**深 圳 大 学 实 验 报 告**

**课程名称： 互联网编程**

**实验项目名称：实验2 多线程/线程池TCP服务器端程序设计**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 毛斐巧**

**报告人： 吴嘉楷 学号： 2022150168 班级： 国际班**

**实验时间： 2024.3.27、4.3、10、17（周三）**

**实验报告提交时间： 2024.4.5**

**教务处制**

一、实验目的与内容：

**目的：**熟悉java线程编程技术，掌握线程技术在JAVA互联网通信程序中的应用。

**内容要求：**

1. 多线程TCP服务器（30分）：

改写socket服务器端程序（即实验1中参考example11-5中server.java的编程结果），设计编写出一个TCP服务器端程序：要求使用多线程处理多个客户端的连接请求（每个线程实例处理一个客户端连接）。客户端与服务器端之间的通信内容，以及服务器端的处理功能等可自由设计拓展，也可保持实验1中的通信内容，无特别限制和要求。

1. 线程池TCP服务器（40分）：

改写socket服务器端程序（即实验1中参考example11-5中server.java的编程结果），设计编写出一个TCP服务器端程序：要求使用线程池处理客户端的连接请求。相应需修改socket客户端程序：从本地一个txt文件（注意：客户端的此txt文件称为信源txt文件，该txt测试数据文件可自由选择或创建，只要里面包含至少10行字符串即可），逐行读取出来，每一行作为一句话，发送给服务器，服务器每收到一个客户端的一句话，就将其保存在服务器本地的一个专用于存放与该客户端通信内容的txt文件中（注意：称为通信记录txt文件，来自不同的客户端的信息需保存在不同的txt文件中，需多少个txt文件应在程序中自动根据实际连接的客户端数量自动创建），服务器还会计算每一个通信记录txt文件的安全算法摘要，将安全算法摘要和该通信记录txt文件名一起写入到服务器中一个名为SafeAbstract.txt的文件中。

1. 编程扩充线程池TCP服务器的功能，增加日志功能模块：日志记录的内容和日志存储方式可自定（比如可以记录客户端的连接时间、客户端IP等，日志存储为.TXT或.log文件等），在线程池服务器程序中调用该日志程序模块，使线程池TCP服务器具备日志功能，注意线程之间的同步操作处理。（20分）

注意：

1. 实验报告中需要有实验结果的截屏图像。

二、实验过程和代码与结果

**1. 给出满足内容要求1的程序源码及运行结果，简述思路或实验过程。**

程序简述：TCP服务端采用多线程模式处理多个客户端的连接请求，通过循环监听客户端连接并启动对应线程，从而实现异步工作模式。多个客户端需要通过建立多个异名程序来实现，便于发送不同的消息。

思路：

1. 创建多个客户端程序Client、一个服务端程序Sever.java以及一个自定义线程类myRunnable（实现Runnable接口）；

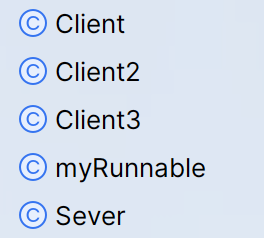


图1.1 多线程服务器源文件集

1. 编写客户端，客户端程序用于创建socket对象并连接服务端，进行收发服务端消息的业务功能。编写完成后，将代码（注意要修改类名）复制粘贴至另外两个客户端代码文件中。

*//引入相关模块*import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.net.Socket;  
import java.util.Scanner;  
  
*//客户端*public class Client {  
 public static void main(String[] args) throws IOException {  
 *//创建socket对象 并 连接服务端* Socket socket = new Socket("127.0.0.1",6666);  
 *//创建Scanner对象 以 手动输入数据* Scanner sc = new Scanner(System.*in*);  
 *//从socket中获取输出流、输入流* OutputStream os = socket.getOutputStream();  
 InputStream ins = socket.getInputStream();  
 *//将字节流进行转化，以防止 中文乱码 问题* InputStreamReader insReader = new InputStreamReader(ins);  
 *//接受服务器发来的提示消息* int ch;  
 while ((ch = insReader.read()) != -1) {  
 *//将依次读取到的字符进行输出* System.*out*.print((char)ch);  
 if(ch=='\n')break;  
 }  
 *//不断发送信息给服务器* while (true){  
 System.*out*.print("请输入您想对小墙墙说的话：");  
 *//手动输入消息* String str = sc.nextLine();  
 *//循环退出条件* if(str.equals("拜拜") || str.equals("88"))break;  
 *//发送消息给 服务端* os.write((str+"\n").getBytes());  
 *//接受服务器发来的反馈消息* while ((ch = insReader.read()) != -1) {  
 *//将依次读取到的字符进行输出* System.*out*.print((char)ch);  
 if(ch=='\n')break;  
 }  
 }  
 System.*out*.println("会话结束，小墙墙还会想你的！");  
 *//释放socket资源* socket.close();  
 }  
}

1. 编写自定义线程线程，用于收发客户端消息，可以理解为每个线程就是一个一对一服务的服务端。

import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.net.Socket;  
  
public class myRunnable implements Runnable{  
 Socket socket;  
 String name;  
 public myRunnable(Socket socket) {  
 this.socket = socket;  
 }  
 @Override  
 public void run() {  
 this.name = Thread.*currentThread*().getName();  
 try {  
 *//从socket中获取输入流、输出流* InputStream ins = socket.getInputStream();  
 OutputStream os = socket.getOutputStream();  
 *//将字节流进行转化，以防止 中文乱码 问题* InputStreamReader insReader = new InputStreamReader(ins);  
 *//向客户端发送通知，告知通信连接已建立* os.write("尊敬的用户，欢迎来到留言墙，小墙墙将耐心倾听您的想法！\n".getBytes());  
 *//接受客户端发来的消息* int ch,flag=1;  
 while ((ch = insReader.read()) != -1) {  
 if(flag==1) System.*out*.print("用户"+name+"留言：");  
 flag = 0; *//防止 每接受一个字符就打印一次提示语  
 //将依次读取到的字符进行输出* System.*out*.print((char)ch);  
 if((char)ch == '\n'){ *//已接收到一句完整的消息* flag = 1; *//可以打印提示语  
 //向客户端发送通知* os.write("小墙墙已收到了您的留言！\n".getBytes());  
 }  
 }  
 System.*out*.println("用户"+name+"关闭了通信！");  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 } finally {  
 if(socket!=null){  
 *//释放socket和服务器资源* try {  
 socket.close();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 }  
}

1. 编写服务端程序：创建服务端对象，通过循环监听客户端的连接请求，一旦接收到请求，即开启线程去服务对应的客户端。

*//引入相关模块*import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
*//服务端*public class Sever {  
 public static void main(String[] args) throws IOException {  
 *//创建服务端对象 并 绑定 6666端口* ServerSocket serverSocket = new ServerSocket(6666);  
  
 while (true) {  
 *//监听等待客户端连接* Socket socket = serverSocket.accept();  
 *//开启一条线程* Thread t = new Thread(new myRunnable(socket));  
 t.start();  
 *//TCP连接已建立* System.*out*.println("用户"+t.getName()+"连接上了本服务器！");  
 }  
 }  
}

运行结果：

A.当三个客户端程序都连接上服务器后，分别向服务器发送消息：

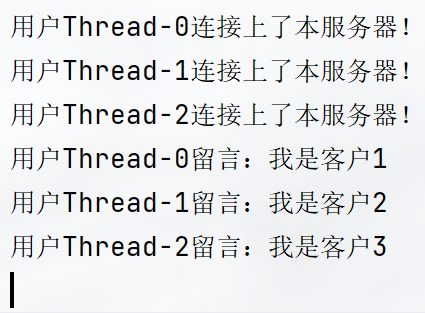


图1.2 服务器的输出日志以及收到的消息

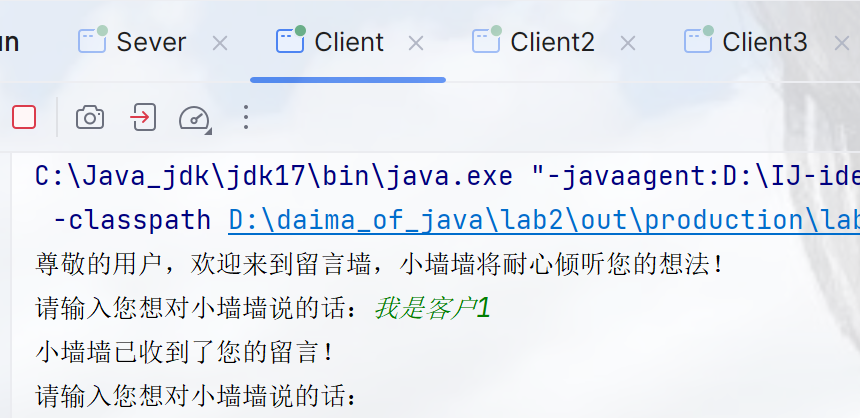


图1.3 客户端1号

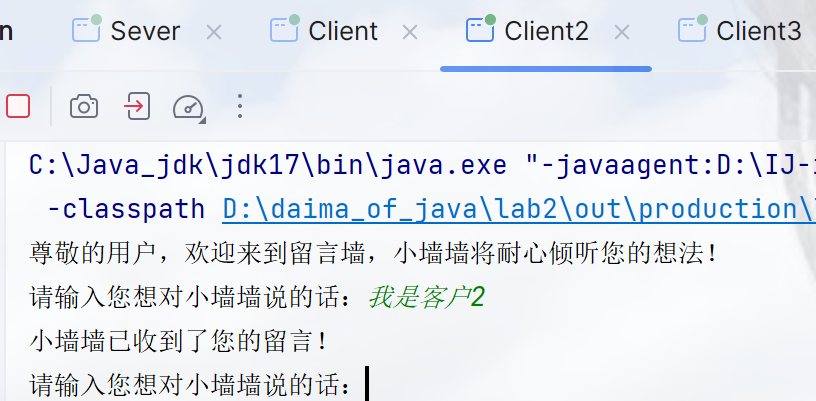


图1.4 客户端2号

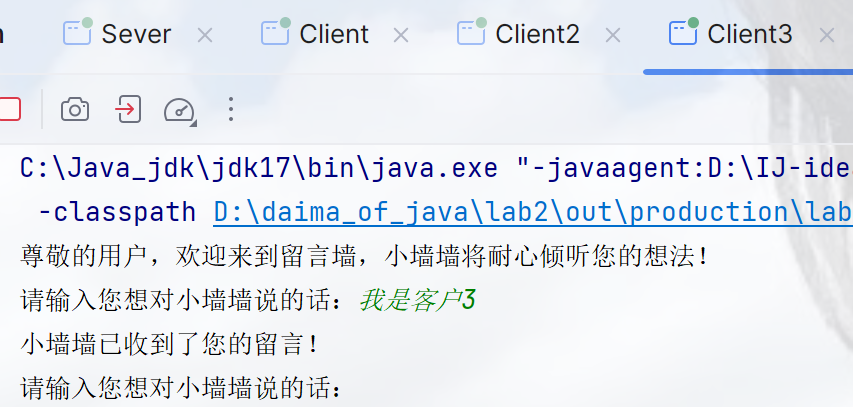


图1.5 客户端3号

B.3个客户端程序均向服务器发送中断请求消息“88”：

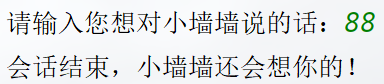


图1.6 客户端发送中断请求

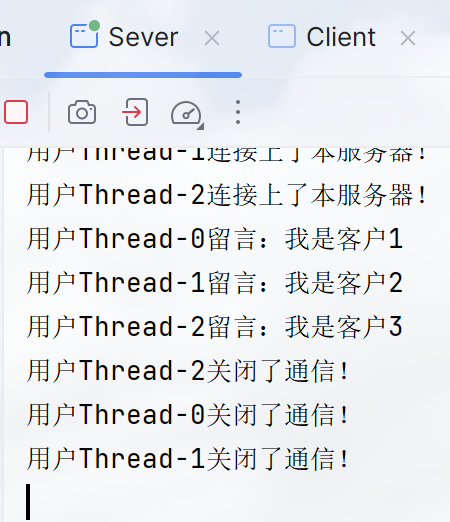


图1.7 客户端关闭连接后服务器的输出日志

**2. 给出满足内容要求2的程序源码及运行结果，简述思路或实验过程。**

程序简述：

TCP服务端采用**线程池**模式处理多个客户端的连接请求，先在服务端创建线程池对象，通过循环监听客户端连接并向线程池提交相应的任务，从而实现异步工作模式。多个客户端需要通过多线程来实现，便于同时从信源txt文件中读取数据并发送给服务端。

思路：

1. 创建信源txt文件，并向其中输入至少10行字符串内容



图2.1 信源txt文件的内容

1. 创建一个客户端程序Thread\_Client.java、服务端程序Pool\_Sever.java以及两个自定义线程类（一个ClientRunnable用于实现客户端任务，另一个myRunnable用于实现服务端的任务），同时创建一个txtFiles文件夹，用于存放来自不同的客户端的通信内容的txt文件。

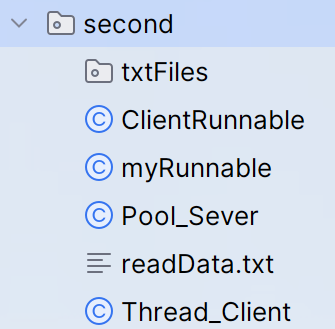


图2.2 项目的目录结构

1. 编写客户端自定义线程ClientRunnable

在ClientRunnable中，建立一个socket对象连接服务端IP地址127.0.0.1的6666端口，然后利用BufferedReader缓冲流逐行读取信源txt文件的消息，并逐一发送给服务端，当读取完整个信源txt文件后，关闭socket对象。

源代码：

package second;*//引入相关模块*import java.io.\*;  
import java.net.Socket;  
import java.util.Scanner;  
  
public class ClientRunnable implements Runnable{  
 Socket socket;  
 public ClientRunnable() throws IOException {  
 *//创建socket对象 并 连接服务端* this.socket = new Socket("127.0.0.1",6666);  
 System.*out*.println("连接已建立，开始读取信源txt文件并发送至服务器...");  
 }  
 @Override  
 public void run() {  
 try {  
 *//创建Scanner对象 以 手动输入数据* Scanner sc = new Scanner(System.*in*);  
 *//从socket中获取输出流、输入流* OutputStream os = socket.getOutputStream();  
 InputStream ins = socket.getInputStream();  
 *//将字节流进行转化，以防止 中文乱码 问题* InputStreamReader insReader = new InputStreamReader(ins);  
 *//从信源txt文件中读取数据，并发送给服务端* String words, filePath = "second/readData.txt";  
 BufferedReader reader = new BufferedReader(new FileReader(filePath));  
 while( (words=reader.readLine()) != null){ *//一次读一整行  
 //发送给服务端* os.write((words+'\n').getBytes());  
 }  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 } finally {  
 if(socket!=null){  
 *//释放socket和服务器资源* try {  
 System.*out*.println("TCP连接关闭！");  
 socket.close();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 }  
}

1. 编写客户端程序

在客户端程序Thread\_Client中，首先静态指定要开启的客户端数量，然后通过循环开启相应个数的线程，异步地去完成ClientRunnable实现类的任务。

源代码：

package second;*//引入相关模块*import java.io.IOException;

*//客户端*public class Thread\_Client {  
 public static void main(String[] args) throws IOException {  
 *//线程数量，即客户端数量* int total = 6;  
 while(total>0){  
 *//创建并开启线程* Thread t = new Thread(new ClientRunnable());  
 t.start();  
 total--;  
 }  
  
 }  
}

1. 编写服务端自定义线程myRunnable

在myRunnable创建时，会获取到服务端程序传入socket对象和安全算法摘要的FileWriter实现对象，然后逐行接收不同客户端发来的消息并存入txtFiles目录下的不同的txt文件中。当文件写入完毕后，在将该文件安全算法摘要和该通信记录txt文件名一起写入SafeAbstract.txt中。

源代码：

package second;  
import java.io.\*;  
import java.net.Socket;  
import java.security.DigestInputStream;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
  
public class myRunnable implements Runnable{  
 Socket socket;  
 String name;  
 FileWriter safeAbstractWriter;  
 static int *index* = 1;  
 public myRunnable(Socket socket, FileWriter fw) {  
 this.socket = socket;  
 this.safeAbstractWriter = fw;  
 }  
 @Override  
 public void run() {  
 this.name = Thread.*currentThread*().getName();  
 try {  
 *//从socket中获取输入流、输出流* InputStream ins = socket.getInputStream();  
 OutputStream os = socket.getOutputStream();  
 *//将字节流进行转化,以便于一次读写一行数据* BufferedReader reader = new BufferedReader(new InputStreamReader(ins));  
 *//接受客户端发来的消息* String words;  
 *//创建文件输入流* String fileRoot = "second/txtFiles/";  
 String fileName = "tcpContent\_of\_poolThread" + *index*++ + ".txt";  
 String filePath = fileRoot + fileName;  
 FileWriter fileWriter = new FileWriter(filePath);  
 while ((words = reader.readLine()) != null) {  
 *//追加至相应文件中* fileWriter.append(words+"\n");  
 }  
 fileWriter.flush();  
 fileWriter.close();  
 *//计算文件的安全算法摘要* FileInputStream in = new FileInputStream(filePath);  
 MessageDigest sha = MessageDigest.*getInstance*("SHA-256");  
 DigestInputStream din = new DigestInputStream(in,sha);  
 while(din.read()!=-1);  
 din.close();  
 *//将安全算法摘要和该通信记录txt文件名一起写入SafeAbstract.txt中* safeAbstractWriter.write("文件名："+fileName+"\n安全算法摘要："+sha.digest()+"\n\n");  
 safeAbstractWriter.flush();  
 } catch (IOException | NoSuchAlgorithmException e) {  
 throw new RuntimeException(e);  
 } finally {  
 if(socket!=null){  
 *//释放socket和服务器资源* try {  
 socket.close();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 }  
}

1. 编写服务端程序

在服务端程序Pool\_Sever中，首先创建一个服务端对象并绑定 6666端口，然后创建一个线程池对象以及一个写入安全算法摘要记录的文件输出流，然后通过循环去监听客户端连接，一旦有客户端连接本服务端，便将相应的业务提交给线程池执行。

源代码：

package second;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.concurrent.\*;  
  
*//服务端*public class Pool\_Sever {  
 public static void main(String[] args) throws IOException {  
 *//创建服务端对象 并 绑定 6666端口* ServerSocket serverSocket = new ServerSocket(6666);  
 *//创建线程池对象* ThreadPoolExecutor pool = new ThreadPoolExecutor(  
 3, *//核心线程数量* 16, *//线程池总大小* 60, *//空闲时间* TimeUnit.*SECONDS*, *//单位* new ArrayBlockingQueue<>(2), *//阻塞队列* Executors.*defaultThreadFactory*(), *//线程工厂* new ThreadPoolExecutor.AbortPolicy() *//任务拒绝策略* );  
 *//创建文件输出流，用于写入安全算法摘要和该通信记录txt文件名* FileWriter fileWriter = new FileWriter("second/SafeAbstract.txt");  
 while (true) {  
 *//监听等待客户端连接* Socket socket = serverSocket.accept();  
 *//把任务题交给线程池* myRunnable Task = new myRunnable(socket, fileWriter);  
 pool.submit(Task);  
 *//TCP连接已建立* System.*out*.println("用户连接上了本服务器！");  
 }  
 }  
}

1. 执行结果

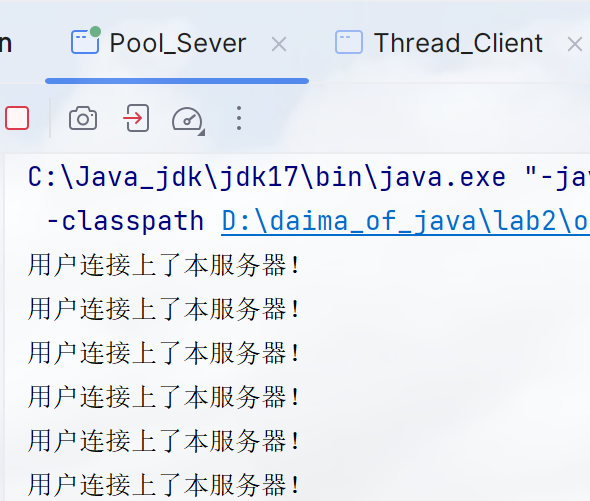
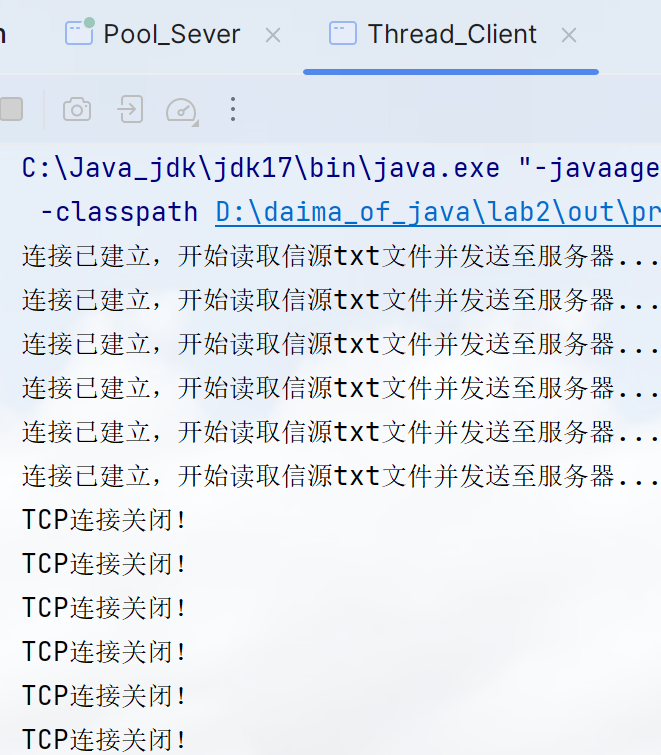
 

图2.3 控制台输出（左服务端，右客户端）

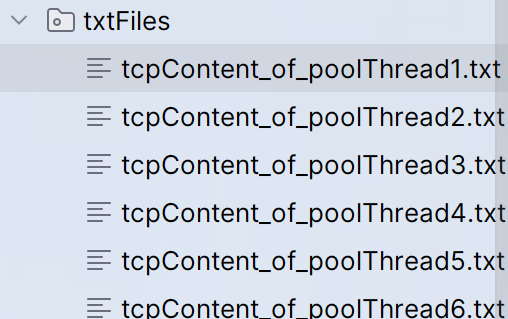


图2.4 生成的所有通信内容txt文件



图2.5 通信内容txt文件的详细内容

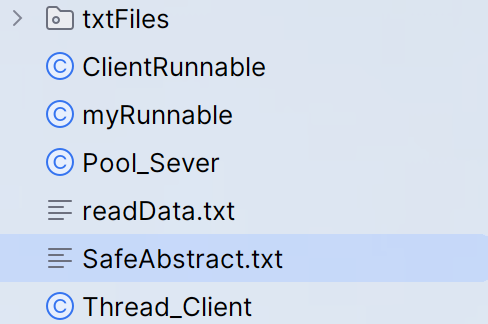


图2.6 生成的安全算法摘要记录的txt文件



图2.7 生成的文件算法摘要记录的内容

**3.给出满足内容要求3的实验结果，包括源码及线程池服务器增加了日志功能后，日志记录运行结果。**

程序简述：

编程扩充线程池TCP服务器的功能，增加日志功能模块：记录客户端的连接时间、客户端IP等，日志存储格式为.log文件，在线程池服务器程序中调用该日志程序模块，使线程池TCP服务器具备日志功能。

1. 将要求2的代码拷贝一份，迁移至third目录下，必要时修改相关代码（如文件路径）

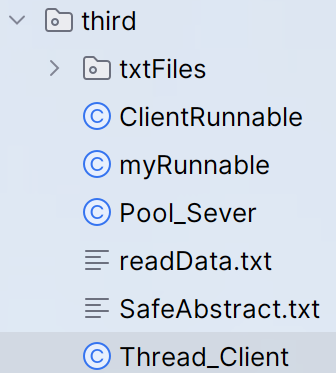


图3.1 迁移后的目录结构

1. 在third目录下创建LogUtil.java文件，用于实现日志功能模块

由于服务端是永远运行着的，日志需要记录下历史信息，所以需要以追加的方式写入.log日志文件。只需要在创建FileWriter对象时传入第二个参数为true即可。



图3.2 创建FileWriter对象并指定为追加写入模式

LogUtil类通过接收自定义线程类myRunnable的TCP连接建立时间、线程名字、需要创建的日志文件路径、客户端IP从而获得并写入日志内容。

源代码：

*//服务器日志功能模块*package third;  
  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class LogUtil {  
 String filePath;  
 String time;  
 String severName;  
 String clientIP;  
  
 public LogUtil(String time, String severName, String filePath, String IP) throws IOException {  
 this.time = time;  
 this.severName = severName;  
 this.filePath = filePath;  
 this.clientIP = IP;  
 }  
  
 public void createLog() throws IOException {  
 FileWriter fileWriter = new FileWriter(filePath+".log",true);  
 fileWriter.append("连接时间："+time+"\n"+  
 "服务线程名称："+severName+"\n"+  
 "客户端IP："+clientIP+"\n\n");  
 fileWriter.flush();  
 fileWriter.close();  
 }  
}

1. 自定义线程类myRunnable的代码扩展

首先，增加一个getTime函数用于获取调用时的系统时间：

public String getTime(){  
 LocalDate localDate = LocalDate.*now*();*//获取日期* LocalTime localTime = LocalTime.*now*();*//获取时间* return localDate.toString()+" "+localTime.toString().substring(0,8);  
}

其次，在run方法开始时，获取TCP连接建立时的系统时间以及客户端IP：



图3.3 获取相关信息

最后，在关闭socket对象之前，调用日志功能模块LogUtil类，输出日志文件：

*//调用日志功能模块，输出日志文件*LogUtil logUtil = new LogUtil(time, this.name, "third/mySeverLog", IP);  
logUtil.createLog();

1. 运行结果

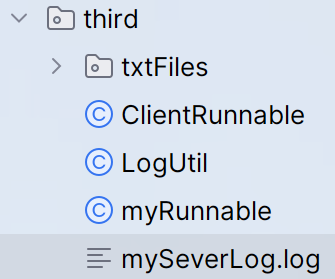


图3.4 目录下创建了日志文件



图3.5 日志文件内容

三、实验总结

（此处写你的过程，比如遇到的错误，以及解决方法，你的所想、所得）

1. 客户端与服务端之间的通信阻塞了

原因：read()方法没有读到结束标志，客户端一直处于读取状态。

解决方案：在read()方法的循环里增加一个结束标志“\n”，在服务端相应的提示消息结尾增加此字符作为结束标志。

1. 对java多线程和线程池的知识有些陌生

解决方案：在bilibili网站观看黑马程序员java课程，学习了多线程和线程池的知识

1. 在自定义线程类的run方法中使用socket但出现报错

解决方案：由于run方法并未抛出异常，于是在此只能使用try--catch--finally来解决。

1. 不清楚如何使用文本输入流逐行读入文本数据

解决方案：将文本输入流使用BufferedReader类封装，转变为缓冲流，便可以使用readLine方法读取一整行的内容。

1. 使用服务端接收客户端发来的一行消息时，发现消息并不是逐行发送过来的

原因：客户端使用readLine读取文本内容时，不会保留每行结尾的换行符“\n”，于是在服务端看来并非是逐行的通信方式。

解决方案：在客户端发送一行消息时，手动在结尾拼接上一个“\n”。

1. 不清楚如何获取当前时间以及客户端IP地址

解决方案：在csdn博客网站查询相关信息，了解到可以使用LocalDate、LocalTime、LocalDateTime时间类来获取当地日期时间，可以使用socket的getInetAddress方法获取客户端IP地址。

1. 在写入日志内容时需要使用文件的追加写入模式

解决方案：询问AI对话模型可知，只需要在创建文件输出流时指定第二个参数为true，即为追加写入模式。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：毛斐巧  2024 年 月 日 |
| 备注： |