Lab7: 基于TCP的Socket编程

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

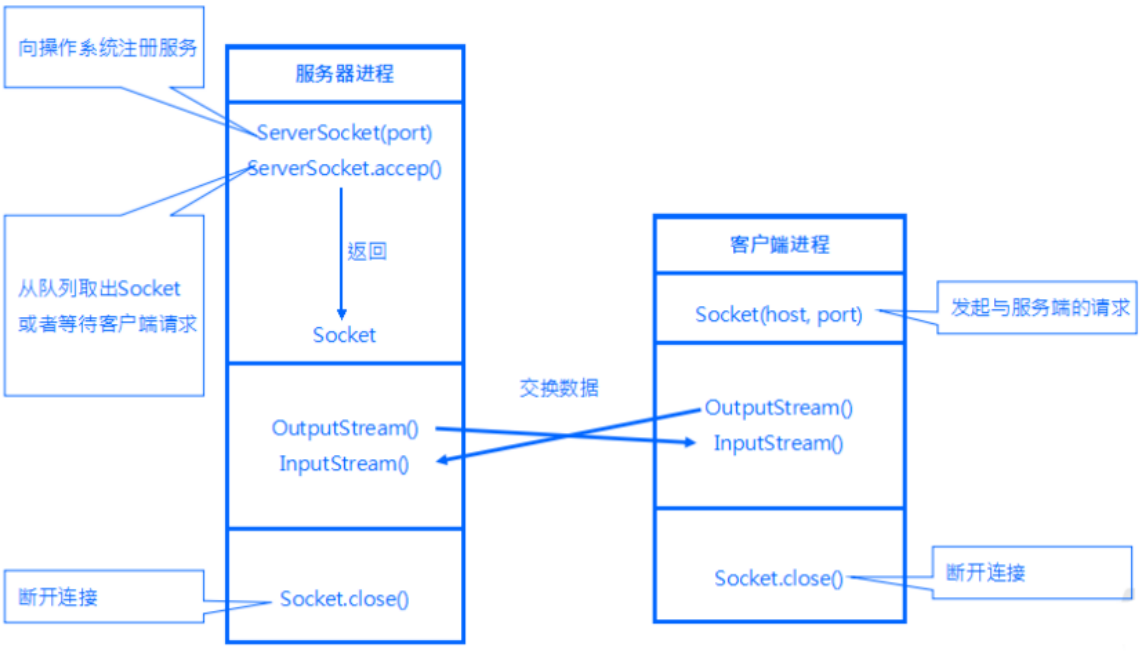
四、实验过程

1. 预备知识

1.1 Socket

> Socket(套接字)常被用来指操作系统提供的允许你通过TCP/IP协议栈进行一整套建立连接，发送数据，断开连接的过程的接口，不同的操作系统封装的socket接口函数可能有所不同。

1.2 Java中ServerSocket和Socket交互过程



1.2.1 建立服务器端步骤

- 创建ServerSocket对象，绑定监听端口
- 通过accept()方法监听客户端请求
- 连接建立后，通过输入流读取客户端发送的请求信息
- 通过输出流向客户端发送响应信息
- 关闭相关资源

1.2.2 建立客户端步骤

- 创建Socket对象，指明需要连接的服务器的地址和端口号
- 连接建立后，通过输出流向服务器发送请求信息
- 通过输入流获取服务器响应的信息
- 关闭相关资源

2. 客户端服务端交互信息

2.1 单服务端单客户端

- 编写TCPServer类

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class TCPServer {
    private ServerSocket serverSocket;
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    public void start(int port) throws IOException {
        // 1. 创建一个服务器端Socket，即ServerSocket，监听指定端口
        serverSocket = new ServerSocket(port);
        // 2. 调用accept()方法开始监听，阻塞等待客户端的连接
        System.out.println("阻塞等待客户端连接中...");
        clientSocket = serverSocket.accept();
        // 3. 获取Socket的字节输出流
        out = new PrintWriter(new OutputStreamWriter(clientSocket.getOutputStream(),
StandardCharsets.UTF_8), true);
        // 4. 获取Socket的字节输入流，并准备读取客户端发送的信息
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(),
StandardCharsets.UTF_8));
        // 5. 阻塞读取客户端发送的信息
        String str = in.readLine();
        System.out.println("我是服务端，客户端说：  " + str);
```

```
        // 消息回写
        out.println("服务端已收到消息" + str);
    }

    public void stop(){
        // 关闭相关资源
        try {
            if(in!=null) in.close();
            if(out!=null) out.close();
            if(clientSocket!=null) clientSocket.close();
            if(serverSocket!=null) serverSocket.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        int port = 9091;
        TCPServer server=new TCPServer();
        try {
            server.start(port);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            server.stop();
        }
    }
}
```

- 编写TCPClient类

```
import java.io.*;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class TCPClient {
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    public void startConnection(String ip, int port) throws IOException {
        // 1. 创建客户端Socket，指定服务器地址，端口
        clientSocket = new Socket(ip, port);
        // 2. 获取输入输出流
        out = new PrintWriter(new OutputStreamWriter(clientSocket.getOutputStream(),
StandardCharsets.UTF_8), true);
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(),
StandardCharsets.UTF_8));
    }
```

```
    public String sendMessage(String msg) throws IOException {
        // 3. 向服务端发送消息
        out.println(msg);
        // 4. 接收服务端回写信息
        String resp = in.readLine();
        return resp;
    }

    public void stopConnection() {
        // 关闭相关资源
        try {
            if(in!=null) in.close();
            if(out!=null) out.close();
            if(clientSocket!=null) clientSocket.close();
        } catch (IOException e){
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        int port = 9091;
        TCPClient client = new TCPClient();
        try {
            client.startConnection("127.0.0.1", port);
            String response = client.sendMessage("用户名: ECNUDaSE;");
            System.out.println(response);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            client.stopConnection();
        }
    }
}
```

- 修改TCPServer的代码，加入以下代码段，将读取一次客户端信息变为循环读取，并尝试让TCPClient发送多次消息。

```
String str;
while((str = in.readLine())!= null){
    System.out.println("我是服务端，客户端说：  " + str);
    // 消息回写
    out.println("服务端已收到消息" + str);
}
```

Task1: 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

2.2 单服务端多客户端

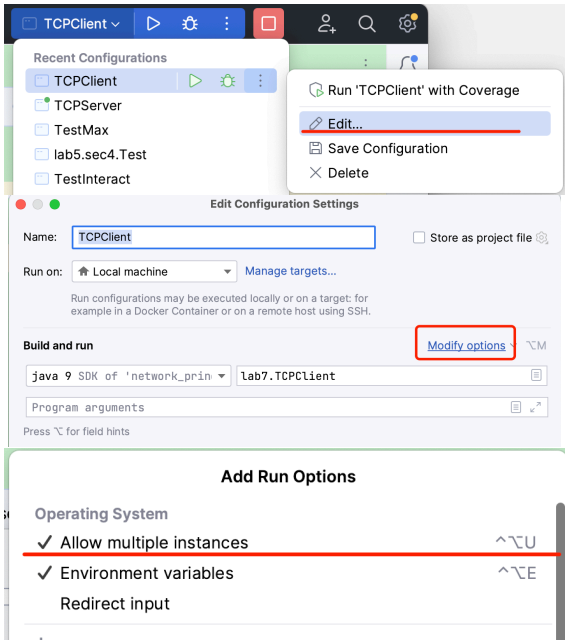
- 提供如下TCPServer类

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class TCPServer {
    private ServerSocket serverSocket;

    public void start(int port) throws IOException {
        // 1. 创建一个服务器端Socket，即ServerSocket，监听指定端口
        serverSocket = new ServerSocket(port);
        // 2. 调用accept()方法开始监听，阻塞等待客户端的连接
        for(;;){
            System.out.println("阻塞等待客户端连接中...");
            // TODO
        }

    }

    public void stop(){
        // 关闭相关资源
        try {
            if(serverSocket!=null) serverSocket.close();
        } catch (IOException e){
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        int port = 9091;
        TCPServer server=new TCPServer();
        try {
            server.start(port);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            server.stop();
        }
    }
}
```

```
class ClientHandler extends Thread {
    private Socket socket;
    ClientHandler(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        super.run();
        // TODO
    }
}
```

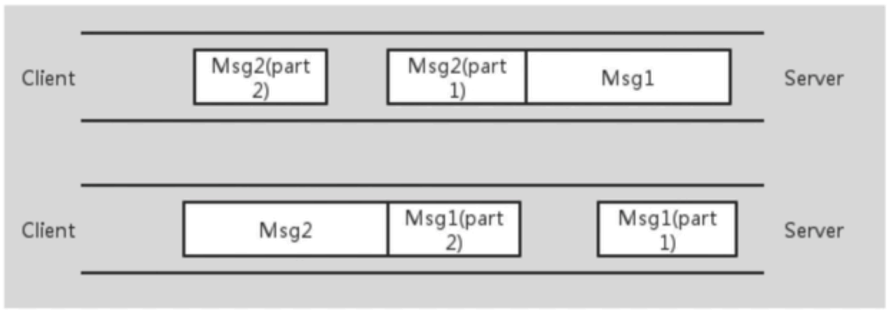
Task2: 修改代码使得每一次accept()的Socket都被一个线程接管，同时接管的逻辑保留Task1的功能，开启一个服务端和三个客户端进行测试，请将实现代码段及运行结果附在实验报告中。

- IDEA运行多个instances需要修改配置，参考如下，修改后可支持在IDEA窗口中同时运行多个TCPClient:



2.3 TCP半包粘包

- TCP协议是基于字节流的，本质上不存在半包粘包问题
- TCP的发送方一定会确保数据有序的到达接收方
- 所谓的半包粘包是应用层遇到的问题



- 提供如下TCPServer类和TCPClient类

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class TCPServer {
    private ServerSocket serverSocket;
    private Socket clientSocket;
    private static int BYTE_LENGTH = 64;

    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();
        for(;;) {
            byte[] bytes = new byte[BYTE_LENGTH];
            // 读取客户端发送的信息
            int cnt = is.read(bytes, 0, BYTE_LENGTH);
            if(cnt>0)
            System.out.println("服务端已收到消息: " + new String(bytes).trim());
        }
    }

    public void stop(){
        // 关闭相关资源
        try {
```

```
            if(clientSocket!=null) clientSocket.close();
            if(serverSocket!=null) serverSocket.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        int port = 9091;
        TCPServer server=new TCPServer();
        try {
            server.start(port);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            server.stop();
        }
    }
}
```

```
import java.io.*;
import java.net.Socket;

public class TCPClient {
    private Socket clientSocket;
    private OutputStream out;

    public void startConnection(String ip, int port) throws IOException {
        clientSocket = new Socket(ip, port);
        out = clientSocket.getOutputStream();
    }

    public void sendMessage(String msg) throws IOException {
        // 重复发送十次
        for(int i=0;i<10;i++){
            out.write(msg.getBytes());
        }
    }

    public void stopConnection() {
        // 关闭相关资源
        try {
            if(out!=null) out.close();
            if(clientSocket!=null) clientSocket.close();
        } catch (IOException e){
            e.printStackTrace();
        }
    }
```

```
    public static void main(String[] args) {
        int port = 9091;
        TCPClient client = new TCPClient();
        try {
            client.startConnection("127.0.0.1", port);
            String message = "NETWORK PRINCIPLE";
            client.sendMessage(message);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            client.stopConnection();
        }
    }
}
```

- **Task3: 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。**