第11天 面向网络编程

今日内容介绍

* 网络编程概述
* UDP
* TCP

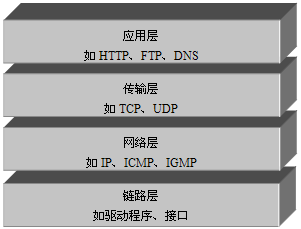
# 网络编程概述

## 网络协议

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。

网络通信协议有很多种，目前应用最广泛的是TCP/IP协议(Transmission Control Protocal/Internet Protoal传输控制协议/英特网互联协议)，它是一个包括TCP协议和IP协议，UDP（User Datagram Protocol）协议和其它一些协议的协议组，在学习具体协议之前首先了解一下TCP/IP协议组的层次结构。

在进行数据传输时，要求发送的数据与收到的数据完全一样，这时，就需要在原有的数据上添加很多信息，以保证数据在传输过程中数据格式完全一致。TCP/IP协议的层次结构比较简单，共分为四层，如图所示。



1. TCP/IP网络模型

上图中，TCP/IP协议中的四层分别是应用层、传输层、网络层和链路层，每层分别负责不同的通信功能，接下来针对这四层进行详细地讲解。

链路层：[链路层](http://zhidao.baidu.com/search?word=%CA%FD%BE%DD%C1%B4%C2%B7%B2%E3&fr=qb_search_exp&ie=gbk)是用于定义物理传输通道，通常是对某些网络连接设备的驱动协议，例如针对光纤、[网线](http://zhidao.baidu.com/search?word=%CB%AB%BD%CA%CF%DF&fr=qb_search_exp&ie=gbk)提供的驱动。

网络层：网络层是整个TCP/IP协议的核心，它主要用于将传输的数据进行分组，将分组数据发送到目标计算机或者网络。

传输层：主要使网络程序进行通信，在进行网络通信时，可以采用TCP协议，也可以采用UDP协议。

应用层：主要负责应用程序的协议，例如HTTP协议、FTP协议等。

## IP地址和端口号

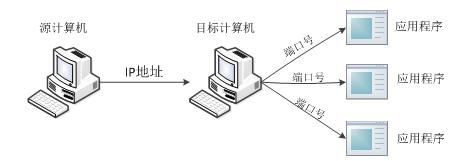
要想使网络中的计算机能够进行通信，必须为每台计算机指定一个标识号，通过这个标识号来指定接受数据的计算机或者发送数据的计算机。

在TCP/IP协议中，这个标识号就是IP地址，它可以唯一标识一台计算机，目前，IP地址广泛使用的版本是IPv4，它是由4个字节大小的二进制数来表示，如：00001010000000000000000000000001。由于二进制形式表示的IP地址非常不便记忆和处理，因此通常会将IP地址写成十进制的形式，每个字节用一个十进制数字(0-255)表示，数字间用符号“.”分开，如 “192.168.1.100”。

随着计算机网络规模的不断扩大，对IP地址的需求也越来越多，IPV4这种用4个字节表示的IP地址面临枯竭，因此IPv6 便应运而生了，IPv6使用16个字节表示IP地址，它所拥有的地址容量约是IPv4的8×1028倍，达到2128个（算上全零的），这样就解决了网络地址资源数量不够的问题。

通过IP地址可以连接到指定计算机，但如果想访问目标计算机中的某个应用程序，还需要指定端口号。在计算机中，不同的应用程序是通过端口号区分的。端口号是用两个字节（16位的二进制数）表示的，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，用户的普通应用程序需要使用1024以上的端口号，从而避免端口号被另外一个应用或服务所占用。

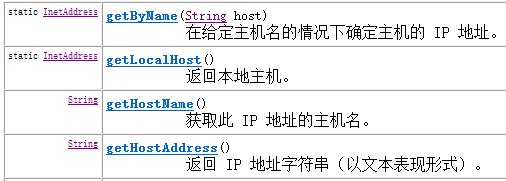
接下来通过一个图例来描述IP地址和端口号的作用，如下图所示。



从上图中可以清楚地看到，位于网络中一台计算机可以通过IP地址去访问另一台计算机，并通过端口号访问目标计算机中的某个应用程序。

## InetAddress

了解了IP地址的作用，我们看学习下JDK中提供了一个InetAdderss类，该类用于封装一个IP地址，并提供了一系列与IP地址相关的方法，下表中列出了InetAddress类的一些常用方法。



### 案例代码一:

**package** com.itheima\_01;

**import** java.net.InetAddress;

**import** java.net.UnknownHostException;

/\*

\* InetAddress:此类表示互联网协议 (IP) 地址。

\*

\*/

**public** **class** InetAddressDemo {

**public** **static** **void** main(String[] args) **throws** UnknownHostException {

//static InetAddress getByName(String host)

//InetAddress address = InetAddress.getByName("itheima");

InetAddress address = InetAddress.*getByName*("192.168.1.107");//ip地址是唯一的

//System.out.println(address);//itheima/192.168.1.107 ipconfig

String hostAddress = address.getHostAddress();//192.168.1.107 返回IP地址

String hostName = address.getHostName();//itheima 返回主机名

System.*out*.println(hostAddress);

System.*out*.println(hostName);

}

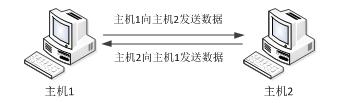
}

# UDP协议

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都使用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。UDP的交换过程如下图所示。



## DatagramPacket

前面介绍了UDP是一种面向无连接的协议，因此，在通信时发送端和接收端不用建立连接。UDP通信的过程就像是货运公司在两个码头间发送货物一样。在码头发送和接收货物时都需要使用集装箱来装载货物，UDP通信也是一样，发送和接收的数据也需要使用“集装箱”进行打包，为此JDK中提供了一个DatagramPacket类，该类的实例对象就相当于一个集装箱，用于封装UDP通信中发送或者接收的数据。

想要创建一个DatagramPacket对象，首先需要了解一下它的构造方法。在创建发送端和接收端的DatagramPacket对象时，使用的构造方法有所不同，接收端的构造方法只需要接收一个字节数组来存放接收到的数据，而发送端的构造方法不但要接收存放了发送数据的字节数组，还需要指定发送端IP地址和端口号。

接下来根据API文档的内容，对DatagramPacket的构造方法进行逐一详细地讲解。

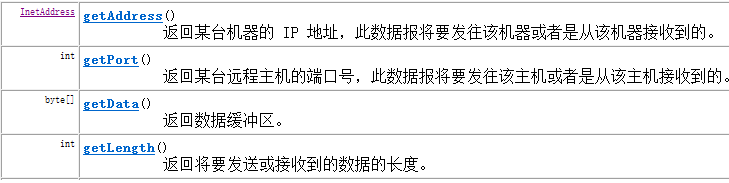


使用该构造方法在创建DatagramPacket对象时，指定了封装数据的字节数组和数据的大小，没有指定IP地址和端口号。很明显，这样的对象只能用于接收端，不能用于发送端。因为发送端一定要明确指出数据的目的地(ip地址和端口号)，而接收端不需要明确知道数据的来源，只需要接收到数据即可。



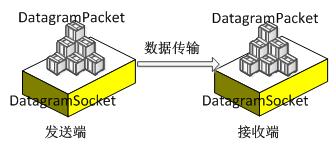
使用该构造方法在创建DatagramPacket对象时，不仅指定了封装数据的字节数组和数据的大小，还指定了数据包的目标IP地址（address）和端口号（port）。该对象通常用于发送端，因为在发送数据时必须指定接收端的IP地址和端口号，就好像发送货物的集装箱上面必须标明接收人的地址一样。

上面我们讲解了DatagramPacket的构造方法，接下来对DatagramPacket类中的常用方法进行详细地讲解，如下表所示。



## DatagramSocket

DatagramPacket数据包的作用就如同是“集装箱”，可以将发送端或者接收端的数据封装起来。然而运输货物只有“集装箱”是不够的，还需要有码头。在程序中需要实现通信只有DatagramPacket数据包也同样不行，为此JDK中提供的一个DatagramSocket类。DatagramSocket类的作用就类似于码头，使用这个类的实例对象就可以发送和接收DatagramPacket数据包，发送数据的过程如下图所示。



在创建发送端和接收端的DatagramSocket对象时，使用的构造方法也有所不同，下面对DatagramSocket类中常用的构造方法进行讲解。



该构造方法用于创建发送端的DatagramSocket对象，在创建DatagramSocket对象时，并没有指定端口号，此时，系统会分配一个没有被其它网络程序所使用的端口号。



该构造方法既可用于创建接收端的DatagramSocket对象，又可以创建发送端的DatagramSocket对象，在创建接收端的DatagramSocket对象时，必须要指定一个端口号，这样就可以监听指定的端口。

上面我们讲解了DatagramSocket的构造方法，接下来对DatagramSocket类中的常用方法进行详细地讲解。



## UDP实现

### 案例代码二:

**package** com.itheima\_02;

**import** java.io.IOException;

**import** java.net.DatagramPacket;

**import** java.net.DatagramSocket;

**import** java.net.InetAddress;

**import** java.net.SocketException;

/\*

\* 使用UDP协议发送数据

创建发送端Socket对象

创建数据并打包

发送数据

释放资源

\*

\* DatagramSocket:此类表示用来发送和接收数据,基于UDP协议的

\*

\* DatagramSocket() ：创建Socket对象并随机分配端口号

\* DatagramSocket(int port) ：创建Socket对象并指定端口号

\*/

**public** **class** SendDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建发送端Socket对象

DatagramSocket ds = **new** DatagramSocket();

//创建数据并打包

/\*

\* DatagramPacket :此类表示数据报包

\* 数据 byte[]

\* 设备的地址 ip

\* 进程的地址 端口号

DatagramPacket(byte[] buf, int length, InetAddress address, int port)

\*/

String s = "hello udp,im comming!";

**byte**[] bys = s.getBytes();

**int** length = bys.length;

InetAddress address = InetAddress.*getByName*("itheima");//发送给当前设备

**int** port = 8888;

//打包

DatagramPacket dp = **new** DatagramPacket(bys,length,address,port);

//发送数据

ds.send(dp);

//释放资源

ds.close();

}

}

**package** com.itheima\_02;

**import** java.io.IOException;

**import** java.net.DatagramPacket;

**import** java.net.DatagramSocket;

**import** java.net.InetAddress;

/\*

\* 使用UDP协议接收数据

创建接收端Socket对象

接收数据

解析数据

输出数据

释放资源

\*/

**public** **class** ReceiveDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建接收端Socket对象

DatagramSocket ds = **new** DatagramSocket(8888);

//接收数据

//DatagramPacket(byte[] buf, int length)

**byte**[] bys = **new** **byte**[1024];

DatagramPacket dp = **new** DatagramPacket(bys,bys.length);

System.*out*.println(1);

ds.receive(dp);//阻塞

System.*out*.println(2);

//解析数据

//InetAddress getAddress() : 获取发送端的IP对象

InetAddress address = dp.getAddress();

//byte[] getData() ：获取接收到的数据，也可以直接使用创建包对象时的数组

**byte**[] data = dp.getData();

//int getLength() ：获取具体收到数据的长度

**int** length = dp.getLength();

//输出数据

System.*out*.println("sender ---> " + address.getHostAddress());

//System.out.println(new String(data,0,length));

System.*out*.println(**new** String(bys,0,length));

//释放资源

ds.close();

}

}

# TCP协议

TCP通信同UDP通信一样，都能实现两台计算机之间的通信，通信的两端都需要创建socket对象。

区别在于，UDP中只有发送端和接收端，不区分客户端与服务器端，计算机之间可以任意地发送数据。

而TCP通信是严格区分客户端与服务器端的，在通信时，必须先由客户端去连接服务器端才能实现通信，服务器端不可以主动连接客户端，并且服务器端程序需要事先启动，等待客户端的连接。

在JDK中提供了两个类用于实现TCP程序，一个是ServerSocket类，用于表示服务器端，一个是Socket类，用于表示客户端。

通信时，首先创建代表服务器端的ServerSocket对象，该对象相当于开启一个服务，并等待客户端的连接，然后创建代表客户端的Socket对象向服务器端发出连接请求，服务器端响应请求，两者建立连接开始通信。

## ServerSocket

通过前面的学习知道，在开发TCP程序时，首先需要创建服务器端程序。JDK的java.net包中提供了一个ServerSocket类，该类的实例对象可以实现一个服务器段的程序。通过查阅API文档可知，ServerSocket类提供了多种构造方法，接下来就对ServerSocket的构造方法进行逐一地讲解。



使用该构造方法在创建ServerSocket对象时，就可以将其绑定到一个指定的端口号上（参数port就是端口号）。

接下来学习一下ServerSocket的常用方法，如表所示。



ServerSocket对象负责监听某台计算机的某个端口号，在创建ServerSocket对象后，需要继续调用该对象的accept()方法，接收来自客户端的请求。当执行了accept()方法之后，服务器端程序会发生阻塞，直到客户端发出连接请求，accept()方法才会返回一个Scoket对象用于和客户端实现通信，程序才能继续向下执行。

## Socket

讲解了ServerSocket对象可以实现服务端程序，但只实现服务器端程序还不能完成通信，此时还需要一个客户端程序与之交互，为此JDK提供了一个Socket类，用于实现TCP客户端程序。

通过查阅API文档可知Socket类同样提供了多种构造方法，接下来就对Socket的常用构造方法进行详细讲解。



使用该构造方法在创建Socket对象时，会根据参数去连接在指定地址和端口上运行的服务器程序，其中参数host接收的是一个字符串类型的IP地址。



该方法在使用上与第二个构造方法类似，参数address用于接收一个InetAddress类型的对象，该对象用于封装一个IP地址。

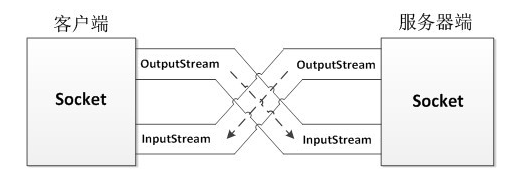
在以上Socket的构造方法中，最常用的是第一个构造方法。

接下来学习一下Socket的常用方法，如表所示。

|  |  |
| --- | --- |
| **方法声明** | **功能描述** |
| int getPort() | 该方法返回一个int类型对象，该对象是Socket对象与服务器端连接的端口号 |
| InetAddress getLocalAddress() | 该方法用于获取Socket对象绑定的本地IP地址，并将IP地址封装成InetAddress类型的对象返回 |
| void close() | 该方法用于关闭Socket连接，结束本次通信。在关闭socket之前，应将与socket相关的所有的输入/输出流全部关闭，这是因为一个良好的程序应该在执行完毕时释放所有的资源 |
| InputStream getInputStream() | 该方法返回一个InputStream类型的输入流对象，如果该对象是由服务器端的Socket返回，就用于读取客户端发送的数据，反之，用于读取服务器端发送的数据 |
| OutputStream getOutputStream() | 该方法返回一个OutputStream类型的输出流对象，如果该对象是由服务器端的Socket返回，就用于向客户端发送数据，反之，用于向服务器端发送数据 |

在Socket类的常用方法中，getInputStream()和getOutStream()方法分别用于获取输入流和输出流。当客户端和服务端建立连接后，数据是以IO流的形式进行交互的，从而实现通信。

接下来通过一张图来描述服务器端和客户端的数据传输，如下图所示。



## TCP协议实现

### 案例代码三:

**package** com.itheima\_04;

**import** java.io.IOException;

**import** java.io.OutputStream;

**import** java.net.InetAddress;

**import** java.net.Socket;

/\*

\* 使用TCP协议发送数据

创建发送端Socket对象（创建连接）

获取输出流对象

发送数据

释放资源

Socket(InetAddress address, int port)

Exception in thread "main" java.net.ConnectException: Connection refused: connect

\*/

**public** **class** ClientDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建发送端Socket对象（创建连接）

Socket s = **new** Socket(InetAddress.*getByName*("itheima"),10086);

//获取输出流对象

OutputStream os = s.getOutputStream();

//发送数据

String str = "hello tcp,im comming!!!";

os.write(str.getBytes());

//释放资源

//os.close();

s.close();

}

}

**package** com.itheima\_04;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.net.InetAddress;

**import** java.net.ServerSocket;

**import** java.net.Socket;

/\*

\* 使用TCP协议接收数据

创建接收端Socket对象

监听（阻塞）

获取输入流对象

获取数据

输出数据

释放资源

ServerSocket：接收端，服务端Socket

ServerSocket(int port)

Socket accept()

\*/

**public** **class** ServerDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建接收端Socket对象

ServerSocket ss = **new** ServerSocket(10086);

//监听（阻塞）

Socket s = ss.accept();

//获取输入流对象

InputStream is = s.getInputStream();

//获取数据

**byte**[] bys = **new** **byte**[1024];

**int** len;//用于存储读到的字节个数

len = is.read(bys);

//输出数据

InetAddress address = s.getInetAddress();

System.*out*.println("client ---> " + address.getHostName());

System.*out*.println(**new** String(bys,0,len));

//释放资源

s.close();

//ss.close();

}

}

## TCP相关案例

### 案例代码四:

使用TCP协议发送数据，服务端将接收到的数据转换成大写返回给客户端

**package** com.itheima\_05;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.InetAddress;

**import** java.net.Socket;

/\*

需求：使用TCP协议发送数据，并将接收到的数据转换成大写返回

客户端发出数据

服务端接收数据

服务端转换数据

服务端发出数据

客户端接收数据

\*/

**public** **class** ClientDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建客户端Socket对象

Socket s = **new** Socket(InetAddress.*getByName*("itheima"),10010);

//获取输出流对象

OutputStream os = s.getOutputStream();

//发出数据

os.write("tcp,im comming again!!!".getBytes());

//获取输入流对象

InputStream is = s.getInputStream();

**byte**[] bys = **new** **byte**[1024];

**int** len;//用于存储读取到的字节个数

//接收数据

len = is.read(bys);

//输出数据

System.*out*.println(**new** String(bys,0,len));

//释放资源

s.close();

}

}

**package** com.itheima\_05;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** ServerDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

//创建服务端Socket对象

ServerSocket ss = **new** ServerSocket(10010);

//监听

Socket s = ss.accept();

//获取输入流对象

InputStream is = s.getInputStream();

//获取数据

**byte**[] bys = **new** **byte**[1024];

**int** len;//用于存储读取到的字节个数

len = is.read(bys);

String str = **new** String(bys,0,len);

//输出数据

System.*out*.println(str);

//转换数据

String upperStr = str.toUpperCase();

//获取输出流对象

OutputStream os = s.getOutputStream();

//返回数据（发出数据）

os.write(upperStr.getBytes());

//释放资源

s.close();

//ss.close();//服务端一般不关闭

}

}

### 案例代码五:

客户端:

1.提示用户输入用户名和密码,将用户输入的用户名和密码发送给服务端

2.接收服务端验证完用户名和密码的结果

服务端:

1.接收客户端发送过来的用户名和密码

2.如果用户名不是itheima或者 密码不是123456,就向客户端写入”登录失败”

否则向客户端写入登录成功

package com.itheima\_06;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

/\*

\* 模拟用户登录

\*/

public class ClientTest {

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

//创建客户端Socket对象

//Socket s = new Socket(InetAddress.getByName("itheima"),8888);

Socket s = new Socket("itheima",8888);

//获取用户名和密码

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("请输入用户名:");

String username = br.readLine();

System.out.println("请输入密码:");

String password = br.readLine();

//获取输出流对象

//BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));

PrintWriter out = new PrintWriter(s.getOutputStream(),true);

//写出数据

out.println(username);

out.println(password);

//获取输入流对象

BufferedReader serverBr = new BufferedReader(new InputStreamReader(s.getInputStream()));

//获取服务器返回的数据

String result = serverBr.readLine();

System.out.println(result);

//释放资源

s.close();

}

}

package com.itheima\_06;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class ServerTest {

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

//创建服务器端Socket对象

ServerSocket ss = new ServerSocket(8888);

//监听

Socket s = ss.accept();

//获取输入流对象

BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

//获取用户名和密码

String username = br.readLine();

String password = br.readLine();

//判断用户名和密码是否正确

boolean flag = false;

if("itheima".equals(username) && "123456".equals(password)) {

flag = true;

}

//获取输出流对象

PrintWriter out = new PrintWriter(s.getOutputStream(),true);

//返回判断信息

if(flag) {

out.println("登陆成功");

}

else {

out.println("登陆失败");

}

//释放资源

s.close();

//ss.close();//服务器一般不关闭

}

}

### 案例代码六:

将用户名和密码封装到一个User类中,提供对应的构造方法和getter/setter方法

新建一个UserDB类里面定义一个集合,在集合中添加以下User对象

new User("zhangsan","123456");

new User("lisi","654321");

new User("itheima","itheima");

new User("admin","password");

客户端:

1.提示用户输入用户名和密码,将用户输入的用户名和密码发送给服务端

2.接收服务端验证完用户名和密码的结果

服务端:

* + - 1. 服务端将客户端发送过来的用户名密码封装成User对象
      2. 集合中如果包括这个User对象,想客户端写入” 登录成功”

否则向客户端写入”登录失败”

package com.itheima\_07;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

/\*

\* 模拟用户登录改写（面向对象版本）

\*/

public class ClientTest {

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

//创建客户端Socket对象

Socket s = new Socket("itheima",8888);

//获取用户名和密码

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("请输入用户名:");

String username = br.readLine();

System.out.println("请输入密码:");

String password = br.readLine();

//获取输出流对象

PrintWriter out = new PrintWriter(s.getOutputStream(),true);

//写出数据

out.println(username);

out.println(password);

//获取输入流对象

BufferedReader serverBr = new BufferedReader(new InputStreamReader(s.getInputStream()));

//获取服务器返回的数据

String result = serverBr.readLine();

System.out.println(result);

//释放资源

s.close();

}

}

package com.itheima\_07;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.List;

public class ServerTest {

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

//创建服务器端Socket对象

ServerSocket ss = new ServerSocket(8888);

//监听

Socket s = ss.accept();

//获取输入流对象

BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

//获取用户名和密码

String username = br.readLine();

String password = br.readLine();

//判断用户名和密码是否正确

boolean flag = false;

/\*if("itheima".equals(username) && "123456".equals(password)) {

flag = true;

}\*/

List<User> users = UserDB.getUsers();

User user = new User(username,password);

if(users.contains(user)) {

//匹配成功

flag = true;

}

//获取输出流对象

PrintWriter out = new PrintWriter(s.getOutputStream(),true);

//返回判断信息

if(flag) {

out.println("登陆成功");

}

else {

out.println("登陆失败");

}

//释放资源

s.close();

//ss.close();//服务器一般不关闭

}

}