华东师范大学数据科学与工程学院实验报告

课程名称： 计算机网络与编程	**年级：** 21 级	**上机实践成绩：**
指导教师： 张召	**姓名：** 杨茜雅	**学号：** 10215501435
上机实践名称： Lab07		**上机实践日期：** 2023.4.14
上机实践编号： 7	**组号：**	**上机实践时间：** 4.14-4.21

一、实验目的

使用ServerSocket和Socket实现TCP通信

了解粘包概念并尝试解决

二、实验任务

使用ServerSocket和Socket编写代码

解决粘包问题

三、使用环境

IntelliJ IDEA

JDK 版本：Java 19

四、实验过程

Task 1：使用Scanner 修改 TCPClient 类，达成如下效果，请将实现代码段及运行结果附在实验报告中。

客户端不断读取用户控制台输入的一行英文字母串并将数据发送给服务器

服务器将收到的字符全部转换为大写字母，服务器将修改后的数据发送给客户端

客户端收到修改后的数据，并在其屏幕上显示

TCPServer：

TCPServer.java TCPClient.java

```
package Task1;
import java.io.*;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;

public class TCPServer {
private static final int PORT = 9091;
public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket();
        serverSocket.setReuseAddress(true);
        serverSocket.setReceiveBufferSize(64 * 1024 * 1024);
        serverSocket.bind(new InetSocketAddress( hostname: "127.0.0.1", PORT),  backlog: 50);

        System.out.println("阻塞等待客户端连接中...");
        Socket socket = serverSocket.accept();

        InputStream is = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
```

```
        String info;
        ArrayList list = new ArrayList();
        while(true){
            info = br.readLine();
            if(info != null){
                list.add(info.toUpperCase());
                System.out.println("我是服务器，收到客户端字符：  " + info);
                }else {
                break;
                }
             }
        list.add(null);
        socket.shutdownInput();
    OutputStream os = socket.getOutputStream();
    int i=0;
    while(list.get(i)!=null){
        String message = list.get(i).toString();
        i++;
        os.write(message.getBytes(StandardCharsets.UTF_8));
        }

    System.out.println("我是服务器，已经转换为大写，并回复客户端");
    socket.shutdownOutput();

    os.close();
    br.close();
    isr.close();
    is.close();
    socket.close();
    serverSocket.close();
     }
}
```

TCPClient:

TCPServer.java | TCPClient.java

```
package Task1;

import java.io.*;
import java.util.*;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
public class TCPClient {
private static final int PORT = 9091;
public static void main(String[] args) throws IOException {

        Socket socket = new Socket();
        socket.connect(new InetSocketAddress( hostname: "127.0.0.1", PORT), timeout: 3000);

        OutputStream os = socket.getOutputStream();
        System.out.println("请输入一行字符：");

        Scanner in = new Scanner(System.in);
        String message= in.nextLine();

    os.write(message.getBytes(StandardCharsets.UTF_8));
    socket.shutdownOutput();
```

```
30              System.out.println("我是客户端，服务器将字符转换为大写：  " + info);
31              }else {
32              break;
33              }
34          }
35      socket.shutdownInput();
36
37      br.close();
38      isr.close();
39      is.close();
40      os.close();
41      socket.close();
42      }}
```

运行结果：

TCPServer

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.
阻塞等待客户端连接中...
我是服务器，收到客户端字符：  yangxiya
我是服务器，已经转换为大写，并回复客户端

Process finished with exit code 0
```

TCPClient

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.ex
请输入一行字符：
yangxiya
我是客户端，服务器将字符转换为大写：  YANGXIYA

Process finished with exit code 0
```

Task 2:

修改代码使得每一次accept()的Socket都被一个线程接管，同时接管的逻辑保留Task1的功能，开启一个服务端和三个客户端进行测试，请将实现代码段及运行结果附在实验报告中。

TCPServer:

```
package Task2;

import java.io.*;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

public class TCPServer {
    private static final int PORT = 9091;
    public static void main(String[] args) throws IOException {

        ServerSocket serverSocket = new ServerSocket();
        serverSocket.setReuseAddress(true);
        serverSocket.setReceiveBufferSize(64 * 1024 * 1024);
        serverSocket.bind(new InetSocketAddress( hostname: "127.0.0.1", PORT),  backlog: 50);
        while(true) {
            System.out.println("阻塞等待客户端连接中...");
            Socket socket = serverSocket.accept();
            InputStream is = socket.getInputStream();
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            OutputStream os = socket.getOutputStream();
            PrintWriter printWriter = new PrintWriter(os);
            String info;
            ArrayList list = new ArrayList();
              while (true) {
                  info = br.readLine();
                  if (info != null) {
                      list.add(info.toUpperCase());
                      System.out.println("我是服务器，收到客户端字符：  " + info);
                  } else {
                      break;
                  }
              }
              list.add(null);
              socket.shutdownInput();
              int i = 0;
              while (list.get(i) != null) {
                  String message = list.get(i).toString();
                  i++;
                  printWriter.println(message);
                  printWriter.flush();
              }
              System.out.println("我是服务器，已经转换为大写，并回复客户端");
              socket.shutdownOutput();
        }
    }
}
```

```
class ClientHandler extends Thread {
    private Socket socket;
    ClientHandler(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        super.run();
        System.out.println("新客户端连接：" + socket.getInetAddress() +
                ": " + socket.getPort());
        try {
            InputStream inputStream = socket.getInputStream();
            OutputStream outputStream = socket.getOutputStream();
            byte[] bytes = new byte[1024];
            while(true){
                int len = inputStream.read(bytes);
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.append(new String(bytes, offset: 0, len));
                System.out.println("收到客户端消息：" + stringBuilder);
                outputStream.write(stringBuilder.toString().getBytes());
            }
        } catch (IOException e) {
            System.out.println("连接异常断开");
        }
    }
}
```

TCPClient:

```
package Task2;
import java.io.*;
import java.util.*;
import java.net.InetSocketAddress;
import java.net.Socket;
public class TCPClient {
    private static final int PORT = 9091;
    public static void main(String[] args) throws IOException {
        while(true) {
            Socket socket = new Socket();
            socket.connect(new InetSocketAddress( hostname: "127.0.0.1", PORT), timeout: 3000);
            OutputStream os = socket.getOutputStream();
            PrintWriter printWriter = new PrintWriter(os);
            System.out.println("请输入一行字符：");
            Scanner in = new Scanner(System.in);
            String message = in.nextLine();
            printWriter.println(message);
            printWriter.flush();
            socket.shutdownOutput();
            InputStream is = socket.getInputStream();
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            while (true) {
                String info = br.readLine();
                if (info != null) {
                    System.out.println("我是客户端，服务器将字符转换为大写：  " + info);
                } else {
                    break;}}
            socket.shutdownInput();}
    } }
```

运行结果（三个 client）：

```
请输入一行字符：
yangxiya
我是客户端，服务器将字符转换为大写： YANGXIYA
请输入一行字符：
dase
我是客户端，服务器将字符转换为大写： DASE
请输入一行字符：
today
我是客户端，服务器将字符转换为大写： TODAY
```

Task 3:

查阅资料，总结半包粘包产生的原因以及相关解决方案，尝试解决以上代码产生的半包粘包问题，将修改代码和解决思路附在实验报告中。

运行所给代码：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program F
阻塞等待客户端连接中...
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLENETWORK PRINCIPLENETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
```

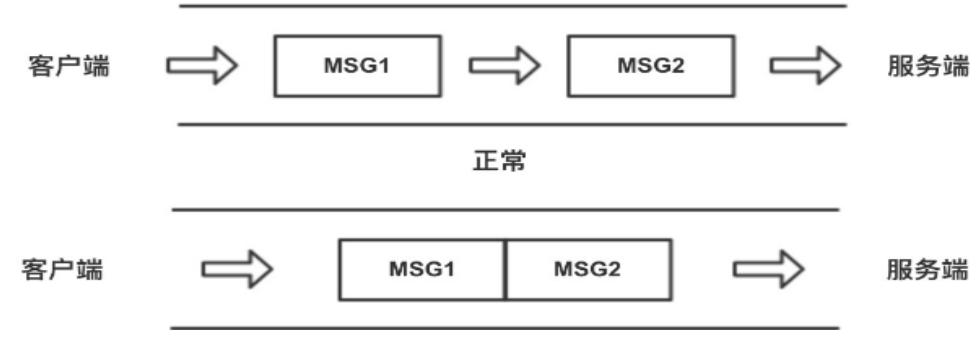
观察结果可以发现这里发生了半包和粘包问题：

```
服务端已收到消息: NETWORK PRINCIPLENETWORK PRINCIPLENETWORK PRINCIPLE
```

原因解释：

1、粘包现象

TCP 传输中，客户端发送数据，实际是把数据写入到了TCP 的缓存中，粘包和半包也就会在此时产生。如果客户端发送的包的大小比TCP 的缓存容量小，并且TCP 缓存可以存放多个包，那么客户端和服务端的一次通信就可能传递了多个包，这时候服务端从TCP 缓存就可能一下读取了多个包，这种现象就叫粘包。



粘包的主要原因：

1、　发送端需要等缓冲区满才发送出去，造成粘包。

发送方引起的粘包是由 TCP 协议本身造成的，TCP 为提高传输效率，发送方往往要收集到足够多的数据后才发送一包数据。若连续几次发送的数据都很少，通常 TCP 会根据优化算法把这些数据合成一包后一次发送出去，这样接收方就收到了粘包数据。

2、接收方不及时接收缓冲区的包，造成多个包接收

接收方引起的粘包是由于接收方用户进程不及时接收数据，从而导致粘包现象。这是因为接收方先把收到的数据放在系统接收缓冲区，用户进程从该缓冲区取数据，若下一包数据到达时前一包数据尚未被用户进程取走，则下一包数据放到系统接收缓冲区时就接到前一包数据之后，而用户进程根据预先设定的缓冲区大小从系统接收缓冲区取数据，这样就一次取到了多包数据。

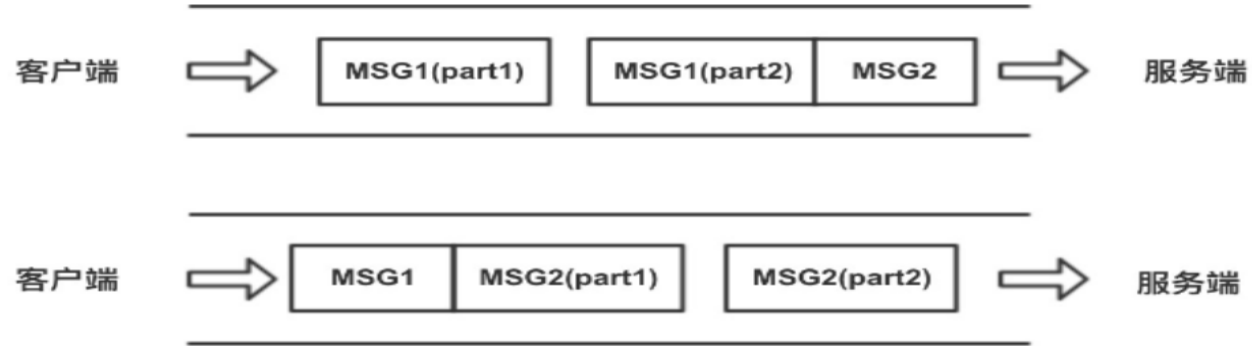
解决方法：

（1）对于发送方引起的粘包现象，用户可通过**编程设置**来避免，TCP 提供了强制数据立即传送的操作指令 push，TCP 软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；**但是**但它关闭了优化算法，降低了网络发送效率，影响应用程序的性能，一般不建议使用。

（2）对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；**但是**第二种方法只能减少出现粘包的可能性，但并不能完全避免粘包，当发送频率较高时，或由于网络突发可能使某个时间段数据包到达接收方较快，接收方还是有可能来不及接收，从而导致粘包

（3）由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。**但是**第三种方法虽然避免了粘包，但应用程序的效率较低，对实时应用的场合不适合。比如说认为确定一个消息边界。

2、半包现象
如果客户端发送的包的大小比TCP 的缓存容量大，那么这个数据包就会被分成多个包，通过 Socket 多次发送到服务端，服务端第一次从接受缓存里面获取的数据，实际是整个包的一部分，这时候就产生了半包。
半包现象：



半包的主要原因：
1、 发送方每次写入数据＞套接字(Socket)缓冲区大小

2、 发送的数据大于协议的最大传输单元，因此必须拆包

解决方法：
（1）将TCP 连接改成短连接，一个请求一个短连接。使得建立连接到释放连接之间的消息即为传输的信息，消息也就产生了边界。

（2）封装成帧，也就是原本发送消息的单位是缓冲大小，现在换成了帧，可以自定义边界了。

（3）固定长度，消息边界也就是固定长度。

（4）分隔符号，消息边界也就是分隔符本身。

（5）一个专门的字段存储消息的长度。作为服务端，接受消息时，先解析固定长度的字段获取消息总长度，然后读取后续内容。

修改代码：
1、 尝试固定长度，由于发送的是“NETWORK PRINCIPLE”，长度设置为 17

```
private static int BYTE_LENGTH = 17;
public void start(int port) throws IOException {
    serverSocket = new ServerSocket(port);
    System.out.println("阻塞等待客户端连接中...");
    clientSocket = serverSocket.accept();
    InputStream is = clientSocket.getInputStream();
```

```
C:\Users\86138\.jdks\openjdk-19.0
阻塞等待客户端连接中...
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
服务端已收到消息: NETWORK PRINCIPLE
```

2、 固定缓冲区大小

需要控制服务器端和客户端发送和接受字节的（数组）长度相同。

虽然这种方式可以解决粘包和半包的问题，但这种固定缓冲区大小的方式**增加了不必要的数据传输**，因为这种方式当发送的数据比较小时会使用**空字符来弥补**，所以这种方式就大大的增加了网络传输的负担，所以它也不是最佳的解决方案。

Server 端：

```
package Task3;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
/**
 * 服务器端，改进版本一（只负责接收消息）
 */
class ServSocketV1 {
    private static final int BYTE_LENGTH = 1024;  // 字节数组长度（收消息用）
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket( port: 9091);
        // 获取到连接
        Socket clientSocket = serverSocket.accept();
        try (InputStream inputStream = clientSocket.getInputStream()) {
            while (true) {
                byte[] bytes = new byte[BYTE_LENGTH];
                // 读取客户端发送的信息
                int count = inputStream.read(bytes,  off: 0, BYTE_LENGTH);
                if (count > 0) {
                    // 接收到消息打印
                    System.out.println("接收到客户端的信息是:" + new String(bytes).trim());
                }
                count = 0;
            }
        }
    }
}
```

Client 端：

```java
package Task3;

import java.io.*;
import java.net.Socket;
/**
 * 客户端，改进版一（只负责接收消息）
 */
class ClientSocketV1 {
    private static final int BYTE_LENGTH = 1024;  // 字节长度
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket( host: "127.0.0.1",  port: 9091);
        final String message = "NETWORK PRINCIPLE"; // 发送消息
        try (OutputStream outputStream = socket.getOutputStream()) {
            // 将数据组装成定长字节数组
            byte[] bytes = new byte[BYTE_LENGTH];
            int idx = 0;
            // 将 message 里面的内容全部移动到 bytes 里面
            for (byte b : message.getBytes()) {
                bytes[idx] = b;
                idx++;
            }
            // 给服务器端发送 10 次消息
            for (int i = 0; i < 10; i++) {
                outputStream.write(bytes,  off: 0, BYTE_LENGTH);
            }
        }
    }
}
```

3、 特殊字符结尾，按行读取

这种解决方案的核心是，使用 Java 中自带的 BufferedReader 和 BufferedWriter，也就是带缓冲区的输入字符流和输出字符流，通过写入的时候加上 \n 来结尾，读取的时候使用 readLine 按行来读取数据，这样就知道流的边界了，从而解决了粘包和半包的问题。

Server 端：

```
package Task3;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
public class TCPSever {
    //端口号
    private static final int port = 9005;
    //数据传输的最大值
    private static final int leng = 1024;
    public static void main(String[] args) throws IOException {
        //1.创建TCP服务器端
        ServerSocket serverSocket = new ServerSocket(port);
        System.out.println("服务器启动成功！");
        //2.等待客户端的连接，连接起来就可以进行交互
        Socket clientSocket =  serverSocket.accept();//会返回socket对象
        //连接起来就可以进行交互
        System.out.println(String.format("有客户端连接了，客户端IP:%s  端口：%d",
                clientSocket.getInetAddress().getHostAddress(),clientSocket.getPort()));
        try(BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))){
            while (true){
                //按行定义边界
                String msg = reader.readLine();
                if(msg!=null && msg!=null){
                    System.out.println("读取到客户端消息："+msg);
                }
            }
        }
    }
}
```

Client 端：

```java
package Task3;

import java.io.*;
import java.net.Socket;
public class TCPClientError {
    private static final String ip="127.0.0.1";
    private static final int port = 9005;
    public static void main(String[] args) throws IOException {
        //创建客户端并连接服务器
        Socket socket = new Socket(ip,port);

        //发送的消息
        String msg = "NETWORK PRINCIPLE\n";
        //构建发送对象
        try(OutputStream outputStream = socket.getOutputStream()){

            for (int i = 0;i<10;i++){
                //以字节方式传输
                outputStream.write(msg.getBytes(), off: 0,msg.getBytes().length);
            }

        }
    }
}
```

五、总结

通过本次实验，我尝试使用ServerSocket和Socket编写代码实现TCP通信，不断修改TCPServer和TCPClient类实现不同功能：如单服务端单客户端、单服务端多客户端、服务器将从客服端收到的字符全部转换为大写字母再发送给客户端等功能。了解了粘包和半包的概念、形成原因和解决方法、并实践修改代码。