# 尚马教育 JAVA 基础课程

# Socket

文档编号：A17

创建日期： 2017-04-12

最后修改日期：2019-09-23

版 本 号：V3.0

电子版文件名：尚马教育-第一阶段-17.Socket专题课程.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Java基础版本更新 | V2.0 |
| 2019-08-09 | 徐丽莎 | Java基础版本更新 | V3.0 |

**主讲人：**

**徐丽莎**

目录

[尚马教育 JAVA 基础课程 1](#_Toc2129)

[Socket 1](#_Toc18171)

[1. 计算机网络 2](#_Toc31085)

[1.1. 概念 2](#_Toc17526)

[1.2. 功能 2](#_Toc29481)

[1.3. 分类 2](#_Toc21208)

[1.4. 工作模式 3](#_Toc24289)

[2. 网络编程 3](#_Toc11832)

[2.1. 组成部分 3](#_Toc18388)

[2.1.1. 网络通讯协议 3](#_Toc32334)

[2.1.2. ip地址 4](#_Toc16999)

[2.1.3. 端口 5](#_Toc11062)

[2.2. 七层模型(osi) 5](#_Toc20073)

[3. 常用类 6](#_Toc5405)

[3.1. InetAddress 6](#_Toc5255)

[3.2. URL 7](#_Toc11880)

[4. 基于TCP/IP的Socket编程 10](#_Toc8274)

[4.1. 数据传输 11](#_Toc13345)

[4.2. 一对一聊天 13](#_Toc898)

## 计算机网络

### 概念

* 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。

### 功能

* 使用远程资源
* 共享信息、程序和数据
* 分布处理

### 分类

* 按照网络规模和范围可以简单划分为：
  + 局域网（LAN，Local Area Network）
  + 城域网（MAN，Metropolis Area Network）
  + 广域网（WAN，Wide Area Network）
* 照网络拓扑结构可以分为
  + 星型网络、 总线网络、 环线网络、 树型网络、 星型环线网络等
* 按照传输介质划分
  + 双绞线、 同轴电缆、 光导纤维、 视线介质

### 工作模式

* 专用服务器结构(Server-Based) b/S
  + 又称为“工作站／文件服务器”结构，由若干台微机工作站与一台或多台文件服务器通过通信线路连接起来组成工作站存取服务器文件，共享存储设备。
* 客户机/服务器模式(client／server)
  + 其中一台或几台较大的计算机集中进行共享数据库的管理和存取，称为服务器，而将其它的应用处理工作分散到网络中其它微机上去做，构成分布式的处理系统。
* 对等式网络：（Peer-to-Peer)
  + 在对等式网络结构中，没有专用服务器 每一个工作站既可以起客户机作用也可以起服务器作用。

B/S: broswer/server

C/S: client/server pc端应用程序

## 网络编程 Socket

* 就是用来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

互联网协议: internet protocol IP

### 组成部分

#### 网络通讯协议

* 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。常用的传输协议有 UDP， **TCP**协议。其他协议：
* TCP/IP协议
  + TCP （Transmission Control Protocol，传输控制协议）
  + IP （Internet Protocol，网际协议）
* HTTP协议: https (3次握手/4次握手)---> 请求和响应
  + HTTP（Hypertext Transfer Protocol，超文本传输协议）
* FTP协议: Linux
  + FTP（File Transfer Protocol，文件传输协议）
* SMTP协议：第三阶段
  + SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）
* POP3/IMAP协议
  + POP3（Post Office Protocol - Version 3， 邮局协议版本3）
  + IMAP（Internet Message Access Protocol，Internet消息访问协议）
* 通信协议的目的就是实现数据的交换，但是需要合理的通信结构。
* 网络通信结构：为了使两个结点之间能进行对话，必须在它们之间建立通信工具(即接口)，使彼此之间 能进行信息交换。
* UDP user datagrame 用户数据报协议

|  |
| --- |
|  |

* TCP/IP是目前世界上应用最为广泛的协议 ，是以TCP和IP为基础的不同层次上多个协议的集合 ，也称：TCP/IP协议族 或 TCP/IP协议栈

#### ip地址

* 每个设备在网络中的唯一标识
* 每台网络终端在网络中都有一个独立的地址，我们在网络中传输数据就是使用这个地址。
* IP地址格式：255.255.255.255
* 目前版本是第四版(Version4/Version6)

#### 端口

* 端口是一种抽象的软件结构（包括一些数据结构和I/O缓冲区）。应用程序通过系统调用与某端口建立连接（binding）后，传输层传给该端口的数据都被相应的进程所接收，相应进程发给传输层的数据都通过该端口输出.
* 端口用一个整数型标识符来表示，即端口号。端口号跟协议相关，TCP/IP传输层的两个协议TCP和UDP是完全独立的两个软件模块，因此各自的端口号也相互独立，端口通常称为协议端口(protocol port) ，简称端口。
* 端口使用一个16位的数字来表示，它的范围是 0~65536，1024以下的端口号保留给预定义的服务。例如：http使用80端口,ftp使用21
* 端口与ip的结合，我们称为域名。

### 七层模型(osi)

* 由于节点之间联系很复杂，在制定协议时，把复杂成分分解成一些简单的成分，再将它们复合起来。最常用的复合方式是层次方式，即同层间可以通信、上一层可以调用下一层，而与下一层不发生关系。
* 通信协议的分层规定：
  + 把用户应用程序作为最高层，把物理通信作为最低层，将其间的协议处理分为若干层，规定每层的处理任务，也规定每层的接口标准。
* 七层模型包含：

|  |
| --- |
|  |
|  |

* TCP协议位于OSI参考模型的传输层。
* IP协议位于OSI参考模型的网络层。

## 常用类

### InetAddress

* 此类表示Internet协议（IP）地址。
* Java.net.InetAddress/Inet4Address/Inet6Address
* 常用方法：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 方法 | 描述 | | public static InetAddress getLocalHost() | 获得存有本机IP的InetAddress对象 | | public static InetAddress getByName(String host) | 获得存有其他电脑IP地址的InetAddress对象 | | public String getHostName() | 从InetAddress对象中获得主机名 | | public String getHostAddress() | 从InetAddress对象中获得IP地址 | |

* 案例：

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  //获得本机的ip  InetAddress address = InetAddress.getLocalHost();//Lisa/192.168.13.40  System.out.println(address);  System.out.println(address.getHostAddress());  System.out.println(address.getHostName());  System.out.println(Arrays.toString(address.getAddress()));  //获得指定计算机ip地址  address = InetAddress.getByName("JAVA-20191122RY");  System.out.println(address);    } **catch** (UnknownHostException e) {  e.printStackTrace();  }  } |

### URL

* 统一资源定位符. 指向万维网上的“资源”的指针
* 常用方法：

|  |
| --- |
| **public** **static** **void** main(String[] args) {  String path ="https://www.javasm.cn/test/a/index.html?name=jim&pass=1234#abc";  **try** {  URL url = **new** URL(path);  System.***out***.println(url.getAuthority());//授权内容www.javasm.cn  System.***out***.println(url.getProtocol());//协议  System.***out***.println(url.getPort());//端口 -1  System.***out***.println(url.getFile());//test/a/index.html?name=jim&pass=1234  System.***out***.println(url.getHost());//ip www.javasm.cn  System.***out***.println(url.getPath());//test/a/index.html  System.***out***.println(url.getQuery());//?name=jim&pass=1234  System.***out***.println(url.getRef());//#abc 锚记  System.***out***.println(url.getUserInfo());//null  System.***out***.println(url.getContent());//资源  } **catch** (MalformedURLException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **public** **static** **void** main(String[] args) {  String path ="https://www.javasm.cn/";//TCP/IP IO  **try** {  //调用接口(服务)  URL url = **new** URL(path);  //抓取一些数据  //读到 一些数据 --->输入流  URLConnection connection = url.openConnection();  connection.connect();  InputStream inputStream = connection.getInputStream();//数据目前都在inputStream里面  BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(inputStream));  String content = "";  **while**((content = bufferedReader.readLine())!=**null**) {  System.***out***.println(content);  }  } **catch** (MalformedURLException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **public** **static** **void** main(String[] args) {  String path ="https://www.javasm.cn/";//TCP/IP IO  path = "http://book.zongheng.com/chapter/892247/58415869.html";// 编码处理  path = "https://read.qidian.com/chapter/g3v\_1pKbJVjMQzCecGvf7A2/3ZPfZz0fjVT4p8iEw--PPw2";  path = "http://t.weather.sojson.com/api/weather/city/101010100";  **try** {  //调用接口(服务)  //HttpClient  URL url = **new** URL(path);  //抓取一些数据  //读到 一些数据 --->输入流  URLConnection connection = url.openConnection();  connection.connect();  InputStream inputStream = connection.getInputStream();//数据目前都在inputStream里面  BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(inputStream));  String content = "";  **while**((content = bufferedReader.readLine())!=**null**) {  System.***out***.println(content);  }  } **catch** (MalformedURLException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

## 基于TCP/IP的Socket编程

* 两个进程间可以通过一个双向的网络通信连接实现数据交换，这种通信链路的端点被称为“套接字”（Socket）。
* Socket通常用来实现Client-Server连接。
* 建立连接时所需的寻址信息
  + 远程计算机的机器名或IP地址
  + 试图连接的端口号（Port number）
* Java.net包中定义的两个类Socket和ServerSocket,分别用来实现双向链接的client和server端。

|  |
| --- |
|  |

### 数据传输

|  |
| --- |
| **public** **class** Server {  **public** **static** **void** main(String[] args) {  **try** {  //1.在本机创建ServerSocket 并监听6666端口  ServerSocket serverSocket = **new** ServerSocket(6666);//0-65536 0-1023  System.***out***.println("服务器已经启动。。。。。");  **while**(**true**) {  //2.等待客户端的连接  Socket scoket = serverSocket.accept();//接收客户端的连接 scoket客户端对象  System.***out***.println("欢迎你:"+scoket.getInetAddress());  //3.数据的交互 服务端给客户端发送数据 ==>将数据写给客户端 write 输出流  DataOutputStream dataOutputStream = **new** DataOutputStream(scoket.getOutputStream());  dataOutputStream.writeUTF("服务器: 欢迎你，客户端");  //4.服务器读取客户端写的数据  DataInputStream dataInputStream = **new** DataInputStream(scoket.getInputStream());  System.***out***.println(dataInputStream.readUTF());  }  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |
| **public** **class** Client {  **public** **static** **void** main(String[] args) {  **try** {  //1.创建客户端的Socket  //指定连接哪一台主机端口是多少的应用程序  Socket socket = **new** Socket("192.168.13.40", 6666);  // Socket socket = new Socket("127.0.0.1", 6666);  // Socket socket = new Socket("localhost", 6666);  //localhost是127.0.0.1映射出来的域名。C:\Windows\System32\drivers\etc\hosts  //2.客户端读取服务端发送的数据  DataInputStream dataInputStream = **new** DataInputStream(socket.getInputStream());  System.***out***.println(dataInputStream.readUTF());  //3.客户端给服务端写数据  DataOutputStream dataOutputStream = **new** DataOutputStream(socket.getOutputStream());  dataOutputStream.writeUTF("客户端: 你好，服务器");  } **catch** (IOException e) {  e.printStackTrace();  }**finally** {  //流关闭  //socket关闭  }  }  } |

### 一对一聊天

|  |
| --- |
| **public** **class** ServerDemo {  **private** **static** **final** String ***SERVER\_NAME*** = "server:";  **public** **static** **void** main(String[] args) {  ServerSocket serverSocket = **null**;  **try** {  serverSocket = **new** ServerSocket(8888);  System.***out***.println("服务器已经启动。。。。。");  **while**(**true**) {  Socket socket = serverSocket.accept();//客户端  **new** ReadMsgThread(socket).start();  **new** WriteMsgThread(socket,***SERVER\_NAME***).start();  }  } **catch** (IOException e) {  e.printStackTrace();  }**finally** {  **try** {  **if**(serverSocket!=**null**) serverSocket.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |
| **public** **class** ClientDemo {  **private** **static** **final** String ***CLINET\_NAME*** = "client:";  **public** **static** **void** main(String[] args) {  **try** {  Socket socket = **new** Socket("localhost", 8888);  //循环的读写数据  **new** ReadMsgThread(socket).start();  **new** WriteMsgThread(socket,***CLINET\_NAME***).start();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |
| **public** **class** ReadMsgThread **extends** Thread{  **private** Socket socket;//null  @Override  **public** **void** run() {  **while**(**true**) {  **try** {  //读取数据  DataInputStream dataInputStream = **new** DataInputStream(socket.getInputStream());  String str = dataInputStream.readUTF();  **if**("bye".equals(str)) {  System.***out***.println("客户端已经关闭");  **break**;  }  System.***out***.println(str);  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  **public** ReadMsgThread(Socket socket) {  **this**.socket = socket;  }  } |
| **public** **class** WriteMsgThread **extends** Thread{  **private** Socket socket;//null  **private** String name;  **private** Scanner input;  @Override  **public** **void** run() {  input = **new** Scanner(System.***in***);  **while**(**true**) {  **try** {  DataOutputStream dataOutputStream = **new** DataOutputStream(socket.getOutputStream());  String str = input.nextLine();  **if**("bye".equals(str)) {  **throw** **new** Exception();  }  dataOutputStream.writeUTF(name+str);  } **catch** (IOException e) {  } **catch** (Exception e) {  //关闭socket  **try** {  System.***out***.println("程序已经关闭了");  socket.close();  } **catch** (IOException e1) {  e1.printStackTrace();  }  }  }  }  **public** WriteMsgThread(Socket socket,String name) {  **this**.socket = socket;  **this**.name = name;  }  } |