中山大學
SUN YAT-SEN UNIVERSITY

《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；

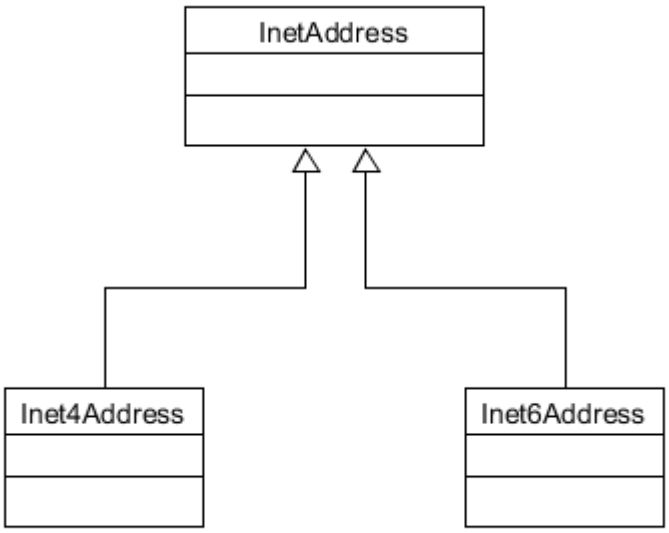
端口号用来指示传输的数据来自于远程主机或者属于本地的哪个应用程序。

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;
```

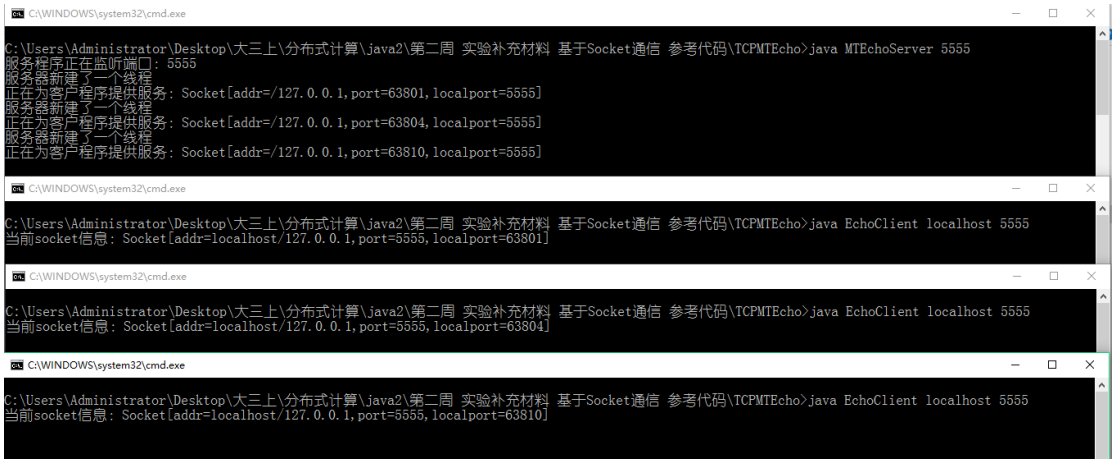
```
                        break;
                    }
                }
                if (need_create_new_thread) {
                    System.out.println("服务器新建了一个线程");
                    EchoThread thread =    new EchoThread(s);
                    thread.start();
                    threads.add(thread);
                }
            }
        }
}

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();
        }
```

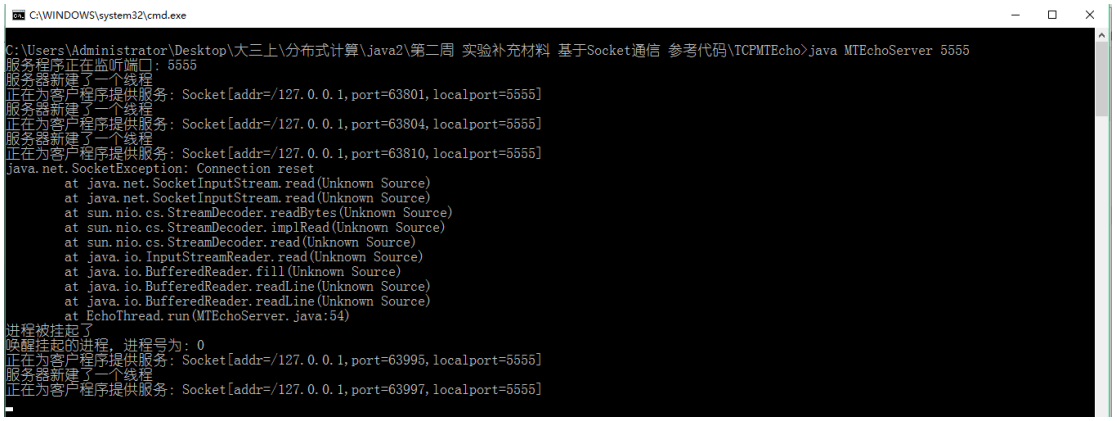
```
                finally {
                    System.out.println("进程被挂起了");
                    isSuspend = true;
                    suspend();
                }
            }
        }
}
```

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

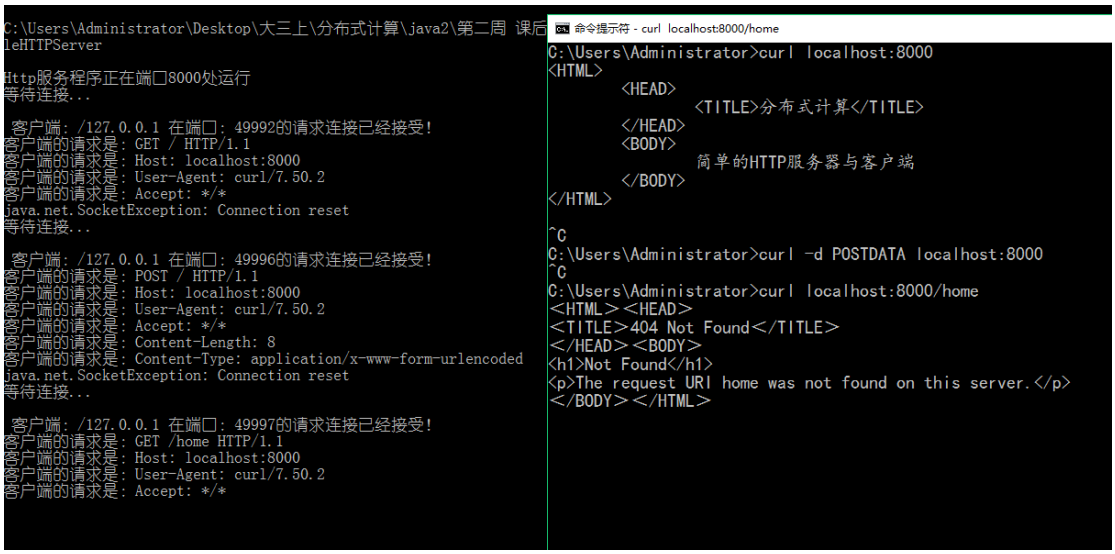
HTTP1.1：2616

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.


C:\Users\Administrator\Desktop\大三上\分布式计算\java2\第二周 课后
leHTTPServer
Http服务程序正在端口8000处运行
等待连接...
客户端: /127.0.0.1 在端口: 49992的请求连接已经接受!
客户端的请求是: GET / HTTP/1.1
客户端的请求是: Host: localhost:8000
客户端的请求是: User-Agent: curl/7.50.2
客户端的请求是: Accept: */*
java.net.SocketException: Connection reset
等待连接...
客户端: /127.0.0.1 在端口: 49996的请求连接已经接受!
客户端的请求是: POST / HTTP/1.1
客户端的请求是: Host: localhost:8000
客户端的请求是: User-Agent: curl/7.50.2
客户端的请求是: Accept: */*
客户端的请求是: Content-Length: 8
客户端的请求是: Content-Type: application/x-www-form-urlencoded
java.net.SocketException: Connection reset
等待连接...
客户端: /127.0.0.1 在端口: 49997的请求连接已经接受!
客户端的请求是: GET /home HTTP/1.1
客户端的请求是: Host: localhost:8000
客户端的请求是: User-Agent: curl/7.50.2
客户端的请求是: Accept: */*
命令提示符 - curl localhost:8000/home
C:\Users\Administrator>curl localhost:8000
<HTML>
<HEAD>
<TITLE>分布式计算</TITLE>
</HEAD>
<BODY>
简单的HTTP服务器与客户端
</BODY>
</HTML>
^C
C:\Users\Administrator>curl -d POSTDATA localhost:8000
^C
C:\Users\Administrator>curl localhost:8000/home
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<h1>Not Found</h1>
The request URI home was not found on this server.
</BODY></HTML>

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