

华东师范大学数据科学与工程学院上机实践报告

课程名称：计算机网络原理与编程

年级：2018

上机实践成绩：

指导教师：张召

姓名：孙秋实

学号：10185501402

上机实践名称：JavaSocket++

上机实践日期：2020/4/27

上机实践编号：Exp5

组号：

上机实践时间：

Part 1

实验目的

- 对数据发送和接收进行优化
- 了解粘包概念

Part 2

实验任务

- 将数据发送与接收并行
- 解决粘包问题

Part 3

使用环境

- IntelliJ IDEA Version 2019.3.4
- JDK 版本 11.0.6

Part 4

实验过程

Task 1

修改 TCPClient 类使其发送和接收并行，即当一个服务端和多个客户端启动时，无论是服务端还是客户端均可随时发送消息并接收消息，将修改后的 Client 端所有代码附在实验报告中

在开始实验前，我先对 TestServer 进行了一些微调，因为模板代码的键盘输入依赖 hasNext 来判断导致 Server 端回复句子的话会被截断为一个个单词，所以调整为了 hasNextLine，这样的话服务端返回的句子消息才能正常的显示。

```

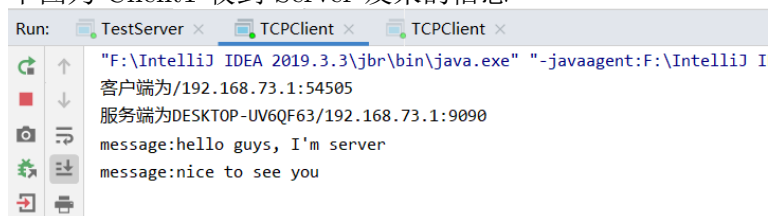
1 public class TestServer {
2     private static final int PORT = 9090;
3     public static void main(String[] args) throws IOException {
4         Server server = new Server(PORT);
5         server.start();
6         Scanner sc = new Scanner(System.in);
7         while (sc.hasNextLine()) { //MODIFIED: HASNEXT->HASNEXTLINE
8             String s = sc.nextLine(); //MODIFIED: .NEXT->.NEXTLINE
9             server.broadcast(s);
10        }
11    }
12 }

```

接下来是修改 TCPClient 的代码，实现多个 Client 和 Server 之间通信

首先是服务端先发出消息，需要在客户端里创建一个新的线程来监听，否则服务端只有在收到客户端消息后才能回信

下图为 Client1 收到 Server 发来的信息

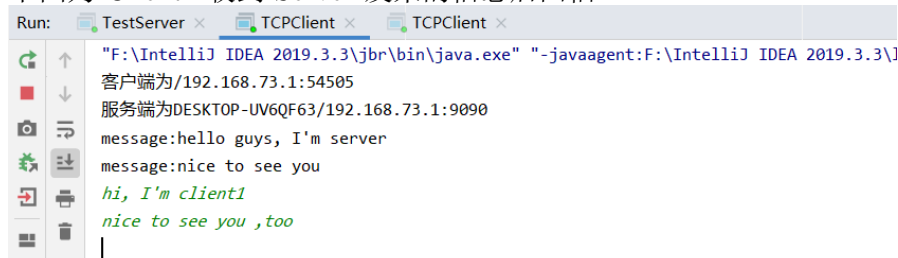


```

Run: TestServer x TCPClient x TCPClient x
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\IntelliJ I
客户端为/192.168.73.1:54505
服务端为DESKTOP-UV6QF63/192.168.73.1:9090
message:hello guys, I'm server
message:nice to see you

```

下图为 Client1 收到 Server 发来的信息后回信

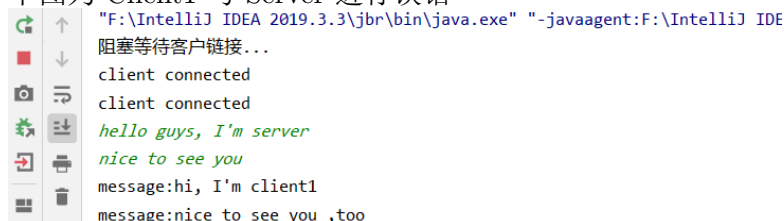


```

Run: TestServer x TCPClient x TCPClient x
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\IntelliJ IDEA 2019.3.3\j
客户端为/192.168.73.1:54505
服务端为DESKTOP-UV6QF63/192.168.73.1:9090
message:hello guys, I'm server
message:nice to see you
hi, I'm client1
nice to see you ,too
|

```

下图为 Client1 与 Server 进行谈话

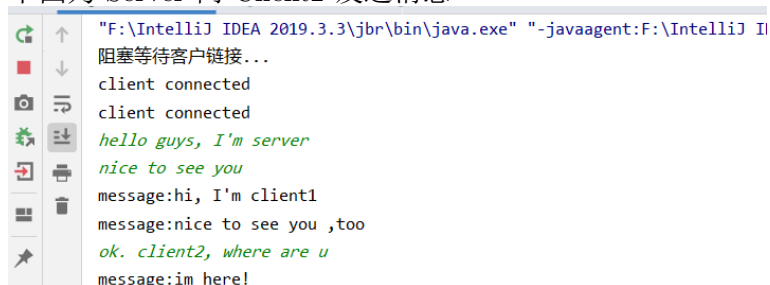


```

Run: TestServer x TCPClient x TCPClient x
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\IntelliJ IDE
阻塞等待客户链接...
client connected
client connected
hello guys, I'm server
nice to see you
message:hi, I'm client1
message:nice to see you ,too

```

下图为 Server 向 Client2 发送消息



```

Run: TestServer x TCPClient x TCPClient x
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\IntelliJ I
阻塞等待客户链接...
client connected
client connected
hello guys, I'm server
nice to see you
message:hi, I'm client1
message:nice to see you ,too
ok. client2, where are u
message:im here!

```

下图为 Client2 与 Server 进行谈话

```
Run: TestServer x TCPClient x TCPClient x
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\
客户端为/192.168.73.1:54512
服务端为DESKTOP-UV6QF63/192.168.73.1:9090
message:hello guys, I'm server
message:nice to see you
message:ok. client2, where are u
im here!
|
```

其次是 Client 端先发送消息，原理基本一致:

```
Run: TestServer x TCPClient x TCPClient x
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\IntelliJ IDEA 2019.3.3
阻塞等待客户链接...
client connected
client connected
message:hello! I'm client2!~
message:hello! I'm client1!~
hello! I'm server!~
```

以下是实现代码

```
1 package exp5;
2 import java.io.*;
3 import java.net.Inet4Address;
4 import java.net.InetSocketAddress;
5 import java.net.Socket;
6 import java.util.Scanner;
7 public class TCPClient {
8     private static final int PORT = 9090;
9     public static void main(String[] args) throws IOException {
10         Socket socket = new Socket();
11         socket.connect(new InetSocketAddress(Inet4Address.getLocalHost(), PORT), 3000);
12         System.out.println("客户端为" + socket.getLocalAddress() + ":" + socket.getLocalPort());
13         System.out.println("服务端为" + socket.getInetAddress() + ":" + socket.getPort());
14         InputStream inputStream = socket.getInputStream();
15         OutputStream outputStream = socket.getOutputStream();
16         // PRINTSTREAM SOCKETPRINTSTREAM = NEW PRINTSTREAM(OUTPUTSTREAM);
17         // BUFFEREDREADER SOCKETINPUT = NEW BUFFEREDREADER(NEW INPUTSTREAMREADER(INPUTSTREAM));
18         Scanner sc = new Scanner(System.in);
19
20         ClientReadHandler ReadMessage2Client = new ClientReadHandler(inputStream); //MODIFIED
21         ReadMessage2Client.start();
22
23         while (sc.hasNext()) { //NEXTLINE?
24             String input = sc.nextLine();
25             ClientWriteHandler Clie2Server = new ClientWriteHandler(outputStream); //MODIFIED
26             Clie2Server.send(input); //MODIFIED
27             ClientReadHandler Server2Client = new ClientReadHandler(inputStream); //MODIFIED
28             Server2Client.start();
29         }
30
31         outputStream.close();
```

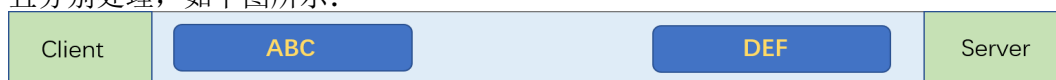
```
32     inputStream.close();
33     socket.close();
34     System.out.println("Client exit!");
35 }
36 }
```

Task 2

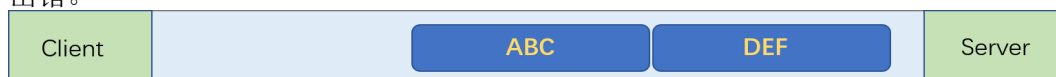
查阅资料，分析粘包可能产生的原因并搜索若干种解决办法

先举例分析一下发生粘包现象可能的原因（为了更直观地理解，我做了几张小插图来演示）：

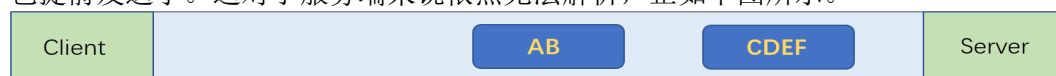
首先，如果我试图从客户端向服务端发送两个数据包，理想的状况下服务端依次接收到这两个包，并且分别处理，如下图所示：



但 TCP 是一个面向字节流的协议，它的性质是流式的，所以它并不会对我发出的两个数据包进行硬性的分段，如下图所示，两个数据包被合并在一起送到了服务端，服务端若按第一个包的逻辑来处理则会出错。



即使他们没有合并成一个包中，也有可能出现两个包的内容互相包含，或是一个包的信息被另一个包提前发送了。这对于服务端来说依然无法解析，正如下图所示。



以下列举一些可以解决该问题的方法

- (1) **固定消息长度**: 把消息设置为固定长度即可，如果数据不够，空位补空格。优点就是实现很简单方便操作，缺点是在连续发送短信息时空间会产生大量浪费。
- (2) **使用分隔符**: 用一个分隔符来确定消息边界。优点是空间不浪费，实现也相对简单。缺点是当内容本身出现分隔符时需要转义处理，所以无论是 Client 端发送时还是 Server 端接收时，都需要对所有数据进行扫描，降低了传输效率。
- (3) **把 TCP 连接形式改成短连接**: 每次进行传输就启动一个短连接。这样的话按逻辑传递的信息就有了边界。但缺点也就很明显了，每次传递消息都要进行三次握手过程重新建立连接，会浪费大量时间用在建立连接上，降低了传输效率。
- (4) **采用变长协议**: 将消息区分为消息头和消息体，在消息头中，使用一个整形数字，如一个 int，来表示消息体的长度，执行该协议时发送方在发送数据之前，需要先获取需要发送内容的二进制字节大小，然后在需要发送的内容前面添加一个整数，表示消息体二进制字节的长度。接收方在解析时，先读取内容长度，其值为实际消息体内容占用的字节数，之后必须读取到这么多字节的内容，才认为是一个完整的数据报文。