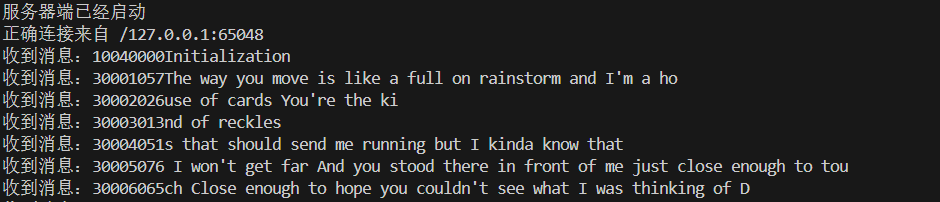
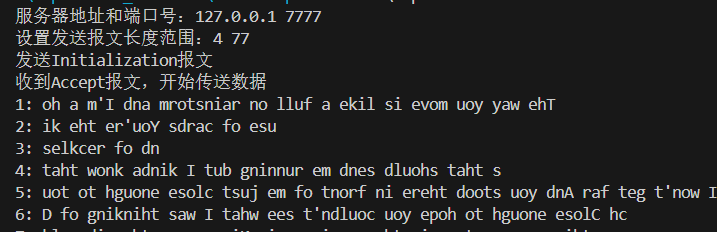
一、**运行结果与wireshark包捕获的截图：**

1.使用VSCode运行Java项目，分为工程tcpServer和tcpClient，**最终目的是实现报文内容的翻转**，运行结果如图一图二所示。

图一、服务器端显示收到消息的内容

图二、客户端显示发送消息的翻转

**2.报文格式**：消息类型（1位）+分割的报文段（4位）+消息编号（3位）+发送的数据

**3.不定长**：这个功能由随机函数实现，具体代码片段如下所示

1. **if** (srcText.length() <= Lmax) {
2. block\_msg.add(srcText);
3. block\_len.add(srcText.length());
4. **break**;
5. }
7. Random random = **new** Random();
8. **int** length = random.nextInt(Lmax - Lmin + 1) + Lmin;
9. String msg = srcText.substring(0, length);
10. srcText = srcText.substring(length);
11. block\_msg.add(msg);
12. block\_len.add(length);

**4.交互过程：**交互过程可以在图一图二中体现。客户端在发送intialization报文后，收到了Accept报文。而Server端最开始收到了消息类型为1的初始报文。报文内容为Intialization。后续接收到了消息类型为3的报文，并返回翻转报文。

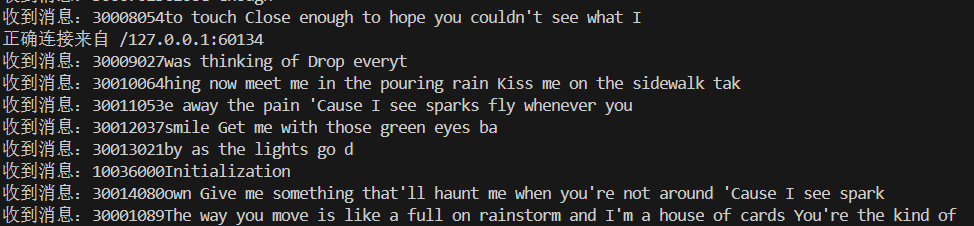
**5.指定serverIP、serverPort**

本功能已实现，如图二所示。

**6.限定长度与计算块数**

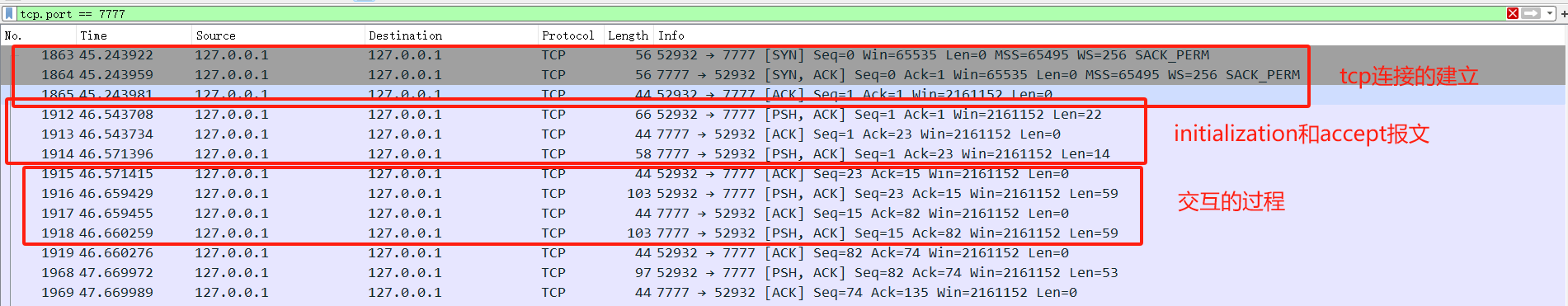
本功能已实现，如图二所示。计算块数代码在3不定长中展示，使用block\_msg的数量来实现。

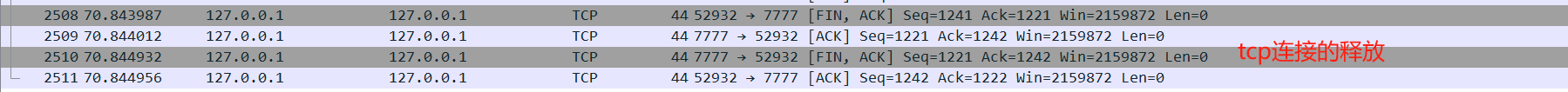
**7.非阻塞**

我利用了**Selector、ServerSocketChannel、SelectionKey组件实现了非阻塞的功能。如图三所示，我们可以发现在接收消息时，客户端还处理了第二个连接，在最下面可以发现收到了第二个连接传来的第一个包（可以从第四、五位的数字看出）。**

**图三、非阻塞的证明**

**8.wireshark截图**

**这部分主要是看报文的交互，如图四、五所示。**

图四、wireshark的截图1

图五、wireshark的截图2

二、**实现的关键点和对应的解决方案**

**1.服务器的非阻塞并发**

在udp的实验中我犯了相关的错误，在tcp的实验中总结了相关的经验教训并查阅了相关select的资料，成功的实现了非阻塞并发。要实现非阻塞并发关键在以下三个部分：

**1.1统一的客户端的socketChannel：**

这里要着重解释为什么服务器的非阻塞并发要写客户端的socketChannel，这是由于我在udp实验中采用了建立连接，聊天，释放连接三个不同的socket，导致服务器没有办法按照发送来时的客户端的端口返回相关的报文，必须每一次都先接收客户端的报文，解析他的端口才能返回。有了上面的经验教训，我在tcp中得以改进。代码如下。

1. **public** **class** tcpclient {
2. **private** **static** SocketChannel socketChannel;

**1.2服务器端发送、接收和处理模块要分开设计**

在过早的完成了udp试验后，发现并发这一功能无法实现。在经过两天半的调试bug后，发现由于最开始设计的模块方向错误，server端和client端的发送和接收逻辑咬死，代码中多次重复出现接收和发送，导致无法修改。有了上面的经验教训，我在tcp中得以改进。代码如下。

1. SocketChannel socketChannel = (SocketChannel) key.channel();
2. ByteBuffer buffer = ByteBuffer.allocate(BUFSIZE);
3. **int** bytesRead = socketChannel.read(buffer);
5. **if** (bytesRead == -1) {
6. System.out.println("断开连接来自 " + socketChannel.getRemoteAddress());
7. socketChannel.close();
8. **return**;
9. }
11. buffer.flip();
12. **byte**[] data = **new** **byte**[buffer.remaining()];
13. buffer.get(data);
14. buffer.clear();
16. String message = **new** String(data, StandardCharsets.UTF\_8).trim();
17. System.out.println("收到消息：" + message);
19. // 模拟处理数据并发送处理后的数据
20. String response = processMessage(message);
21. socketChannel.write(ByteBuffer.wrap(response.getBytes(StandardCharsets.UTF\_8)));

**1.3组件的合理使用**

Selector是Java NIO中的多路复用器，它用于监控多个通道（Channel）的I/O事件（如读、写、连接等），可以使单个线程同时管理多个通道，从而实现非阻塞I/O。Selector允许一个线程管理多个通道，在没有I/O事件发生时，线程可以执行其他任务或等待，提高了程序的效率和响应性。

ServerSocketChannel是一个可以监听新进来的TCP连接的通道（Channel），它是非阻塞的，并且可以配置为非阻塞模式。ServerSocketChannel可以在非阻塞模式下配置，使得accept方法不会阻塞，并且允许程序在没有连接请求时继续执行其他任务。

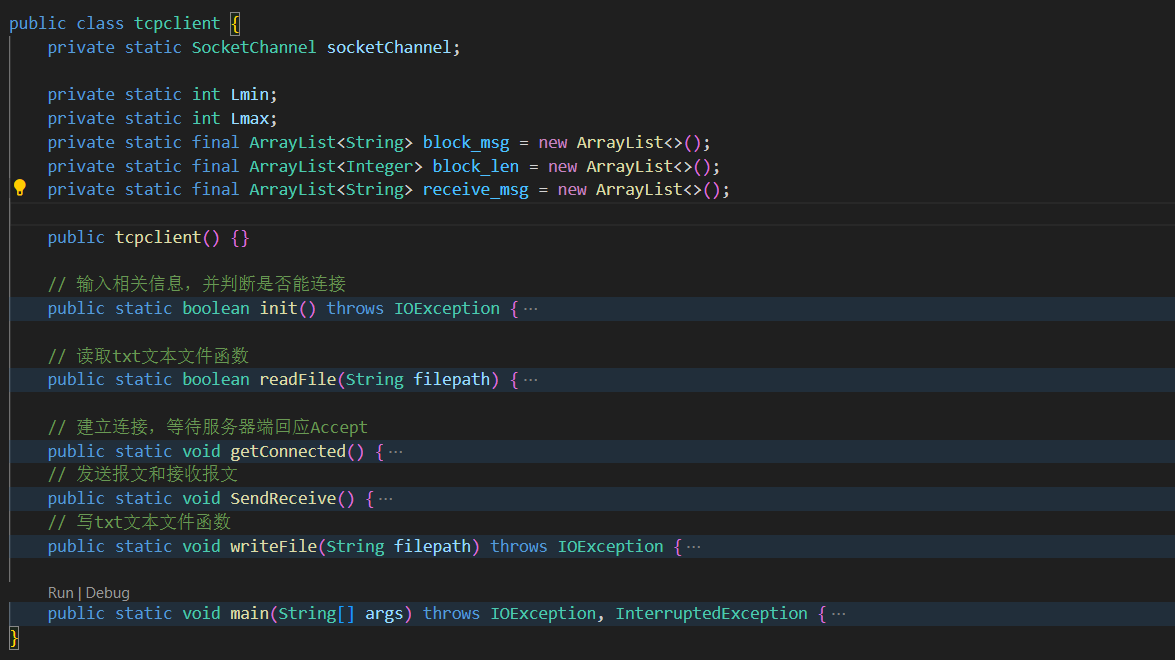
SelectionKey代表了Selector监控的通道和事件。每个注册到Selector的通道会生成一个SelectionKey，用于标识该通道的I/O事件（如OP\_ACCEPT、OP\_READ等）。SelectionKey的isAcceptable()和isReadable()方法用于检查特定事件是否已经就绪。

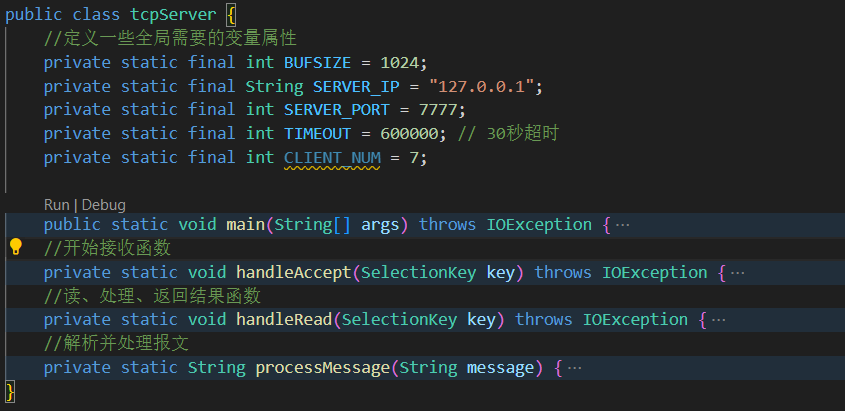
**2.报文的设计与封装**

一个合理的报文设计可以有效的提高信息处理的效率，因此单独设计一个模块来封装与解析报文是必要的。在这一模块中，我们需要定义相关的属性、报文结构以及协议的规则（也就是相关的函数）。报文的每个字段都应有明确的定义，包括字段的长度、类型和顺序。确保接收方能够正确解析和理解报文。要有明确的协议规则，确保通信双方按照相同的规则进行交互。下面是我设计的报文封装与解析：

1. **public** **class** tcpheader {
2. **private** **int** type;
3. **private** **int** N;
4. **private** **int** Length;
5. **private** String Data;
7. **public** tcpheader() {}
8. **public** tcpheader(**int** type, **int** N, **int** Length, String Data) {
9. **this**.type = type;
10. **this**.N = N;
11. **this**.Length = Length;
12. **this**.Data = Data;
13. }
15. **public** **int** getType() {
16. **return** type;
17. }
19. **public** **void** setType(**int** type) {
20. **this**.type = type;
21. }
23. **public** **int** getN() {
24. **return** N;
25. }
27. **public** **void** setN(**int** n) {
28. N = n;
29. }
31. **public** **int** getLength() {
32. **return** Length;
33. }
35. **public** **void** setLength(**int** length) {
36. Length = length;
37. }
39. **public** String getData() {
40. **return** Data;
41. }
43. **public** **void** setData(String data) {
44. Data = data;
45. }
47. @Override
48. **public** String toString() {
49. **return** type + String.format("%04d", N) + String.format("%03d", Length) + Data;
50. }
52. **public** tcpheader decode(String str) {
53. tcpheader tcpheader = **new** tcpheader();
54. tcpheader.setType(Integer.parseInt(str.substring(0, 1)));
55. tcpheader.setN(Integer.parseInt(str.substring(1, 5)));
56. tcpheader.setLength(Integer.parseInt(str.substring(5, 8)));
57. tcpheader.setData(str.substring(8));
58. **return** tcpheader;
59. }
60. }

**3.健壮性与模块的清晰化**

在完成本次计网udp大作业的过程中，我发现我编写代码时总是采用面向过程的思路进行，这就导致了代码冗长混乱的问题。在tcp的编写过程中，我着重注意了这一点，提前设计好了各个模块和相关的性质行为。这使得我的代码清晰条理，简单易懂。如图六、七所示。

图六、客户端的代码截图

图七、服务器端的代码截图

**三、学习与掌握的知识点：**

**1.TCP Socket编程：**

**1.1服务器端：**由于采用了非阻塞式的select组件，和标准的tcp socket编程过程有所区别。但是核心过程都包括了创建套接字，绑定套接字，监听端口，接受连接，处理连接，关闭连接这六大过程。

**1.2客户端：**客户端发起请求连接，服务器端相应并进行数据传输，关闭连接。

**1.3交互过程：**通过编程以及wireshark的抓包，我对tcp的交互过程有了更加清楚的认识，三次握手，四次释放清晰分明。并且为了达到wireshark抓包使过程更清晰这个目的，还利用了线程的休眠。

**2.构造应用层报文结构并进行交互**

构造应用层报文需要注意的点很多，包括报文的结构，协议的规则，报文的属性等等。还有一点需要注意的是，如果报文中可能包含中文，那么编码与解码方式要采用utf-8。在这次的设计过程中，我就着重按照上面的要点进行设计。相关的知识包括：

**2.1**报文的每个字段都应有明确的定义，包括字段的长度、类型和顺序。确保接收方能够正确解析和理解报文。比如固定头部的前三个部分，让他按照设计好的顺序与所占的位数进行交互。

**2.2**制定明确的协议规则，确保通信双方按照相同的规则进行交互。

**3.并发式多线程的使用**

**3.1Selector**是Java NIO库中的一个核心组件，用于实现非阻塞I/O操作。它允许一个单线程同时监控多个通道（Channel）的I/O事件（如连接、读取、写入等）。Selector能够监控多个通道，检测哪些通道准备好了某种I/O操作，从而提高资源利用率和系统性能。使用Selector可以使应用程序在处理I/O时不被阻塞，这样应用程序可以继续处理其他任务，提高了系统的并发性和响应性。有Selector.open()：打开一个新的Selector。select(long timeout)：阻塞直到至少有一个通道准备好I/O操作或者指定的超时时间过去。返回就绪通道的数量。selectedKeys()：返回SelectionKey集合，表示哪些通道已经准备好了I/O操作。

**3.2 ServerSocketChannel**是一个可以监听新进来的TCP连接的通道。它是非阻塞I/O的服务器端实现。ServerSocketChannel能够监听新进来的TCP连接，类似于传统的ServerSocket。可以配置为非阻塞模式，使得accept()方法不会阻塞，允许程序在没有连接请求时继续执行其他任务。有open()：打开一个新的ServerSocketChannel。bind(SocketAddress)：绑定到特定的地址（IP地址和端口）。configureBlocking(boolean)：配置该通道为阻塞或非阻塞模式。accept()：接受一个连接，在非阻塞模式下，如果没有连接准备好，返回null。

**3.3** **SelectionKey**是Selector监控的通道的标识符，用于区分不同的通道和它们的事件。代表了一个通道在Selector中注册的关系和对应的I/O事件。它包含了通道的注册信息、感兴趣的事件和已经就绪的事件。SelectionKey包含四种事件类型：连接（OP\_CONNECT）、接受（OP\_ACCEPT）、读取（OP\_READ）和写入（OP\_WRITE）。有hannel()：返回这个SelectionKey对应的通道。selector()：返回这个SelectionKey对应的Selector。interestOps()：获取或设置这个键感兴趣的操作集。readyOps()：返回这个键的通道已经准备好的操作集。isAcceptable()：检查这个键的通道是否准备好接受新的连接。isReadable()：检查这个键的通道是否准备好读取数据。isWritable()：检查这个键的通道是否准备好写入数据。

**四、URL：**