实验 3 Java 网络编程

一、实验目的

1、了解网络协议的基本概念和区分 OSI 模型和 TCP/IP 模型的不同点。
2、熟悉输入输出流、以及常见的流处理工具。
3、掌握基于 TCP 编程和 UDP 编程的基本步骤和对相关核心类的了解。
4、掌握简单的应用服务器工作原理，为后面 Web 服务器开发打好基础。

二、实验内容

1、经典的“ECHO”程序讲解网络编程的通讯。

服务器程序：

```
public class EchoServer {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(9999);       // 设置服务监听端口
        System.out.println("等待客户端连接.............");   // 打印提示信息
        Socket client = server.accept();                    // 等待客户端连接
        // 首先需要先接收客户端发送来的信息，而后才可以将信息处理之后发送回客户端
        Scanner scan = new Scanner(client.getInputStream());    // 客户端输入流
        scan.useDelimiter("\n");                                // 设置分隔符
        PrintStream out = new PrintStream(client.getOutputStream()); // 客户端输出流
        boolean flag = true;                                        // 循环标记
        while (flag) {
            if (scan.hasNext()) {                           // 有数据接收
                String val = scan.next().trim();            // 接收数据内容
                if ("byebye".equalsIgnoreCase(val)) {       // 结束标记
                    out.println("ByeByeBye....");           // 回应信息
                    flag = false;                           // 结束循环
                } else {
                    out.println("【ECHO】" + val);          // Echo信息
                }
            }
        }
        scan.close();                                   // 关闭输入流
        out.close();                                    // 关闭输出流
        client.close();                                 // 关闭客户端
        server.close();                                 // 关闭服务端
    }
}
```

客户端程序：

```
public class EchoClient {
    private static final BufferedReader KEYBOARD_INPUT = new BufferedReader(
                new InputStreamReader(System.in));
    public static String getString(String prompt) throws Exception {    // 键盘信息输入
        System.out.print(prompt);
        String str = KEYBOARD_INPUT.readLine();
        return str;
    }
    public static void main(String[] args) throws Exception {
        Socket client = new Socket("localhost", 9999);      // 定义服务端的连接信息
        // 现在的客户端需要有输入与输出的操作支持，所以依然要准备出Scanner与PrintWriter
        Scanner scan = new Scanner(client.getInputStream());    // 接收服务端输入内容
        scan.useDelimiter("\n");
        PrintStream out = new PrintStream(client.getOutputStream()); // 向服务器端发送内容
        boolean flag = true;                                                    // 循环标记
        while (flag) {                                                          // 循环处理
            String input = getString("请输入要发送的内容：").trim();      // 获取键盘输入数据
            out.println(input);                                                 // 加换行
            if (scan.hasNext()) {                               // 服务器端有回应
                System.out.println(scan.next());                // 输出回应信息
            }
            if ("byebye".equalsIgnoreCase(input)) {             // 结束判断
                flag = false;                                   // 修改循环标记
            }
        }
        scan.close();                                       // 关闭输入流
        out.close();                                        // 关闭输 出流
        client.close();                                     // 关闭客户端
    }
}
```

2、UDP 服务器与客户端进行消息通信
客户端程序：

```
public class UDPClient {
    public static void main(String[] args) throws Exception {   // 接收数据信息
        DatagramSocket client = new DatagramSocket(9999);       // 9999端口监听
```

```
        byte data[] = new byte[1024];                                   // 保存接收数据
        DatagramPacket packet = new DatagramPacket(data, data.length);// 创建数据报
        System.out.println("客户端等待接收发送的消息.....");   // 提示信息
        client.receive(packet);                                         // 接收消息内容
        System.out.println("接收到的消息内容为：" + new String(data, 0,
packet.getLength()));
        client.close();                                         // 关闭连接
    }
}
```

服务器程序：

```
public class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket server = new DatagramSocket(9000);       // 9000端口监听
        String str = "www.mldn.cn";                             // 发送消息
        DatagramPacket packet = new DatagramPacket(str.getBytes(), 0, str.length(),
                InetAddress.getByName("localhost"),9999);       // 发送数据
        server.send(packet);                                    // 发送消息
        System.out.println("消息发送完毕.....");
        server.close();                                         // 关闭服务端
    }
}
```

三、根据下面要求编写程序（二选一）

1、实现利用 BIO 模型（传统阻塞 IO 模型）实现多用户访问,要求实现服务器与客户端代码。提示服务器端需要多线程响应客户端请求。

2、实现简单 WebServer，要求服务器可以处理返回给客户单一段字符串，该字符串信息是读取服务器文件系统的文件，文件格式可以是 XML 文件或者 txt 文件。具体参考实现课本 5.4.5。

四、实验结果

写实验报告。内容包括：

1. 习题的运行结果，源程序。
2. 程序调试中出现的错误提示。（英文、中文对照）

若有没通过的程序，分析原因。

附：windows 操作系统下 kill 进程的方法：

windows中，端口查看&关闭进程及Kill使用

测试过程中遇到的问题，杂记一：

1、netstat -ano | findstr "8001"　查看端口8001被哪个进程占用；由下图可以看出，被进程为3736的占用

```
C:\Users\DELL>netstat -ano | findstr "8001"
  TCP    0.0.0.0:8001           0.0.0.0:0              LISTENING       3736
  TCP    [::]:8001              [::]:0                 LISTENING       3736
```

2、查看进程号为3736对应的进程；由下图可以看出，是被java.exe占用了

命令：tasklist | findstr "3736"

```
C:\Users\DELL>tasklist | findstr "3736"
java.exe                     3736 Console                    1        976 K
```

3、结束该进程

命令：taskkill /f /t /im java.exe

```
C:\Users\DELL>taskkill /f /t /im java.exe
成功: 已终止 PID 3736 (属于 PID 3704 子进程)的进程。
```

4、查看所有的端口占用情况

命令：netstat -ano

```
C:\Users\DELL>netstat -ano

活动连接

  协议  本地地址          外部地址        状态           PID
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       1004
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:3306           0.0.0.0:0              LISTENING       1688
  TCP    0.0.0.0:49152          0.0.0.0:0              LISTENING       696
  TCP    0.0.0.0:49153          0.0.0.0:0              LISTENING       528
  TCP    0.0.0.0:49154          0.0.0.0:0              LISTENING       612
  TCP    0.0.0.0:49155          0.0.0.0:0              LISTENING       752
  TCP    0.0.0.0:49158          0.0.0.0:0              LISTENING       808
  TCP    127.0.0.1:4300         0.0.0.0:0              LISTENING       9256
  TCP    127.0.0.1:4301         0.0.0.0:0              LISTENING       9256
  TCP    127.0.0.1:32767        0.0.0.0:0              LISTENING       6984
  TCP    127.0.0.1:32768        0.0.0.0:0              LISTENING       9788
  TCP    127.0.0.1:49573        127.0.0.1:49574        ESTABLISHED     2724
  TCP    127.0.0.1:49574        127.0.0.1:49573        ESTABLISHED     2724
  TCP    127.0.0.1:52046        0.0.0.0:0              LISTENING       2256
  TCP    172.18.84.39:139       0.0.0.0:0              LISTENING       4
```