Java

Socket应用---通信是这样练成的

Mikle

Published
with GitBook

Table of Contents

My Awesome Book

This file file serves as your book's preface, a great place to describe your book's content and ideas.

第1章 网络基础知识

Socket编程

1.网络的基础知识

```
两台计算机IP地址
共同语言协议
应用程序端口区分

1）TCP/TP是目前世界上应用最为广泛的的协议

是以TCP/TP为基础的不同层面上多个协议的集合

TCP:传输控制协议

IP：互联网协议

位于第四传输层

第五层：HTTP等应用层协议

2）端口：
用于区分不同应用程序端口

端口号范围0~65535，其中1~1023为系统锁保留

IP地址和端口号组成了所谓的Socket,Socket是网络上运行的程序之间双向通讯链路的终

HTTP：80
FTP:21
TELNET:23
```

2.Java中的网络支持

1.InetAddress：用于标识网络上的硬件资源
2.URL：统一资源定位符 通过URL可以直接读取或写入网络上的数据
3.Sockets：使用TCP协议实现网络通信的Socket相关的类
4.Datagram：使用UDP协议，将数据保存在数据报中，通过网络进行通信

第2章 Java 中网络相关 API 的应用

1.InetAddress：用于标识网络上的硬件资源，表示互联网协议（IP）地址。

---

package com.imooc;

import java.net.InetAddress; import java.net.UnknownHostException; import java.util.Arrays;

/*

- InetAddress类 */ public class Test01 {

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

```
// TODO Auto-generated method stub
InetAddress address=InetAddress.getLocalHost();
System.out.println("计算机名："+address.getHostName());//计算机名
System.out.println("IP地址："+address.getHostAddress());//计算机
byte[] bytes=address.getAddress();//获取字节数组形式的IP地址
System.out.println("字节数组形式的IP："+Arrays.toString(bytes));
System.out.println(address);

//根据机器名获取InetAddress实例
//InetAddress address2=InetAddress.getByName("PC201311281336")
InetAddress address2=InetAddress.getByName("192.168.1.102");
System.out.println("计算机名："+address2.getHostName());//计算机
System.out.println("IP地址："+address2.getHostAddress());//计算
```

```
}

}
```

2.URL：统一资源定位符 通过URL可以直接读取或写入网络上的数据，表示Internet上某一资源的地址

```
1)通过URL对象的openStream（）方法可以得到指定资源的输入流
2）通过输入流可以读取、访问网络上的数据
```

package com.imooc;

import java.net.MalformedURLException; import java.net.URL;

/*

- URL常用方法 */ public class Test02 {

  public static void main(String[] args) {

```
// TODO Auto-generated method stub
//创建一个URL实例
try {
    URL imooc=new URL("http://www.imooc.com");
    //？后面表示参数，#号后面表示锚点
    URL url=new URL(imooc,"/index.html?username=tom#test");
    System.out.println("协议："+url.getProtocol());
    System.out.println("主机："+url.getHost());
    //如果未指定端口号，则使用默认的端口号，此时getPort()方法返回值为-1
    System.out.println("端口："+url.getPort());
    System.out.println("文件路径："+url.getPath());
    System.out.println("文件名："+url.getFile());
    System.out.println("相对路径："+url.getRef());
    System.out.println("查询字符串："+url.getQuery());

} catch (MalformedURLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

  }

  }

package com.imooc;

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

public class Test03 {

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    try {
        //穿件一个URL实例
        URL url=new URL("http://www.baidu.com");
        //通过URL的openStream方法获取URL对象所表示的资源的字节输入流
        InputStream is=url.openStream();
        //将字节输入流转换为字符输入流
        InputStreamReader isr=new InputStreamReader(is,"UTF-8");
        //为字符输入流添加缓冲
        BufferedReader br=new BufferedReader(isr);
        String data=br.readLine();//读取数据
        while(data!=null){//循环读取数据
            System.out.println(data);//输出数据
            data=br.readLine();
        }
        br.close();
        isr.close();
        is.close();
    } catch (MalformedURLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

}
```

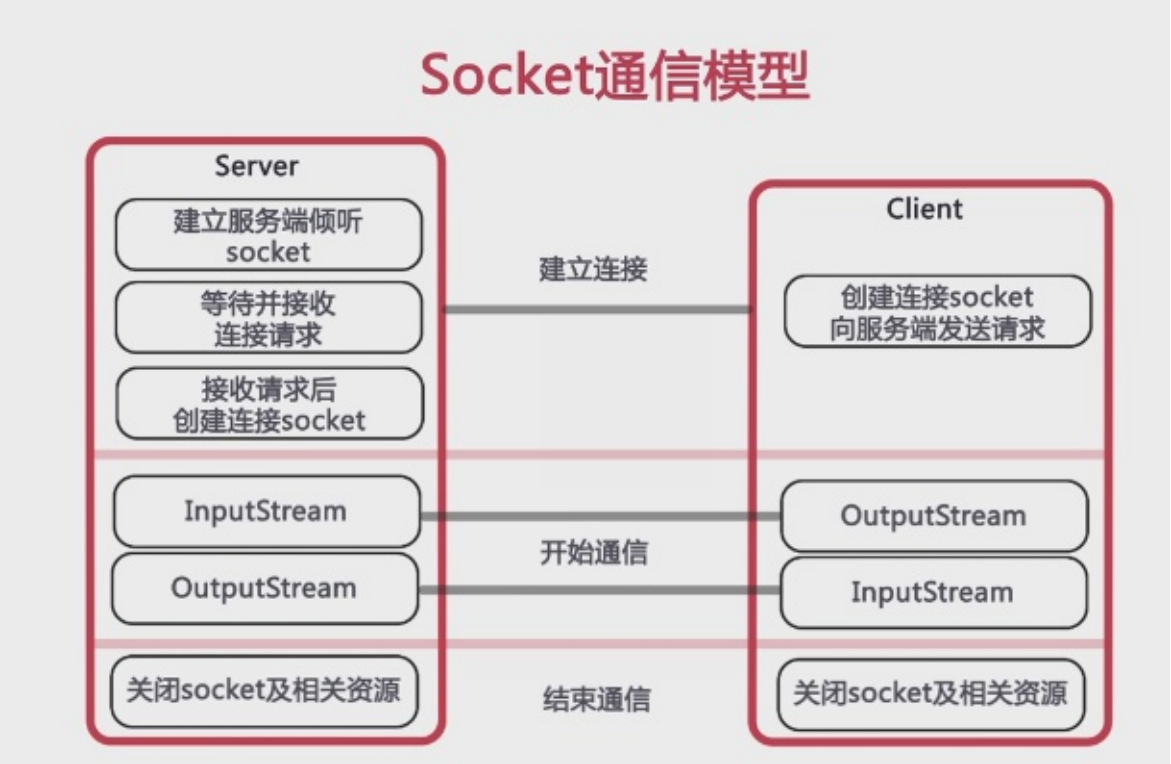
第三章 通过Socket实现TCP编程

Socket通信

1）TCP协议是面向连接、可靠的、有序的，以字节流的方式发送数据

2）基于TCP协议实现网络通信的类

```
客户端的Socket类
服务器端的ServerSocket类
```



3)Socket通信实现步骤 1.创建ServerSocket和Socket 2.打开连接到Socket的输入、输出流 3.按照协议对Socket进行读、写操作 4.关闭输入输出流、关闭Socket

编程实例：用户登录

1）服务端：

```
1.创建ServerSocket对象，绑定监听端口
2.通过accept（）方法监听客户端请求
3.连接建立后，通过输入流读取客户端发送的请求信息
4.通过输出流向客户端发送响应信息
5.关闭相关资源
```

---

package com.imooc;

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.io.OutputStream; import java.io.PrintWriter; import java.net.ServerSocket; import java.net.Socket;

/*

- 基于TCP协议的Socket通信，实现用户登录
- 服务端 */ public class Server { public static void main(String[] args) {

```
try {
    //1.创建服务器Socket，即ServerSocket,制定绑定的端口，并监听此端口
    ServerSocket serversocket=new ServerSocket(8888);
    //2.调用accept（）方法开始监听，等待客户端连接
    System.out.println("*****服务器即将启动，等待客户端的连接*****");
    Socket socket=serversocket.accept();
    //3.获取输入流，并读取客户端信息
    InputStream is=socket.getInputStream();//字节输入流
    InputStreamReader isr=new InputStreamReader(is);//将字节流转
    BufferedReader br=new BufferedReader(isr);//为输入流添加缓冲
    String info=null;
    while((info=br.readLine())!=null){//循环读取客户端的信息
        System.out.println("我是服务器，客户端说："+info);
    }
    socket.shutdownInput();//关闭输入流
    //4.获取输出流，响应客户端的请求
    OutputStream os=socket.getOutputStream();
    PrintWriter pw=new PrintWriter(os);
    pw.write("欢迎你！");
    pw.flush();//调用flush()方法将缓存输出

    //5.关闭资源
    pw.close();
    os.close();
    br.close();
    isr.close();
    is.close();
    socket.close();
    serversocket.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
}
```

2）客户端：

```
1.创建Socket对象，指明需要连接的服务端的地址和端口号
2.连接建立后，通过输出流向服务器端发送请求信息
3.通过输入流获取服务端相应的信息
4.关闭相关资源
```

---

package com.imooc;

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.io.OutputStream; import java.io.PrintWriter; import java.net.Socket; import java.net.UnknownHostException;

/*

- 客户端 */ public class Client { public static void main(String[] args) {

```
try {
    //1.创建客户端Socket，制定服务器地址和端口
    Socket socket=new Socket("localhost",8888);
    //2.获取输出流，向服务器发送信息
    OutputStream os=socket.getOutputStream();//字节输出流
    PrintWriter pw=new PrintWriter(os);//将输出流包装为打印流
    pw.write("用户名：admin；密码：123");
    pw.flush();
    socket.shutdownOutput();//关闭输出流
    //3.获取输入流，并读取服务器断的响应信息
    InputStream is=socket.getInputStream();
    BufferedReader br=new BufferedReader(new InputStreamReader
    String info=null;
    while((info=br.readLine())!=null){//循环读取客户端的信息
        System.out.println("我是客户端，服务器说："+info);
    }

    //4.关闭资源
    br.close();
    is.close();
    pw.close();
    os.close();
    socket.close();

} catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

}}

多线程服务器

应用多线程来实现服务器与多客户端之间的通信

基本步骤：

```
1.服务器端创建ServerSocket，循环调用accept（）等待客户端连接

2.客户端创建一个socket并请求和服务器端连接

3.服务器端接受客户端请求，创建socket与该客户端建立专线连接

4.建立连接的两个socket在一个单独的线程上对话

5.服务器端继续等待新的连接
```

---

package com.imooc;

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.io.OutputStream; import java.io.PrintWriter; import java.net.ServerSocket; import java.net.Socket;

/*

- 基于TCP协议的Socket通信，实现用户登录
- 服务端 */ public class Server { public static void main(String[] args) {

```
try {
    //1.创建服务器Socket，即ServerSocket,制定绑定的端口，并监听此端口
    ServerSocket serversocket=new ServerSocket(8888);
    Socket socket=null;
    //记录客户端数量
    int count=0;
    System.out.println("*****服务器即将启动，等待客户端的连接*****");
    while(true){
        //2.调用accept（）方法开始监听，等待客户端连接
        socket=serversocket.accept();
        //创建一个新线程
        ServerThread serverThread=new ServerThread(socket);
        //启动线程
        serverThread.start();
        count++;
        System.out.println("客户端数量："+count);
        //获取客户端IP
        InetAddress address=socket.getInetAddress();
        System.out.println("当前客户端的IP："+address.getHostAdd
    }
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

}}

---

package com.imooc;

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.io.OutputStream; import java.io.PrintWriter; import java.net.Socket; import java.net.UnknownHostException;

/*

- 客户端 */ public class Client { public static void main(String[] args) {

```
try {
    //1.创建客户端Socket，制定服务器地址和端口
    Socket socket=new Socket("localhost",8888);
    //2.获取输出流，向服务器发送信息
    OutputStream os=socket.getOutputStream();//字节输出流
    PrintWriter pw=new PrintWriter(os);//将输出流包装为打印流
    pw.write("用户名：admin；密码：123");
    pw.flush();
    socket.shutdownOutput();//关闭输出流
    //3.获取输入流，并读取服务器断的响应信息
    InputStream is=socket.getInputStream();
    BufferedReader br=new BufferedReader(new InputStreamReader
    String info=null;
    while((info=br.readLine())!=null){//循环读取客户端的信息
        System.out.println("我是客户端，服务器说："+info);
    }

    //4.关闭资源
    br.close();
    is.close();
    pw.close();
    os.close();
    socket.close();

} catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

}}

package com.imooc;

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.io.OutputStream; import java.io.PrintWriter; import java.net.Socket;

/*

- 服务端线程处理类 */ public class ServerThread extends Thread { //和本线程相关的Socket Socket socket=null;

  public ServerThread(Socket socket){

  ```
  this.socket=socket;
  ```

  }

  //线程执行的操作，响应客户端请求 public void run(){

  ```
  //3.获取输入流，并读取客户端信息
  InputStream is=null;
  InputStreamReader isr = null;
  BufferedReader br = null;
  OutputStream os = null;
  PrintWriter pw = null;
  try {
      is = socket.getInputStream();
      //字节输入流
      isr=new InputStreamReader(is);//将字节流转换成字符流
      br=new BufferedReader(isr);//为输入流添加缓冲
      String info=null;
      while((info=br.readLine())!=null){//循环读取客户端的信息
          System.out.println("我是服务器，客户端说："+info);
      }
      socket.shutdownInput();//关闭输入流
      //4.获取输出流，响应客户端的请求
      os=socket.getOutputStream();
      pw=new PrintWriter(os);
      pw.write("欢迎你！");
      pw.flush();//调用flush()方法将缓存输出
  } catch (IOException e) {
      // TODO Auto-generated catch block
  ```

```
            e.printStackTrace();
        }finally{
            //5.关闭资源

                try {

                    if(pw!=null)
                        pw.close();
                    if(os!=null)
                    os.close();
                    if(br!=null)
                        br.close();
                    if(isr!=null)
                        isr.close();
                    if(is!=null)
                        is.close();
                    if(socket!=null)
                        socket.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

        }
```

```
 }
```

}

第四章 通过Socket实现UDP编程

1.UDP协议（用户数据报协议）是无连接，不可靠的，无序的

进行数据传输时候，首先要把数据定义成数据报（Datagram），在数据报中指明数据要到达

的Socket（主机地址和端口号），然后将数据报发送出去

相关操作类： DatagramPacket：表示数据报包 DatagramSocket：进行端到端通信的类

2.编程实现基于UDP的Socket通信之服务器端

服务器端实现步骤：

```
1）创建DatagramSocket，指定端口号
2）创建DatagramPacket
3）接收客户端发送的数据信息
4）读取数据
```

客户端实现步骤：

```
1）定义发送信息
2）创建DatagramPacket，包含将要发送的信息
3）创建DatagramSocket
4）发送数据
```

---

package com.imooc;

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

```
/*
 * 服务器端，实现基于UDP的用户登录
 */
public class UDPSever {
public static void main(String[] args) throws IOException {

/*
 * 接受客户端发送的数据
 */
//1.创建服务器端DatagramSocket，指定端口
DatagramSocket socket=new DatagramSocket(8800);
//2.创建数据报，用于接收客户端发送的数据
byte[] data=new byte[1024];
DatagramPacket packet=new DatagramPacket(data,data.length);
//3.接受客户端发送的数据
System.out.println("****服务器端被启动****");
socket.receive(packet);//此方法在接受到数据报之前会一直处于阻塞
//4.读取数据
String info=new String(data, 0, packet.getLength());
System.out.println("我是服务器，客户端说："+info);
/*
 * 向客户端响应数据
 */
//1.定义客户端的地址，端口号，数据
InetAddress address=packet.getAddress();
int port=packet.getPort();
byte[] data2="欢迎您！".getBytes();
//2.创建数据报，包含响应的信息
DatagramPacket packet2=new DatagramPacket(data2, data2.length, addr
//3.响应客户端
socket.send(packet2);
//4.关闭资源
socket.close();
}
}
```

package com.imooc;

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 UDPClient { public static void main(String[] args) throws UnknownHostException, IOException { /*

```
     * 发送数据到服务器
     */
    //1.定义服务器的地址，端口号，数据
    InetAddress address=InetAddress.getByName("localhost");
    int port=8800;
    byte[] data="用户名：admin；密码：123".getBytes();
    //2.创建一个数据报，包含发送的数据信息
    DatagramPacket packet=new DatagramPacket(data, data.length, add
    //3.创建DatagramSocket对象
    DatagramSocket socket=new DatagramSocket();
    //4.向服务器端发送数据报
    socket.send(packet);
    /*
     * 接收服务器端响应的数据
     */
    //1.创建数据报，用于接收服务器端响应的数据
    byte[] data2=new byte[1024];
    DatagramPacket packet2=new DatagramPacket(data2, data2.length);
    //2.接收服务器响应的数据
    socket.receive(packet2);//接受数据放在packet2中
    //3.读取数据
    String reply=new String(data2,0,packet2.getLength());
    System.out.println("我是客户端，服务器说："+reply);
    //4.关闭资源
    socket.close();


}
}
```

第五章 Socket 总结

重点

```
Socket通信原理
基于TCP的Socket通信
```

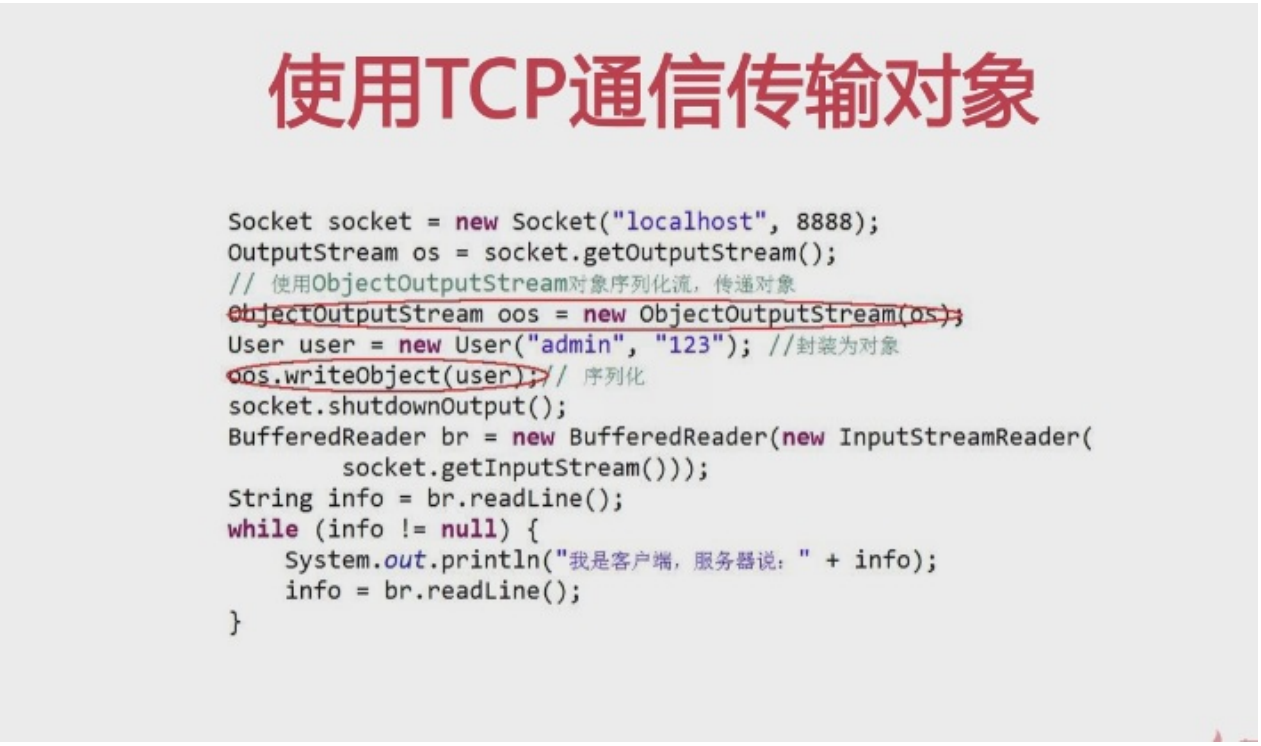
问题：

1.多线程的优先级

2.是否关闭输出流和输入流

3.使用TCP通信传输对象

传输的数据应该打包成User对象



4.Socket编程传递文件


socket编程传递文件
ObjectInputStream clientInputStream = null;
//下载后首先生成一个临时文件
String fileNameTemp = "temp.txt";
String downloadPath = ConfigManager.getInstance().getString(Constants.CLIENT_DOWNLOAD_PA
try {
clientInputStream = new ObjectInputStream(clientSocket.getInputStream());
File fileTemp = new File(downloadPath+"/"+fileNameTemp);
if (fileTemp.exists()){
fileTemp.delete();
}
fileTemp.createNewFile();
BufferedOutputStream fos = new BufferedOutputStream(new FileOutputStream(fileTemp));
//接收服务器的文件
byte[] buf = new byte[1024];
int len;
while((len = clientInputStream.read(buf)) != -1){
fos.write(buf, 0, len);
fos.flush();
}

第六章 综合练习