****

**《分布式计算》**

JAVA进阶与Socket通讯

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 数据科学与计算机学院 | | | | | |
| **成员** | **：** | 陈伟宸 | | | | | |
| **时间** | **：** | 2016 | 年 | 9 | 月 | 4 | 日 |

1. 开放式系统互联通信参考模型：

第7层：应用层

第6层：表示层

第5层：会话层

第4层：传输层

第3层：网络层

第2层：数据链路层

第1层：物理层

传输层：把传输表头(TH)加至数据以形成数据包。

网络层：决定数据的路径选择和转寄，将网络表头(NH)加至数据包，以形成分组。

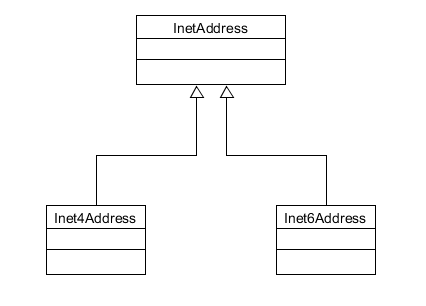
2. 会话层；

端口号用来指示传输的数据来自于远程主机或者属于本地的哪个应用程序。

3. 存在，TCP 提供按序、可靠的传输服务，与流式文件提供的服务特性相似。

4. 不相同；连接Socket用来处理客户端的连接，并创建数据Socket用于与客户端通信；通常，3位及3位以下的端口号为著名端口号，用于常见的服务如 http、ftp等，5位的端口号被用于系统临时分配，所以通常我们只定义4位数的端口号。

5.



6. （1）服务器：ServerSocket listenSocket = new ServerSocket(Integer.parseInt(args[0]));

Socket socket = listenSocket.accept();

（2）客户端：Socket socket = new Socket(args[0], Integer.parseInt(args[1]));

（3）客户端：Socket socket = new Socket(args[0], Integer.parseInt(args[1]));

服务器：Socket socket = listenSocket.accept();

（4）客户端：PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

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

服务器：PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

（5）客户端：out.close();

in.close();

socket.close();

服务器：out.close();

in.close();

socket.close();

listenSocket.close();

7. 注：编译时请使用 -Xlint:deprecation 选项

// 采用多线程方式实现的服务端程序

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class MTEchoServer {

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

List<EchoThread> threads = new ArrayList<EchoThread>();

if (args.length != 1) {

System.out.println("用法：MTServer <端口号>");

return ;

}

ServerSocket ss = new ServerSocket(Integer.parseInt(args[0]));

System.out.println("服务程序正在监听端口：" + args[0]);

for (;;) {

Socket s = ss.accept();

boolean need\_create\_new\_thread = true;

for (int i = 0; i < threads.size(); i++) {

if (threads.get(i).isSuspend == true) {

System.out.println("唤醒挂起的进程，进程号为：" + i);

threads.get(i).setSocket(s);

threads.get(i).resume();

threads.get(i).isSuspend = false;

need\_create\_new\_thread = false;

break;

}

}

if (need\_create\_new\_thread) {

System.out.println("服务器新建了一个线程");

EchoThread thread = new EchoThread(s);

thread.start();

threads.add(thread);

}

}

}

}

class EchoThread extends Thread {

Socket socket;

public boolean isSuspend = false;

EchoThread(Socket s) {

socket = s;

}

void setSocket(Socket s) {

socket = s;

}

public void run() {

while (true) {

System.out.println("正在为客户程序提供服务：" + socket);

try {

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

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

String message;

while ((message = in.readLine()) != null) {

System.out.println(socket + "请求：" + message);

out.println(message.toUpperCase());

}

out.close();

in.close();

socket.close();

}

catch (Exception exc) {

exc.printStackTrace();

}

finally {

System.out.println("进程被挂起了");

isSuspend = true;

suspend();

}

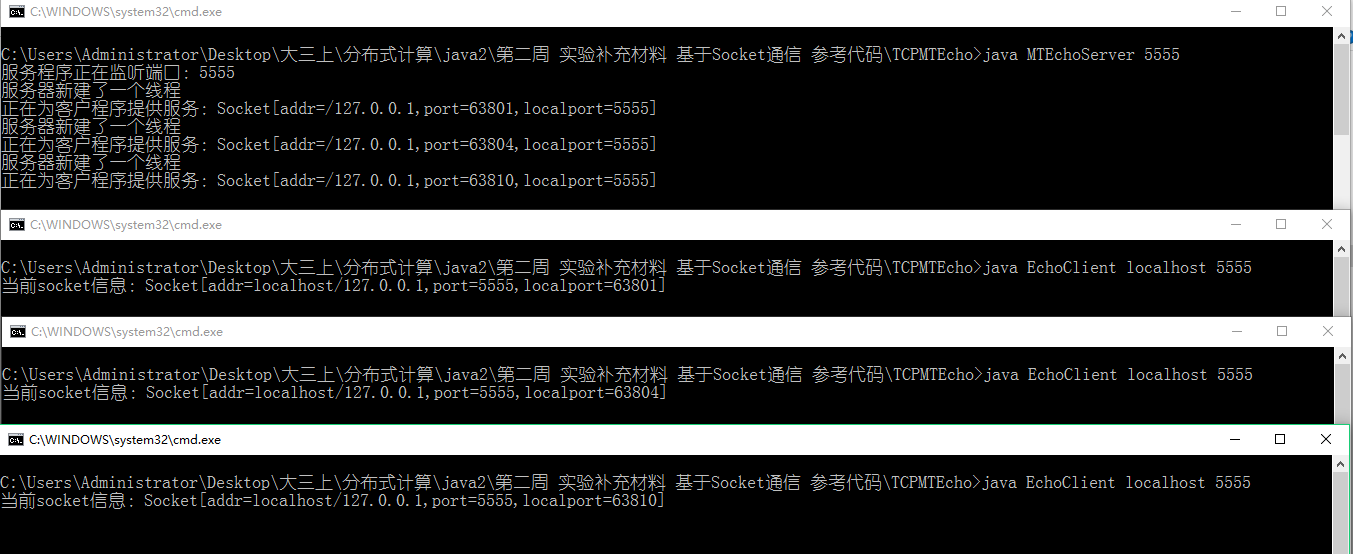
}

}

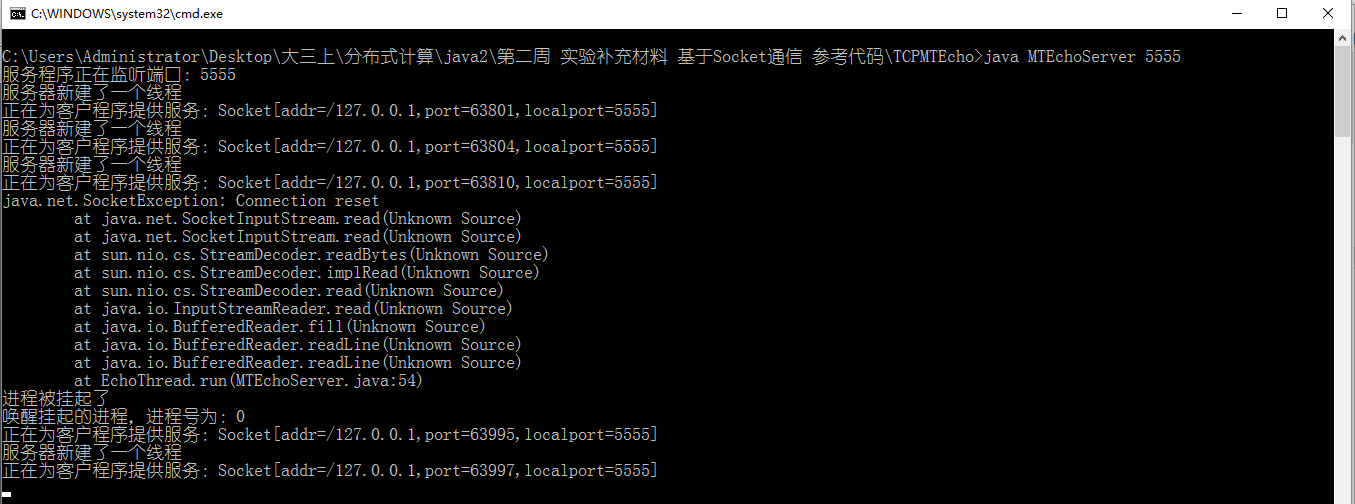
}

运行结果截图如下：

运行服务端，并运行三个客户端，



关闭其中一个客户端，再打开两个客户端，



服务器成功唤起之前挂起的进程。

8. RFC：Request For Comments，征求意见稿

IETF：Internet Engineering Task Force，互联网工程任务小组

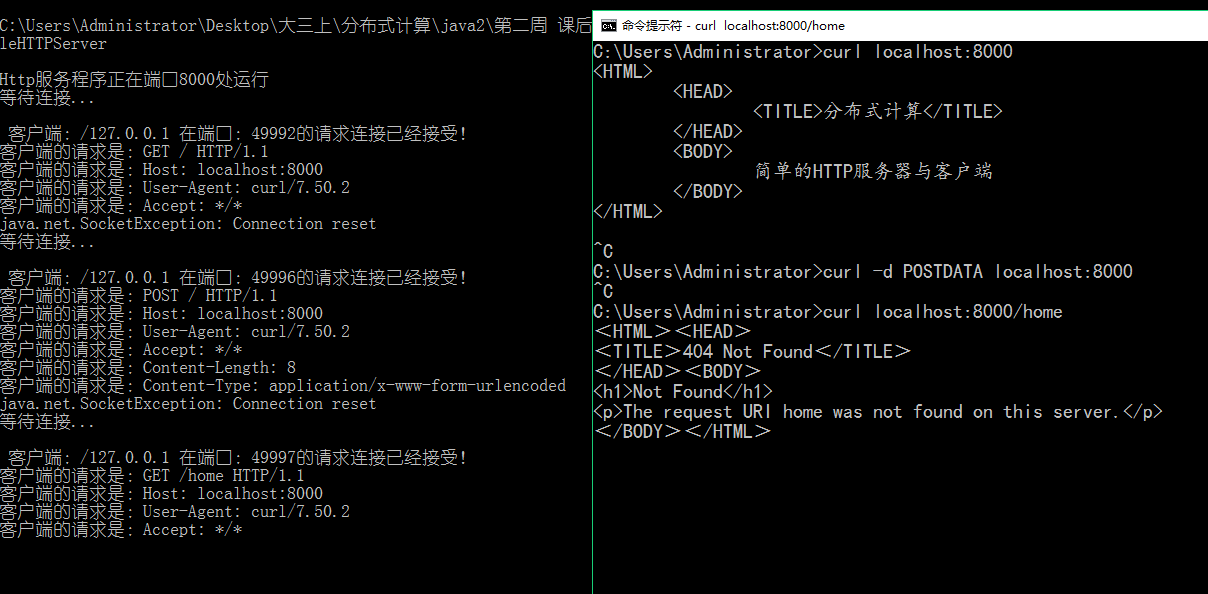
HTTP1.1：2616

<https://tools.ietf.org/html/rfc2616>

9. 图2-11描述了各个类之间互相调用其他类的实例的关系，使读者在阅读程序之前对程序的结构有一个整体的了解，便于理解各个类之间逻辑关系。

10. 客户端帮助类创建MyStreamSocket 类的实例对象mySocket，由实例对象的receiveMessage()方法接收服务端的响应消息。

11.



小结

1. 重温了网络七层协议，进一步理解了应用层和传输层的具体实现方式

2. 学习了Java中的Socket编程

3. 学习了如何编写多线程服务器

4. 学习了Java中线程的调度

5. 学习了在Windows下使用curl

6. 学习了客户端和服务器之间的多种交互方式