**安全科学与工程学院**

**2022－2023 学年第 1 学期**

**网络编程技术大作业**

**题目：航班显示系统仿真网络软件设计与实现**

**——含综合管理系统设计与实现**

**班级： 20034201**

**学号： 200340164**

**姓名： 卡通人物

低可信度描述已自动生成**

目 录

1. 系统概述 3

2. 开发环境 3

3. 系统总体设计 4

4. 项目程序包/类设计 4

5. 各功能模块的详细设计和测试 4

6. 项目分析总结 21

7. 总结 22

8. 参考文献 22

# 航班显示系统仿真网络软件设计与实现——含综合管理系统设计与实现

## 系统概述

选题：

航班显示系统 仿真网络软件 设计与实现— —含综合管理 系统设计与实 现

系统实现功能：

1. 基于 TCP 协议的航班显示系统服务器的设计与实现
2. 基于 TCP 的航班显示系统客户端设计与实现
3. 基于 UDP 的航班显示系统服务器和客户端的设计与实现
4. GUI 界面航班显示服务器的管理程序
5. GUI 航班显示客户端的管理程序

背景知识：

航班信息显示系统，是基于计算机网络，由系统软件、终端显示设备、服务器和存储设备构成，通过终端显示设备向旅客和机场工作人员发布航班计划和动态信息，提供值机、候机、登机、行李提取、行李分拣等信息的显示系统。

本系统基于java语言开发，同时兼具服务器和客户端的功能，能够实现客户端和服务器的通信和客户端的航班信息动态显示。

## 开发环境

jdk 17

idea 2022.3

eclipse

windowbuilder

## 系统总体设计

系统设计思路（包含系统架构、功能模块设计等）。

服务器：

同时支持TCP和UDP的连接。

输入地址和端口配置

按下lanuch按钮开启TCP或者UDP服务器。

当客户端建立连接的时候在滚动面板上显示与服务器建立的客户端的信息。

按下shutdown按钮即可关闭对应的服务器

客户端：

下拉框选择TCP或者UDP连接。

输入地址和端口后点击launch按钮，启动对应的服务。

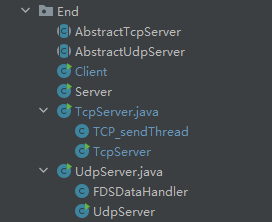
点击pause按钮暂停对应的服务

## 项目程序包/类设计

列出主要程序包/类的设计并加以说明。

本程序在end包下：

结构如下：



其中服务器程序是Server,java

客户端程序是Client.java

AbstractTcpServer为TCP服务器的抽象类

AbstractUdpServer为UDP服务器的抽象类

其中TcpServer继承了其抽象类

UdpServer继承了其抽象类

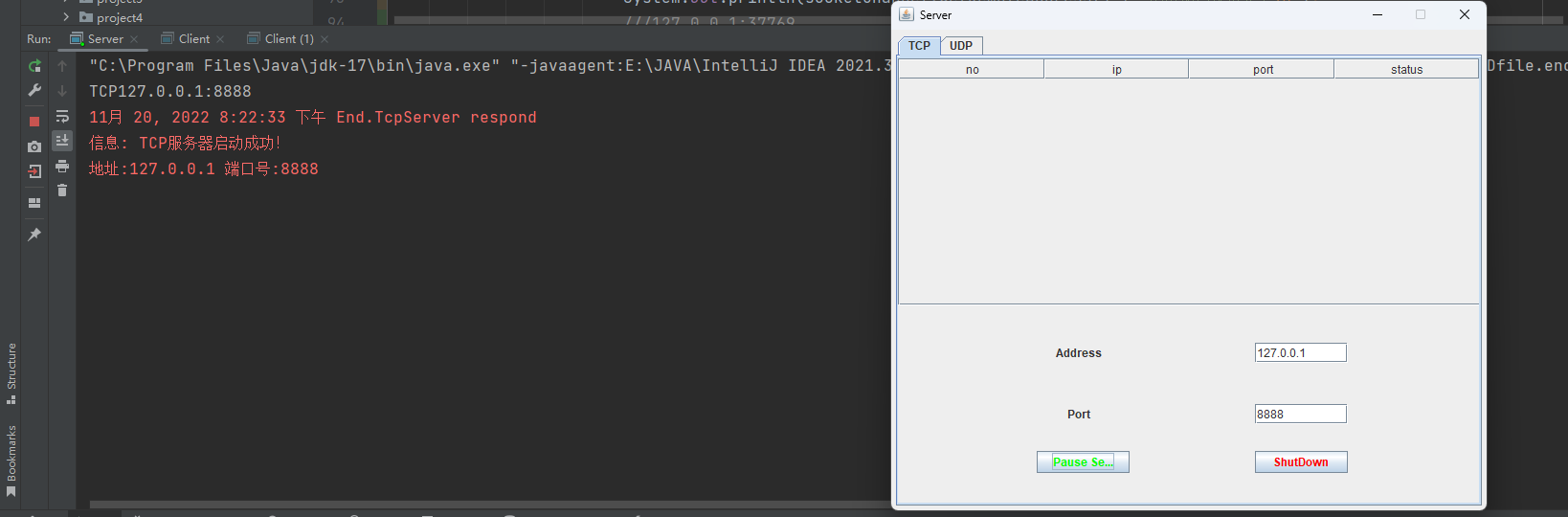
## 各功能模块的详细设计和测试

* 1. **基于TCP协议的航班显示系统服务器的设计与实现**
     1. 服务器的启动，包括服务端口的选择和绑定

TCP服务器的启动：

1. */\*\**
2. \* TCP的启动
3. \*/
4. JButton TCP\_launchButton = new JButton("Launch");
5. TCP\_launchButton.addActionListener(new ActionListener() {
6. @Override
7. public void actionPerformed(ActionEvent e) {
8. *//TCP address 非空*
9. if (!"".equals(TCP\_addressText.getText()))
10. TCP\_Address = TCP\_addressText.getText();
11. *//TCP port 非空*
12. if (!"".equals(TCP\_portText.getText()))
13. TCP\_Port = TCP\_portText.getText();
14. *//配置TCP服务器*
15. setTcpServer(TCP\_Address, TCP\_Port);
16. *//启动TCP服务器的线程*
17. TCP\_thread.start();
18. System.out.println("TCP" + TCP\_Address + ":" + TCP\_Port);
19. TCP\_launchButton.setText("Pause Server");
20. }
21. });

测试：



测试开启TCP服务器成功！

* + 1. 服务器接收客户端连接

TCP：

1. *// 接受连接*
2. if (key.isAcceptable()) { *// key 就绪事件抓手*
3. ServerSocketChannel serverSocketChannel = (ServerSocketChannel) key.channel(); *// 所有信息都找key要*
4. SocketChannel socketChannel = serverSocketChannel.accept(); *// 创建SocketChannel*
5. *// 配置新生的SocketChannel*
6. socketChannel.configureBlocking(false);
7. SelectionKey selectionKey = socketChannel.register(selector, SelectionKey.OP\_WRITE);
8. System.out.println(socketChannel.getRemoteAddress() + "已连接"); *// 客户端地址*

使用工厂方法创建select，配置select并监听，每次取出key值来处理连接请求

* + 1. 服务器处理客户端连接（如何接收客户端的数据或向客户端发送数据）

TCP:

1. SocketChannel socketChannel = (SocketChannel) key.channel();
2. ByteBuffer bufferToSend = (ByteBuffer) key.attachment(); *// 取出附件*
3. if (TCP\_currentThreadIsShutDown){
4. *// 发送结束*
5. byte[] endMessage = "shutDown!".getBytes();
6. ByteBuffer endBuffer = ByteBuffer.wrap(endMessage);
7. socketChannel.write(endBuffer);
8. continue;
9. }
10. if (bufferToSend.hasRemaining()) {
11. socketChannel.write(bufferToSend);

当key.isWriteable的时候，服务器开始向客户端输出数据

* + 1. 服务器处理断开的客户端连接

TCP:

1. try {
2. key.channel().close();
3. } catch (IOException e1) {
4. e1.printStackTrace();
5. }
6. key.cancel();
7. *// 客户端主动断开*
8. if (e.getMessage().contains("你的主机中的软件中止了一个已建立的连接。")) {
9. System.out.println("客户端已主动断开连接\n");
10. }

使用trycatch捕获异常

* + 1. 服务器主动断开与客户端连接

TCP:

1. if (TCP\_currentThreadIsShutDown){
2. *// 发送结束*
3. byte[] endMessage = "shutDown!".getBytes();
4. ByteBuffer endBuffer = ByteBuffer.wrap(endMessage);
5. socketChannel.write(endBuffer);
6. continue;
7. }

当按下shutdown按钮的时候，同时设置TCPServer的服务器的此成员变量为true，当输出数据的时候，如果为true的话则说明服务器断开连接

* + 1. 服务器的并发服务功能的设计设计与测试

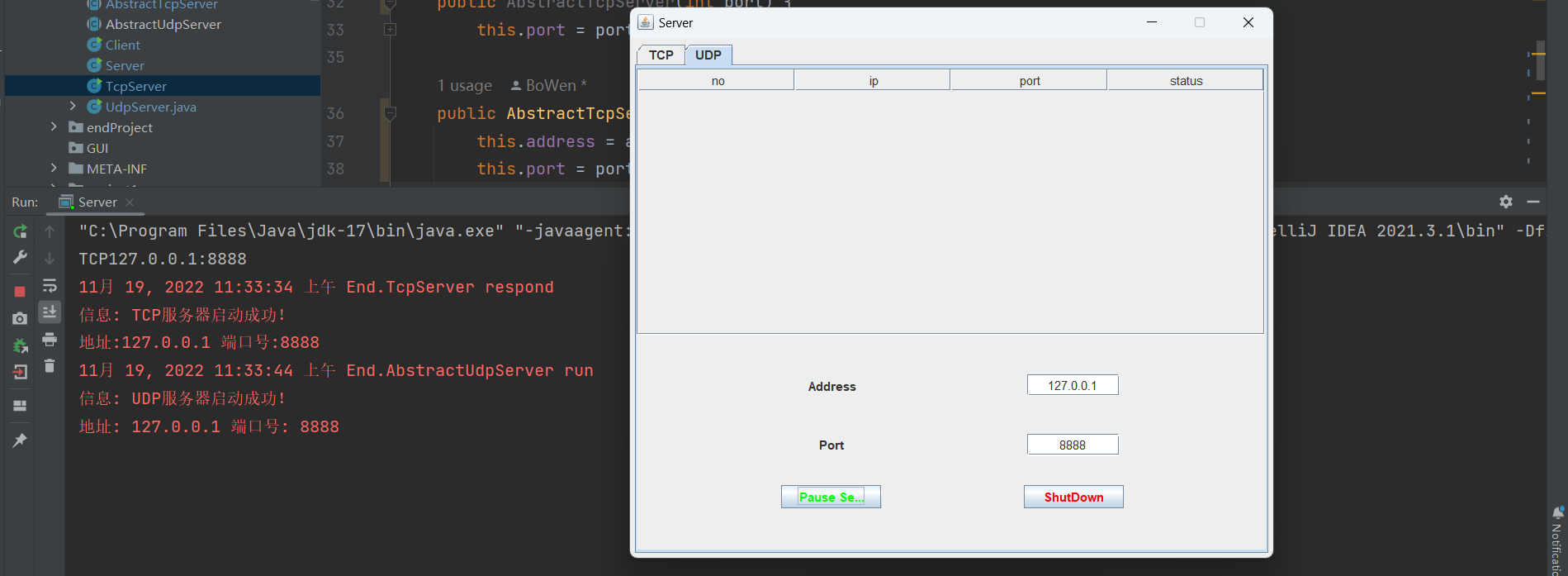
TCP:

1. *// 1.创建Selector对象，提供监听服务*
2. selector = Selector.open();
3. *// 2.创建serverSocketChannel，并将其绑定在指定端口，设成非阻塞工作模式*
4. ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
5. serverSocketChannel.bind(new InetSocketAddress(address, port));
6. serverSocketChannel.configureBlocking(false);
7. *// 3.serverSocket向Selector注册“接受连接就绪事件”*
8. serverSocketChannel.register(selector, SelectionKey.OP\_ACCEPT);

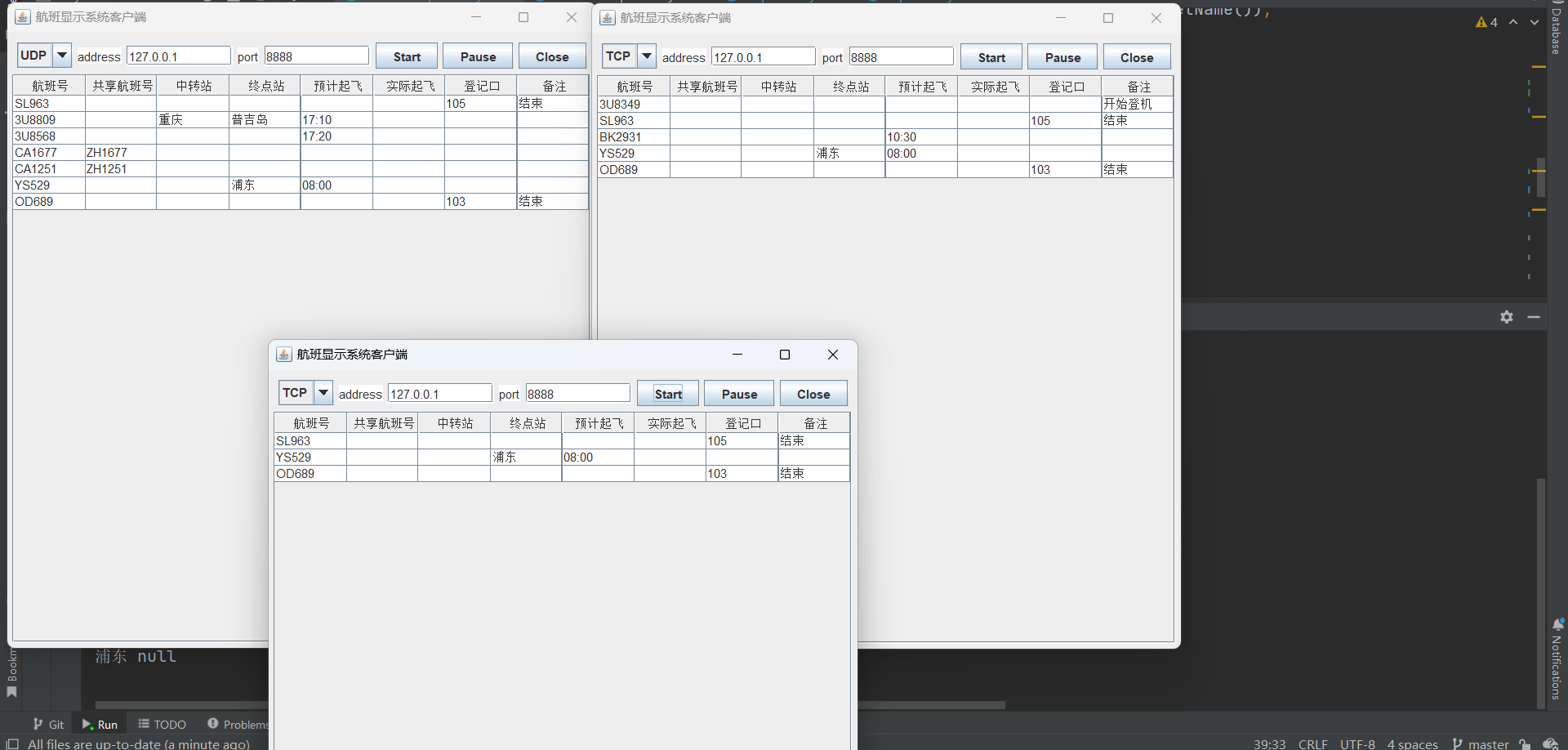
使用java的selector对象处理非阻塞请求,使用while循环分别处理连接请求，写入请求

测试：

同时开启TCP和UDP服务器



开启两个TCP客户端和一个UDP客户端



成功

* + 1. 服务器对运行过程中的各种异常情况的处理与测试

TCP：

1. try {
2. key.channel().close();
3. } catch (IOException e1) {
4. e1.printStackTrace();
5. }
6. key.cancel();
7. *// 客户端主动断开*
8. if (e.getMessage().contains("你的主机中的软件中止了一个已建立的连接。")) {
9. System.out.println("客户端已主动断开连接\n");
10. }

当客户端主动断开连接的时候处理异常

* 1. **基于TCP的航班显示系统客户端设计与实现**
     1. 与服务器建立连接

1. Socket TCP\_Socker = new Socket(TCP\_Address, Integer.parseInt(TCP\_Port));
2. System.out.println("TCP" + TCP\_Address + ": " + TCP\_Port);
3. TCP\_Socker.setSoTimeout(5000);

设置Socket对象并设置超时时间

* + 1. 向服务器发送请求数据或者是接收服务器的数据

1. Scanner sc = new Scanner(TCP\_Socker.getInputStream());
2. line = sc.nextLine();
3. JOptionPane.showMessageDialog(null, "连接服务器成功!" + "\n");

使用Scanner对象接受流

* + 1. 分析提取特定航班显示终端需输出的字段数据的算法和关键代码

1. private void analysisEveryLine(String line) {
2. */\*\**
3. \* 每次分析一行数据初始化置空
4. \*/
5. flid = null;  *// 即数据文件中每行数据的flid*
6. flightNumber = null; *// 航班号，如CA1402*
7. sfawOne = null; *// 共享航班公司1*
8. sfnoOne = null; *// 共享航班号1*
9. sfawTwo = null; *// 共享航班公司2*
10. sfnoTwo = null; *// 共享航班号2*
11. sfawThree = null; *// 共享航班公司3*
12. sfnoThree = null; *// 共享航班号3*
13. aimStation = null; *// 目的站*
14. transferStation = null; *// 中转站*
15. destinationStation = null;
16. planToFlyTime = null; *// 预计起飞时间*
17. changeToFlyTime = null; *// 变更起飞时间*
18. actualFlyTime = null;*//实际起飞时间*
19. gate = null; *// 登机口*
20. state = null; *// 登机状态*
21. */\*\**
22. \* 求航班唯一标识(fild)和航班号flightNumber
23. \* 航向分析以判断是否为出港航班
24. \*/
25. *//匹配[xxxxxx]*
26. Matcher matcher = Pattern.compile("\\[.\*\\]").matcher(line);
27. if (matcher.find()) {
28. String[] flight = matcher.group().split("=");
29. flid = flight[1].substring(0, flight[1].lastIndexOf(','));
30. String ffid = flight[2];
31. flightNumber = Stream.of(ffid.split("-")).limit(2).collect(Collectors.joining());
32. *//如果是进港航班，直接返回*
33. if ("A".equals(ffid.substring(ffid.lastIndexOf(',') - 1, ffid.lastIndexOf(',')))) {
34. return;
35. }
36. }
37. */\*\**
38. \* 求共享航班号
39. \*/
40. if (line.contains("SFLG")) {
41. matcher = Pattern.compile("SFLT \\[.\*\\]").matcher(line);
42. if (matcher.find()) {
43. String shareNo = matcher.group();
44. shareNo = shareNo.substring(6, shareNo.length() - 4);
45. String[] shareNos = shareNo.split("\\], SFLT \\[");
46. if (shareNos.length == 1) {
47. sfawOne = shareNos[0].substring(5, 7);
48. sfnoOne = shareNos[0].substring(14, 18);
49. } else if (shareNos.length == 2) {
50. sfawOne = shareNos[0].substring(5, 7);
51. sfnoOne = shareNos[0].substring(14, 18);
52. sfawTwo = shareNos[1].substring(5, 7);
53. sfnoTwo = shareNos[1].substring(14, 18);
54. } else if (shareNos.length == 3) {
55. sfawOne = shareNos[0].substring(5, 7);
56. sfnoOne = shareNos[0].substring(14, 18);
57. sfawTwo = shareNos[1].substring(5, 7);
58. sfnoTwo = shareNos[1].substring(14, 18);
59. sfawThree = shareNos[2].substring(5, 7);
60. sfnoThree = shareNos[2].substring(14, 18);
61. }
62. System.out.println("共享航班号:" + sfawOne + sfnoOne);
63. }
64. }
65. */\*\**
66. \* 求起始地与中转地
67. \*/
68. if (line.contains("apcd")) {
69. matcher = Pattern.compile("ARPT\\[.\*\\]").matcher(line);
70. if (matcher.find()) {
71. String stationTemp = matcher.group();
72. stationTemp = stationTemp.substring(5, stationTemp.length() - 4);
73. String[] stationTempStrings = stationTemp.split("\\], ARPT\\[");
74. *//含有目的站和终点站*
75. if (stationTempStrings.length == 2) {
76. String destination = stationTempStrings[1].substring(13, 16);
77. destinationStation = airports.getProperty(destination);
78. }
79. *//含有目的站中转站和终点站*
80. if (stationTempStrings.length == 3) {
81. String transfer = stationTempStrings[1].substring(13, 16);
82. transferStation = airports.getProperty(transfer);
83. String destination = stationTempStrings[2].substring(13, 16);
84. destinationStation = airports.getProperty(destination);
85. }
86. }
87. System.out.println(destinationStation + " " + transferStation);
88. }
89. */\*\**
90. \* 求登机门
91. \*/
92. if (line.contains("GATE")) {
93. matcher = Pattern.compile("GATE\\[.\*?\\]").matcher(line);
94. if (matcher.find()) {
95. String gatee = matcher.group();
96. gatee = gatee.substring(5, gatee.length() - 1);
97. String[] strings = gatee.split(", ");
98. gate = strings[2].substring(5);
99. }
100. }
101. */\*\**
102. \* 求起飞落地时间
103. \* <FPTT> 计划起飞时间
104. \* <FETT> 预计起飞时间
105. \* <FRTT>   实际起飞时间
106. \* 后续将简化
107. \*/
108. if (line.contains("fptt") || line.contains("fett")) {*//一个航班同时fett和fptt，就用fett*
109. matcher = Pattern.compile("ARPT\\[.\*?\\]").matcher(line);
110. if (matcher.find()) {
111. String timeFly = matcher.group();
112. String[] strings = timeFly.split(", ");
113. *//选择planToFlyTime*
114. String planFly = strings[2].substring(5);
115. *//20180923080000*
116. if (!"null".equals(planFly)) {
117. StringBuilder builder1 = new StringBuilder(planFly.substring(8, planFly.length() - 2));
118. builder1.insert(2, ":");
119. planToFlyTime = builder1.toString();
120. }
121. *//选择*
122. String realFly = strings[4].substring(5);
123. if (!"null".equals(realFly)) {
124. StringBuilder builder2 = new StringBuilder(realFly.substring(8, planFly.length() - 2));
125. builder2.insert(2, ":");
126. actualFlyTime = builder2.toString();
127. System.out.println("实际" + actualFlyTime);
128. }
129. }
130. }
131. if (!line.contains("fptt") && line.contains("fett")) {*//一个航班只有fett就用fett*
132. matcher = Pattern.compile("\\[.\*\\]").matcher(line);
133. if (matcher.find()) {
134. *//                    System.out.println(matcher5.group());*
135. *//                    [flid=1418075, ffid=GS-7892-20180925-A, fett=null, felt=20180925012500]*
136. Pattern pattern2 = Pattern.compile("(?<=fett=).\*?(?=,)");
137. Matcher matcher6 = pattern2.matcher(line);
138. if (matcher6.find()) {
139. String predictToFly = matcher6.group();
140. if (!"null".equals(predictToFly)) {
141. StringBuilder builder3 = new StringBuilder(predictToFly.substring(8, predictToFly.length() - 2));
142. builder3.insert(2, ":");
143. *//临时更换成actualFlyTime*
144. actualFlyTime = builder3.toString();
145. }
146. }
147. }
148. }
149. */\*\**
150. \* 求航班的状态
151. \*/
152. if (line.contains("BORE"))
153. state = "开始登机";
154. if (line.contains("POKE"))
155. state = "结束";
156. if (line.contains("DLYE"))
157. state = "航班延误";
158. if (line.contains("CANE"))
159. state = "航班取消";
160. */\*\**
161. \* 归并分析结果
162. \*/
163. synchronized (outputData) {
164. if (outputData.get(flid) == null) {
165. outputData.put(flid, new String[]{flightNumber, sfawOne, sfnoOne, sfawTwo, sfnoTwo, sfawThree, sfnoThree,
166. transferStation, destinationStation, planToFlyTime, actualFlyTime, gate, state});
167. } else {
168. String[] values = outputData.get(flid);
169. String[] items = {flightNumber, sfawOne, sfnoOne, sfawTwo, sfnoTwo, sfawThree, sfnoThree,
170. transferStation, destinationStation, planToFlyTime, actualFlyTime, gate, state};
171. for (int i = 0; i < 13; i++) {
172. values[i] = items[i] != null ? items[i] : values[i];
173. }
174. outputData.put(flid, values);
175. outputData.notify();
176. }
177. }
178. }
179. }

同实验二的设计思路，加以更改以使用

* + 1. 实现航班数据的接收、处理、显示并发的多线程设计

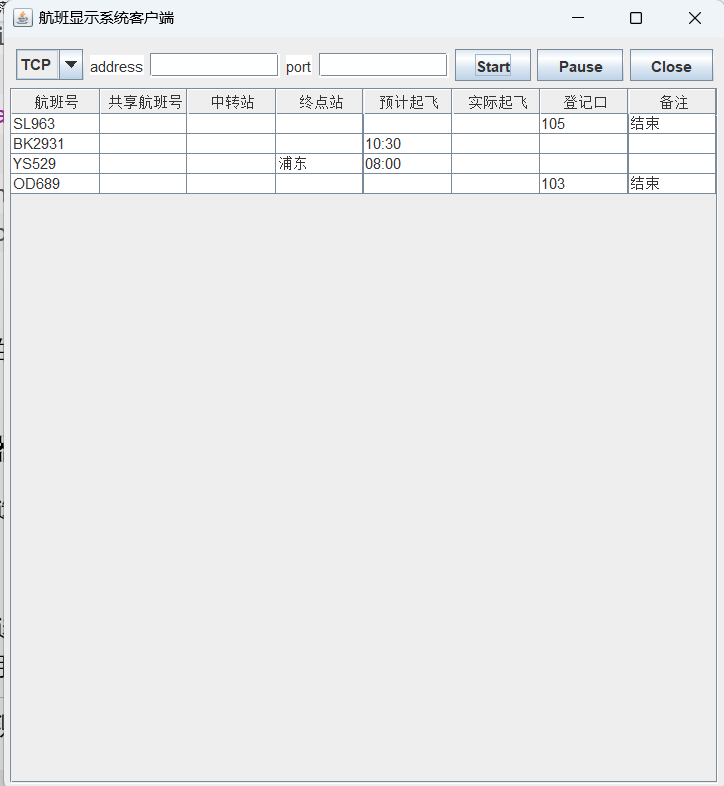
1. readDataThread.start();
2. analysisDataThread.start();
3. publish();

开启两个线程加上一个线程（在publish）方法上。

分别对应读入数据线程，数据分析线程，数据输出线程

* + 1. 航显数据在图形界面上显示输出的代码，并与实际的机场航显终端输出规范进行比较

1. Iterator<String> iterator = outputData.keySet().iterator();
2. int count = 0;
3. while (iterator.hasNext()) {
4. String fild = iterator.next();
5. String[] values = outputData.get(fild);
6. *//共享航班陈对出现，如果为空航班号也为空*
7. for (int n = 1; n <= 6; n += 2) {
8. if (values[n] == null) {
9. values[n] = " ";
10. values[n + 1] = " ";
11. }
12. }
13. */\*\**
14. \* 需要优化来控制格式相同
15. \*/
16. values[0] = String.format("%-10s", values[0]);
17. tm.addRow(new String[]{values[0], values[1] + values[2], values[7], values[8], values[9], values[10], values[11], values[12]});
18. if ((++count) % 10 == 0) {
19. System.out.println("---------------------------------------------------------------------------------");



测试成功！

* + 1. 与服务器断开连接的处理

1. catch (SocketTimeoutException socketTimeoutException) {
2. JOptionPane.showMessageDialog(null, "连接超时");
3. } catch (IOException e1) {
4. // TODO Auto-generated catch block
5. e1.printStackTrace();
6. } catch (InterruptedException e) {
7. throw new RuntimeException(e);
8. }

使用try catch处理超时异常

* + 1. 客户端运行过程中各种异常情况的处理

1. */\*\**
2. \* 清空数据
3. \*/
4. System.out.println("此时应该清空数据");
5. try {
6. Thread.sleep(1000);
7. tm.getDataVector().removeAllElements();
8. } catch (InterruptedException e) {
9. e.printStackTrace();
10. }

线程休眠的异常处理

* 1. **基于UDP的航班显示系统服务器和客户端的设计与实现**
     1. 服务器的启动，包括服务端口的选择和绑定

1. public static void setUdpServer(String UDP\_Address, String UDP\_Port) {
2. *//处理端口号*
3. int port = Integer.parseInt(UDP\_Port);
4. udpServer = new UdpServer(8192, UDP\_Address, port);
5. }
6. *//创建一个未来任务类用于udp使用*
7. FutureTask udpServerThread = new FutureTask(new Callable() {
8. @Override
9. public Object call() throws Exception {
10. udpServer.run();
11. return "启动成功";
12. }
13. });
14. *//创建线程对象*
15. Thread UDP\_thread = new Thread(udpServerThread);

向下初始化线程，并传入参数配置。

* + 1. 管理不同网络客户端的信息

1. Server.UDP\_tm.addRow(new String[]{String.valueOf(connectingCount++), incoming.getAddress().toString(), String.valueOf(incoming.getPort()),"normal"});
   1. Server.UDP\_tm.addRow(new String[]{String.valueOf(connectingCount++), incoming.getAddress().toString(), String.valueOf(incoming.getPort()),"finished!"});

添加当前服务和客户端的信息到滚动面板上

* + 1. 接收客户端的数据或向客户端发送数据

1. *//给客户端发送航班数据*
2. int count = 0;
3. while (count < 10000) {
4. try {
5. while ((lineInfo = bufferedReader.readLine()) != null) {
6. if (UDP\_currentThreadIsShutDown){
7. System.out.println("停止线程!");
8. lineData = "shutDown!".getBytes();
9. incoming.setData(lineData);
10. incoming.setLength(lineData.length);
11. socket.send(incoming);
12. return;
13. }
14. lineData = lineInfo.getBytes();
15. incoming.setData(lineData);
16. incoming.setLength(lineData.length);
17. socket.send(incoming);
18. System.out.println("发送客户端:" + incoming.getSocketAddress() + "第" + count + "
19. System.out.println("数据如下" + lineInfo);
20. count++;
21. Thread.sleep(100);
22. }
23. } catch (IOException e) {
24. throw new RuntimeException(e);
25. } catch (InterruptedException e) {
26. throw new RuntimeException(e);
27. }
28. }

服务器向客户端发送数据

* + 1. 客户端向服务器发送数据和接收服务器的数据

1. try (
2. DatagramSocket socket = new DatagramSocket(0);
3. ) {
4. InetSocketAddress server = new InetSocketAddress(UDP\_Address, Integer.parseInt(UDP\_Port));
5. *//发完请求包*
6. DatagramPacket requestData = new DatagramPacket(new byte[1], 1, server);
7. socket.send(requestData);
8. *//设置超时时间(10s)*
9. socket.setSoTimeout(10000);
10. byte[] buffer = new byte[8 \* 1024];
11. DatagramPacket responseData = new DatagramPacket(buffer, buffer.length);
12. socket.receive(responseData);
13. *//转换为字符串*
14. totalLine = new String(responseData.getData(), responseData.getOffset(), responseData.getLength());
15. System.out.println(totalLine);
16. tm.setRowCount(0);
17. *//弹窗显示信息*
18. JOptionPane.showMessageDialog(null, "连接服务器成功\r\n" + totalLine);

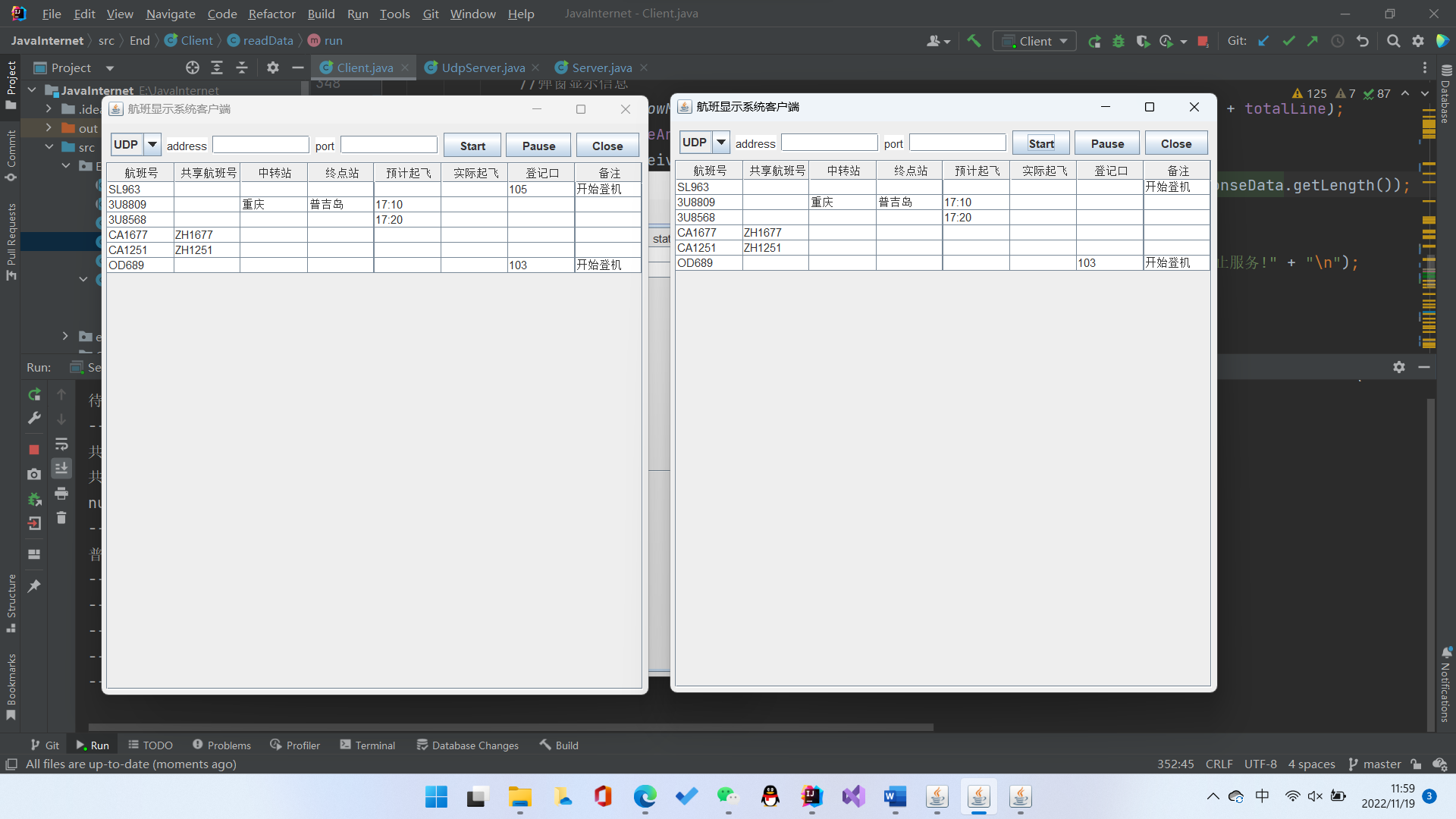
客户端接受服务器的数据

* + 1. 服务器的并发服务功能的设计与测试

1. @Override
2. protected void respond(DatagramSocket socket, DatagramPacket incoming) {
3. *//向线程池添加线程*
4. UDP\_theadPool.execute(new FDSDataHandler(socket, incoming));
5. }
6. *//提供指定最大线程数量的线程池*
7. public static ExecutorService UDP\_theadPool = Executors.newFixedThreadPool(10);

为每个连接开启线程，同时设置线程池，最大为10

测试：



默认为本地+9999端口，所以此处未输入端口

如果配置则也能成功！

* 1. **GUI界面航班显示服务器的管理程序设计与实现**
     1. GUI界面航班显示服务器的管理程序设计与实现

1. */\*\**
2. \* GUI
3. \*/
4. private JPanel contentPane;
5. private JTextField TCP\_addressText
6. private JTextField TCP\_portText;
7. private JTextField UDP\_addressText
8. private JTextField UDP\_portText;
9. private JTable TCP\_table;
10. static DefaultTableModel TCP\_tm;
11. private JTable UDP\_table;
12. static DefaultTableModel UDP\_tm;
13. setTitle("Server");
14. setResizable(false);
15. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
16. setBounds(100, 100, 638, 541);
17. contentPane = new JPanel();
18. contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
19. setContentPane(contentPane);
20. contentPane.setLayout(new BorderLayout(0, 0));
21. JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
22. contentPane.add(tabbedPane, BorderLayout.CENTER);

图形用户界面, 应用程序

描述已自动生成

设计如上

* + 1. GUI航班显示客户端的管理程序设计与实现

1. */\*\**
2. \* GUI
3. \*/
4. private JPanel contentPane;
5. private JTable table;
6. private DefaultTableModel tm;
7. private int count = 0;
8. private ReceiveAndProcessData receiveAndProcessData;
9. *//地址*
10. private JTextField addressText;
11. *//端口*
12. private JTextField portText;
13. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
14. setBounds(100, 100, 591, 640);
15. setTitle("航班显示系统客户端");
16. contentPane = new JPanel();
17. contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
18. contentPane.setLayout(new BorderLayout(0, 0));
19. setContentPane(contentPane);
20. JPanel panel = new JPanel();
21. contentPane.add(panel, BorderLayout.NORTH);
22. JComboBox comboBox = new JComboBox();
23. comboBox.addItem("TCP");
24. comboBox.addItem("UDP");
25. panel.add(comboBox);
26. JTextArea txtAddress = new JTextArea();
27. txtAddress.setText("address");
28. txtAddress.setEditable(false);
29. panel.add(txtAddress);
30. addressText = new JTextField();
31. panel.add(addressText);
32. addressText.setColumns(10);
33. JTextArea txtPort = new JTextArea();
34. txtPort.setText("port");
35. txtPort.setEditable(false);
36. panel.add(txtPort);
37. portText = new JTextField();
38. panel.add(portText);
39. portText.setColumns(10);

图形用户界面, 应用程序, Word

描述已自动生成

## 项目分析总结

此项目完成了大作业要求的90%的内容，完美完成了客户端航班显示系统的刷新而不是新增加数据，能够自动的清理过时的数据而不是输出新的数据。客户端可以同时开启TCP和UDP服务并且能够显示当前服务器和连接的客户端的状态。在停止服务器的后，会向所有客户端发送停止信号。而不是直接关闭。

在实验的过程中，对java的非阻塞TCP连接不够熟练，需要查看以往的教师视频和将以复习java的非阻塞通信。加上互联网搜索引擎与B站网课加以完成此项目。

## 总结

本项目总结了之前的五次实验，将五次实验汇合到一起，同时增加了GUI的功能，能够直观地开启和查看服务器。同时优化了实验二未完成的客户端的功能-实现航班信息的滚动刷新。

代码开源如下：

[WenTesla/javaInternet (github.com)](https://github.com/WenTesla/javaInternet)

如果有错误或bug直接在github提交issues即可。

## 参考文献

https://blog.csdn.net/anlian523/article/details/105020210

《java网络编程》