("localhost",6666));
            //4. 发送包裹send
            socket.send(packet);
            //退出判断
            if (data.equals("bye")){
                break;
            }
        }
```

```
        //5. 释放资源
        socket.close();
    }
}
```

服务端

```
package com.kuang.udpchat;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class UdpTalkServer {
public static void main(String[] args) throws Exception {
DatagramSocket socket = new DatagramSocket(6666);
while (true) {
try {
//准备接收包裹;
byte[] container = new byte[1024];
DatagramPacket packet = new DatagramPacket(container, 0,
container.length);
socket.receive(packet); //阻塞式接收包裹
byte[] datas = packet.getData();
int len = packet.getLength();
String data = new String(datas,0,len);
System.out.println(data);
    //退出判断
if (data.equals("bye")){
break;
}
}catch (Exception e){
e.printStackTrace();
}
}
socket.close();
}
}
```

问题：现在需要两遍需要接受和发送，我们可以使用多线程来解决！

发送端多线程

```
package com.kuang.udpchat;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetSocketAddress;
import java.net.SocketException;
//发送端
public class TalkSend implements Runnable {
    private DatagramSocket socket;
    private BufferedReader reader;
    private String toIP;
    private int toPort;
    public TalkSend(int port,String toIP,int toPort) {
        this.toIP = toIP;
        this.toPort = toPort;
        try {
```

```
            socket = new DatagramSocket(port);
            reader = new BufferedReader(new InputStreamReader(System.in));
        } catch (SocketException e) {
            e.printStackTrace();
        }
    }
@Override
    public void run() {
        while (true){
            try {
                String data = reader.readLine();
                byte[] datas = data.getBytes();
                //3. 封装成DatagramPacket包裹，需要指定目的地
                DatagramPacket packet = new
DatagramPacket(datas,0,datas.length,new
InetSocketAddress(this.toIP,this.toPort));
                //4. 发送包裹send
                socket.send(packet);
                //退出判断
                if (data.equals("bye")){
                    break;
                }
            }catch (Exception e){
                e.printStackTrace();
            }
        }
        //5. 释放资源
        socket.close();
    }
}
```

接收端多线程

```
package com.kuang.udpchat;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
//接收端
public class TalkReceive implements Runnable {
    private DatagramSocket socket;
    private String msgFrom;
    public TalkReceive(int port,String msgFrom) {
        this.msgFrom = msgFrom;
        try {
            socket = new DatagramSocket(port);
        } catch (SocketException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void run() {
        while (true) {
            try {
                //准备接收包裹;
                byte[] container = new byte[1024];
                DatagramPacket packet = new DatagramPacket(container,
0,container.length);
```

```
                socket.receive(packet); //阻塞式接收包裹
                byte[] datas = packet.getData();
                int len = packet.getLength();
                String data = new String(datas,0,len);
                System.out.println(msgFrom+":"+data);
                //退出判断
                if (data.equals("bye")){
                    break;
                }
            }catch (Exception e){
                e.printStackTrace();
            }
        }
        socket.close();
    }
}
```

学生端

```
package com.kuang.udpchat;
public class TalkStudent {
    public static void main(String[] args) {
        new Thread(new TalkSend(7777,"localhost",9999)).start();
        new Thread(new TalkReceive(8888,"老师")).start();
    }
}
```

老师端

```
package com.kuang.udpchat;
public class TalkTeacher {
    public static void main(String[] args) {
        new Thread(new TalkReceive(9999,"学生")).start();
        new Thread(new TalkSend(5555,"localhost",8888)).start();
    }
}
```

4、URL编程

4.1、url类

URL （Uniform Resource Locator）： 统一资源定位符，它表示 internet 上某一资源的地址。

它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate：定位这个资源。

通过URL 我们可以访问Internet上的各种网络资源，比如最常见的 www,ftp站点。

浏览器通过解析 给定的URL可以在网络上查找相应的文件或其他资源。 URL 的 基本结构由 5部分组成：

传输协议：//主机名：端口号/文件名 #片段名？参数列表

例如：http://localhost:8080/helloworld/index.jsp#a?username=kuangshen&password=123

片段名，即锚链接，比如我们去一些小说网站，可以直接定位到某个章节位置

参数列表格式： 参数名=参数值 & 参数名=参数值...

实例化

```
package com.kuang.lesson6;
import java.net.MalformedURLException;
import java.net.URL;
public class URLDemo01 {
    public static void main(String[] args) {
        try {
            URL url = new URL("http://localhost:8080/helloworld/index.jsp?
username=kuangshen&password=123");
            System.out.println(url.getProtocol()); //获取URL的协议名
            System.out.println(url.getHost()); //获取URL的主机名
            System.out.println(url.getPort()); //获取URL的端口号
            System.out.println(url.getPath()); //获取URL的文件路径
            System.out.println(url.getFile()); //获取URL的文件名
            System.out.println(url.getQuery()); //获取URL的查询名
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

4.2、下载tomcat下的文件

首先需要在tomcat中放入一个资源文件!

```
package com.kuang.lesson6;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
public class URLDemo02 {
    public static void main(String[] args) {
        try {
            //1. 定位到服务器端的资源
            URL url = new URL("http://localhost:8080/helloworld/qinjiang.jpg");
            //2. 创建连接
            HttpURLConnection connection =
(HttpURLConnection)url.openConnection();
            //3. 获取输入流
            InputStream is = connection.getInputStream();
            //4. 写出文件
            FileOutputStream fos = new FileOutputStream("qinjiang2.jpg");
            byte[] buffer = new byte[1024];
            int len;
            while ((len=is.read(buffer))!=-1){
                fos.write(buffer,0,len);
            }
            //关闭资源
            fos.close();
            is.close();
            connection.disconnect();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

```
}
```