**北京邮电大学软件学院**

**2020-2021学年第一学期实验报告**

**课程名称： 通信协议软件设计**

**实验名称： 协议实现**

**项目完成人：**

**指导教师：\_\_\_\_\_\_雷友珣\_\_\_\_\_\_\_\_**

**日 期： 2020年 12月 14 日**

1. **实验目的**

通过本实验使学生理解协议工程基本过程，使学生在对协议的分析、建模、验证的基础上，具备协议软件实现能力。

1. **实验任务**

在对通信协议分析、建模和验证实验的基础上，对选择待实现的协议确定协议软件的运行环境、对外接口，选择编程语言、编码实现该协议软件，并对所实现的协议软件进行测试。

1. **实验内容**
2. 确定所需实现的协议软件的运行环境，确定协议软件与运行环境的接口。
3. 选定编程语言。
4. 对协议软件进行设计。
5. 对协议软件进行编码实现。
6. 设计协议软件测试环境，并编码实现用于模拟运行环境的测试用软件。
7. 对协议软件进行测试。
8. **实验环境**
9. Windows系统主机
10. 高级语言开发环境：Java语言开发环境，JDK1.8，IDEA集成开发环境。
11. **实验结果**
12. 实验总体描述：

本次实验采用java语言实现回退N协议，协议在操作系统中以进程/线程的方式运行，对外接口采用**javafx图形界面**的方式实现，用户可以在图形界面的文本框中输入数据，然后点击发送按钮，来模拟网络层向数据链路层发送数据包，之后数据链路层在向物理层发送数据。

其中客户端的图形界面可以表示为网络层、其本身可以表示为数据链路层、服务端本身可以表示为物理层、其图形界面向用户展示接收的结果。

用户可以通过客户端的图形界面向服务端发送数据，在发送数据时会将数据封装成帧，通过IO流发向服务端，之后客户端界面上会显示帧和数据链路层的信息例如ack、seq、缓冲区大小等。服务端接受到帧后，也会将真的信息输出到界面上，以供用户查看。帧的传输方式采用java.net.socket的方式实现，数据链路层（客户端）与物理层（服务端）建立连接与数据流。

用户可以通过服务端的图形界面向客户端发送数据帧，用户选择帧的类型data||ack，然后输入帧的信息，点击发送按钮后，向客户端发送信息，并且在图形界面上输出帧的信息，当客户端接受到帧后，首先会输出帧的信息，然后根据这些信息进行相应操作，例如向识别ack释放buffer、取消某一帧的重传计时器等。

1. 软件运行环境分析

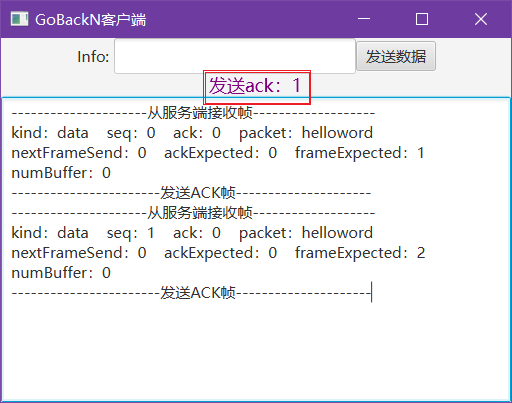
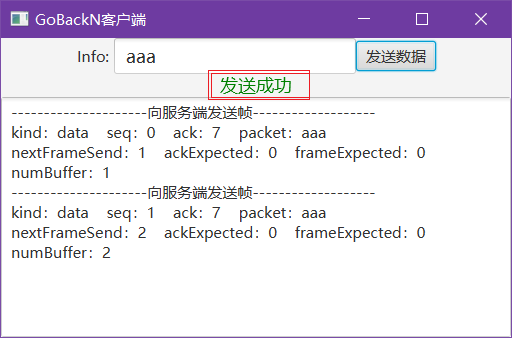
运行环境为windows10 系统，协议软件采用java语言编写，使用了抽象的物理层和网络层来和我们定义的 gobackN 方法(作为数据链路层)来进行交互，其中数据的输出输出均在图形界面上展示。软件协议的内部实现逻辑将以函数调用和创建多线程的方式运行。

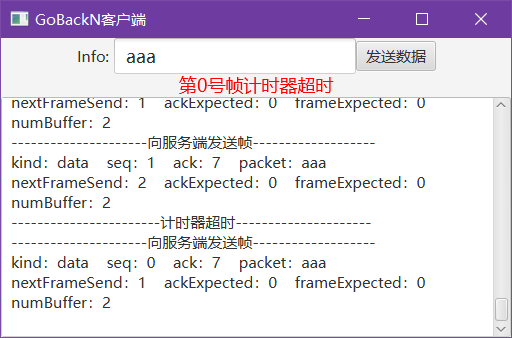
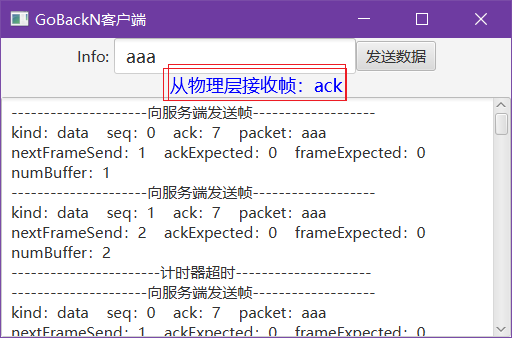
1. 对外接口分析与设计

本协议软件采用图形界面的方式实现对外借口，用户通过在图形界面上输入信息，来实现对数据的传递。

***【客户端界面讲解】***

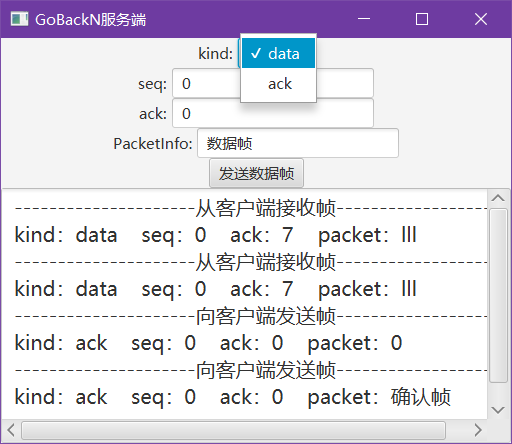
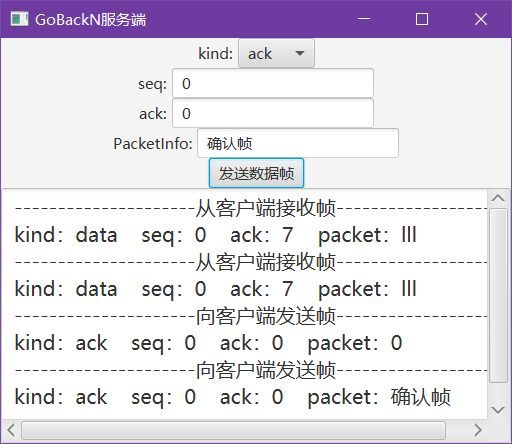
用户向该界面的输入框输入自己想要向服务端发送的数据，然后点击发送数据按钮，客户端程序就会读取用户输入的数据，然后封装成帧，将数据发送到服务端，在发送成功后，客户端界面的文本域上会显示帧和客户端（数据链路层）的信息，然后用绿色的字体提示用户发送成功。如果客户端从服务端接受到数据帧，则显示蓝色提示，并也会输出帧的信息和客户端信息，当发生超时时间时会显示红色提示信息。发送ack帧会显示紫色提示信息，具体实例如下图所示：





***【服务端界面讲解】***

用户通过该界面的选择框和输入框输行程一个帧的数据，点击发送按钮后，服务端程序就会将这些帧信息封装成帧，然后通过数据流发送到客户端，同时在文本域中显示出发送的帧信息。具体实例如下图所示：



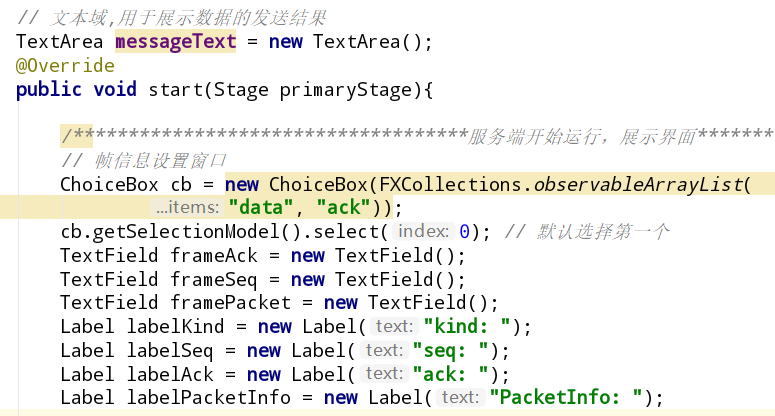
***对外接口代码简介：***

【客户端】



在界面窗口中定义infoTip文本，当通过客户端某一动作完成后，改变infoTip中的值，在界面上提示用户完成结果。定义messageText文本域，用于显示客户端程序输入输出信息。定义packetData文本框，用于用户输入数据，定义sendButton按钮，当用户点击该按钮时，客户端程序读取packetData中的数据，然后将其封装成帧发送到服务端。

【服务端】



在界面窗口中定义messageText文本域，用于显示服务端程序输入输出信息。定义一个cb选择框，包含“data”和“ack“两个选项，用于用户对发送帧的类型进行选择，定义三个文本框frameAck、framSeq、framePacket分别用于输入帧的ack、seq、以及携带的包数据。

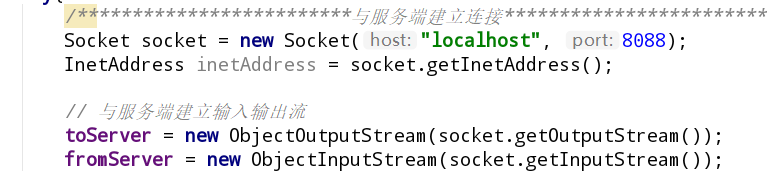
1. 协议软件设计

***协议软件整体设计实现思路：***

1. 首先使用图形界面实现协议的整体控制，用于数据的输入输出，方便数据链路层的交互以及观察GoBackN协议的实现结果。
2. 使用java Socket实现客户端与服务端之间的通信，创建对象IO流，用于传输Frame对象。
3. 对Frame类进行设计，思考需要包含那些属性。
4. 客户端实现用户点击按钮后发送数据帧的功能，通过对象IO流的方式传输，在实现时设置客户端帧缓冲区大小，
5. 设计服务端，实现向客户端发送数据和接收数据功能。
6. 在客户端和服务端能够成功发送或接收数据帧的基础上，在客户端设计超时重传功能，定义一个计时器数组，记录每个帧的发送时间，超时则回退重传。
7. 在超时重传完成的基础之上，添加捎带确认帧功能，当客户端要返回一个ack确认帧后，等待一定时间，如果在这段时间内有数据帧要发送，则在数据帧上加上ack信息，执行捎带确认，如果没有数据帧要发送，则单独生成一个确认帧，发送给服务端。

***具体设计代码讲解：***

【客户端】

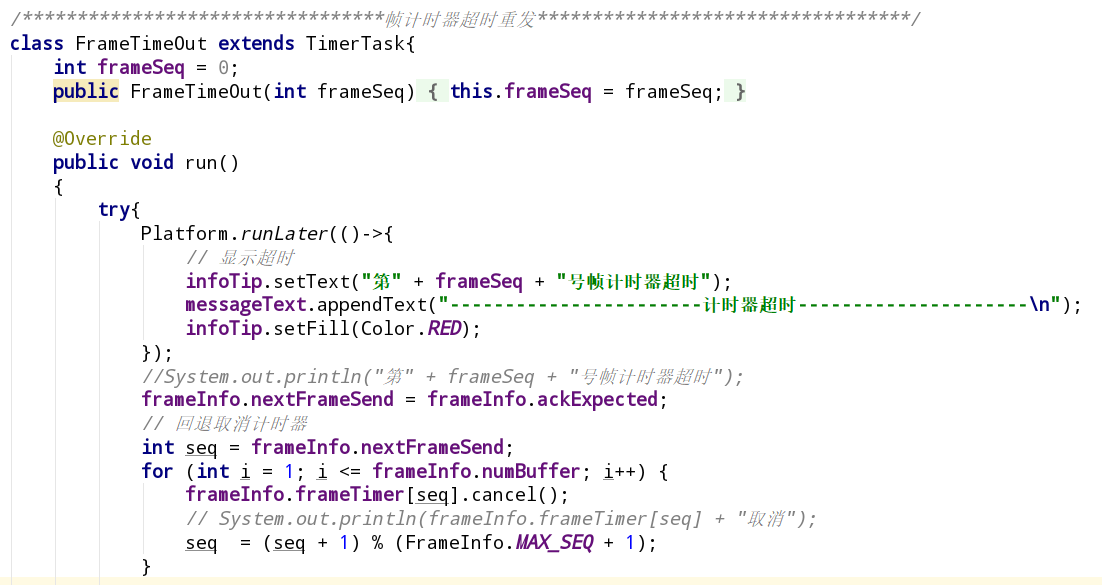


**创建与服务端连接：**创建Socket，建立与服务端的连接

**读取服务端发来的帧：**建立连接后，创建一个线程，用于检测服务端发来的帧，并接收进行如下操作：将帧数据显示到界面中，改变期待帧号，设置捎带确认帧的计时器，查看帧的ack信息，改变ackExpected的值，然后将帧数据在界面中显示。

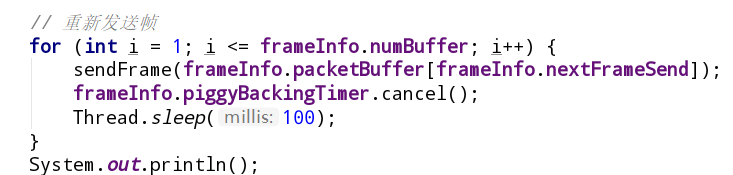


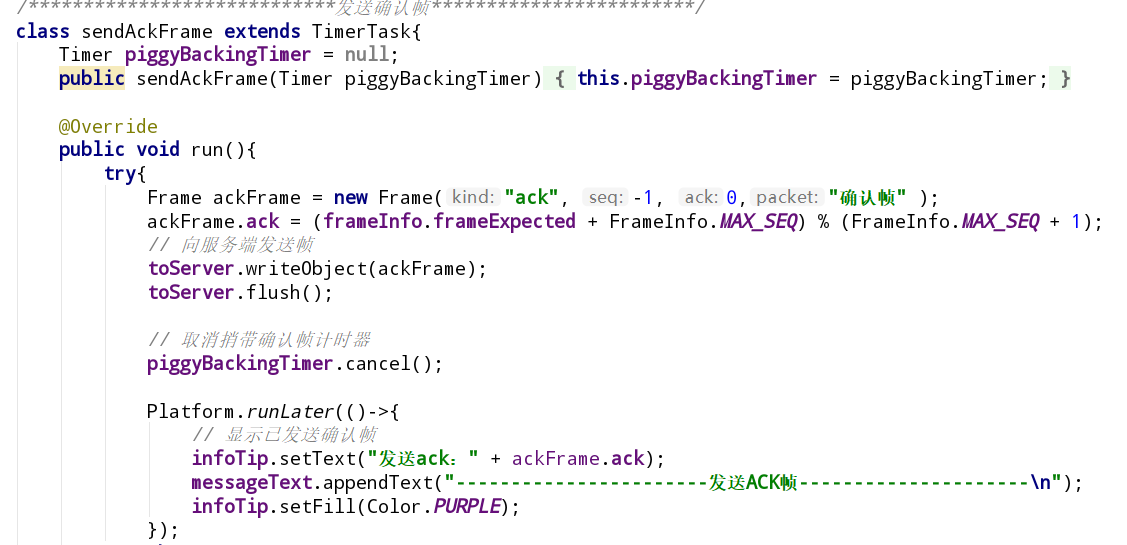
**发送数据帧：**实现点击sendButton后的发送数据帧逻辑，首先从文本框packetData中读取用户输入的数据，存到帧缓冲区中，缓冲区计数自增，之后通过sendFrame函数发送数据帧，sendFrame主要操作为将帧通过对象IO流发送到服务端，对该帧创建一个重传计时器线程，改变下一发送帧的帧号。之后取捎带确认帧的计时器。



**计时器超时重传：**当某一帧计时器超时后，计时器线程执行重传操作，首先向界面输出超时提示，然后循环取消该帧之后所有帧的计时器。

之后循环重新发送数据帧。





**发送确认帧：**当捎带确认帧计时器超时后，客户端需要单独发送一个确认帧到服务端，该计时线程执行sendAckFrame对应的run函数，创建一个frame，设置帧类型为”ack”,设置帧的ack值，然后通过对象IO流发送，最后取消捎带确认帧计时器，在界面上输出结果。

【服务端】

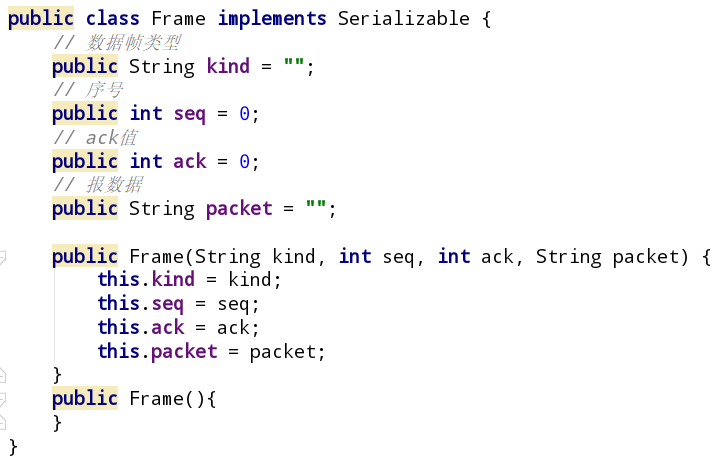


**服务器启动：**生成操作界面后，创建线程使用ServiceSocket，以本机作为服务器启动，然后循环检测从客户端发来的数据帧，然后输出到界面上。

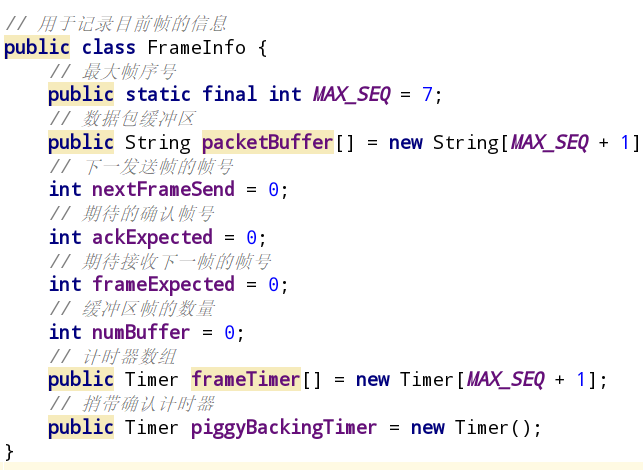


**发送数据帧：**当用户点击客户端的sendButton按钮后，服务端读取文本框中的数据，设置该帧的各项属性值，然后通过流发送出去。

1. 主要数据结构



**Frame类，主要用于对数据包的帧封装。**

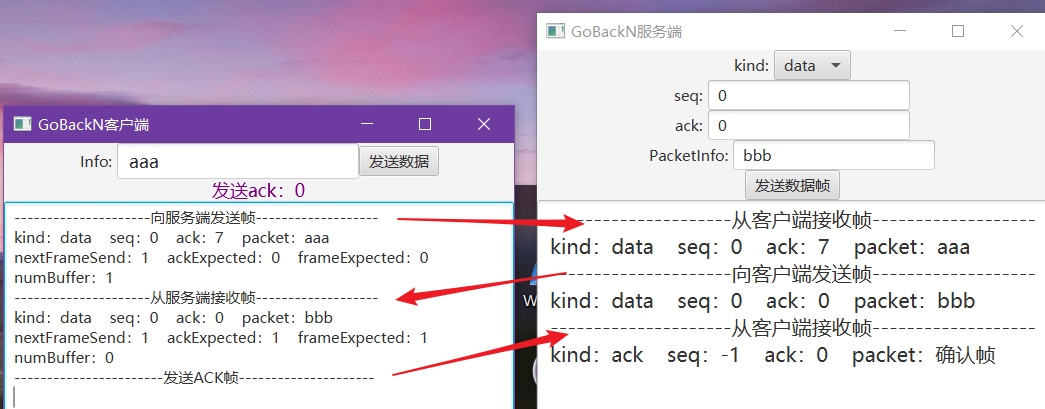


**FrameInfo类，主要用于保存数据链路层的信息。**

1. 协议软件测试

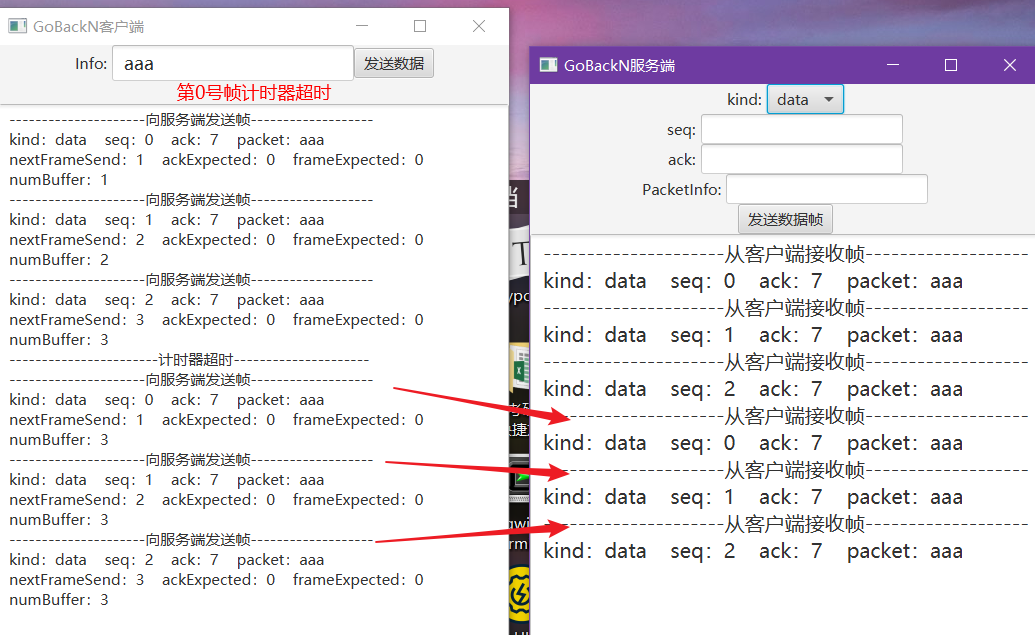
**【客户端与服务端收发帧】**

客户端向服务端发帧，服务端向客户端发送带有ack信息的数据帧，客户端捎带确认计时器超时，单独发送一个ack帧向服务端。

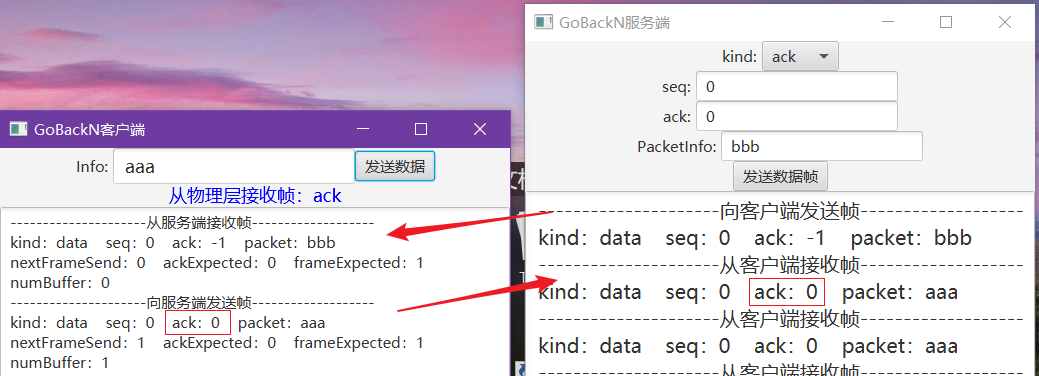


**【超时重传】**

客户端向服务端连续发送三个数据帧，由于没有接受到来自服务端的ack=0的响应，因此seq=0的帧超时，因此重新发送这三个帧。

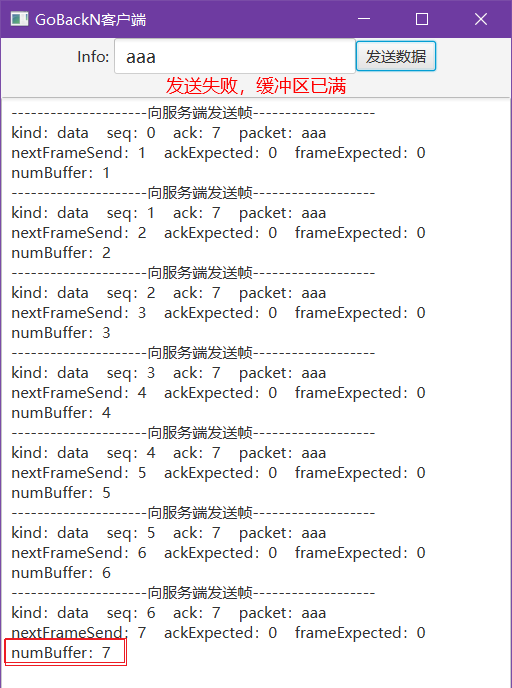


**【捎带确认】**

客户端从服务端接收了一个数据帧，而此时客户端正好要给服务端发送一个数据帧，因此此帧携带ack=0的确认信息，发送到了服务端，服务端接受到数据帧，并获取了确认信息，这避免了客户端额外发送一个ack帧到服务端。

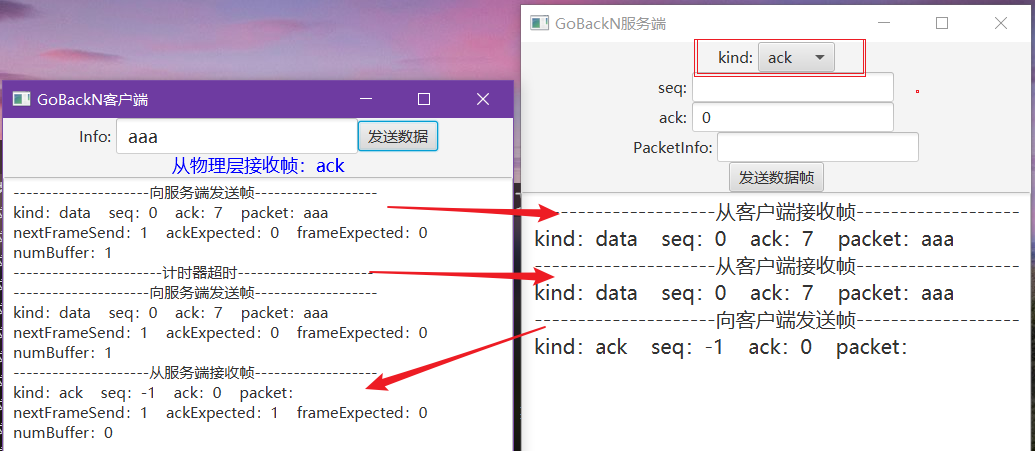
**【客户端缓冲区已满】**

用户连续向客户端发送8个数据，超过了客户端缓冲区大小（7），界面提示缓冲区已满



**【超时重传与确认】**

客户端向服务端发送一个数据帧，但是在计时器超时后，没有接受到服务端的ack信息，因此重传该数据帧，之后收到服务端的ack信息，于是取消该帧的计时器，不在重发。



老师和助教辛苦了

1. **实验源码**
2. **Client类**
3. **import** javafx.application.Application;
4. **import** javafx.application.Platform;
5. **import** javafx.event.EventHandler;
6. **import** javafx.geometry.Pos;
7. **import** javafx.scene.Scene;
8. **import** javafx.scene.control.\*;
9. **import** javafx.scene.layout.HBox;
10. **import** javafx.scene.layout.VBox;
11. **import** javafx.scene.paint.Color;
12. **import** javafx.stage.Stage;
13. **import** javafx.scene.text.Text;
14. **import** javafx.stage.WindowEvent;
16. **import** java.io.ObjectInputStream;
17. **import** java.io.ObjectOutputStream;
18. **import** java.net.InetAddress;
19. **import** java.net.Socket;
20. **import** java.util.Timer;
21. **import** java.util.TimerTask;
23. **public** **class** Client **extends** Application {
24. FrameInfo frameInfo = **new** FrameInfo();
26. // 声明输入输出流
27. ObjectOutputStream toServer = **null**;
28. ObjectInputStream fromServer = **null**;
30. // 发送结果提示
31. Text infoTip = **new** Text();
32. // 文本域,用于展示帧的发送结果
33. TextArea messageText = **new** TextArea();
35. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*客户端开始运行，展示界面\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
36. @Override
37. **public** **void** start(Stage primaryStage){
38. // 输入文本框，用于输入数据
39. TextField packetData = **new** TextField();
40. // 文本框提示标签
41. Label tipLabel = **new** Label("Info: ");
42. // 发送数据
43. Button sendButton = **new** Button();
44. sendButton.setText("发送数据");
45. HBox inputDataHBox =**new** HBox();
46. inputDataHBox.setAlignment(Pos.CENTER);//居中
47. packetData.setStyle("-fx-font-size: 18");
48. inputDataHBox.getChildren().addAll(tipLabel, packetData, sendButton);
50. // 文本域,用于展示帧的发送结果
51. messageText.setPrefColumnCount(30);
52. messageText.setPrefRowCount(80);
54. VBox vBox = **new** VBox();
55. vBox.setAlignment(Pos.CENTER);
56. infoTip.setStyle("-fx-font-size: 18");
57. vBox.getChildren().addAll(inputDataHBox, infoTip, messageText);
59. Scene scene = **new** Scene(vBox, 510, 800);
60. primaryStage.setTitle("GoBackN客户端");
61. primaryStage.setScene(scene);
62. // 展示图形界面
63. primaryStage.show();
65. // 当退出窗口时，程序结束
66. primaryStage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {
67. @Override
68. **public** **void** handle(WindowEvent event) {
69. System.exit(0);
70. }
71. });
73. **try**{
74. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*与服务端建立连接\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
75. Socket socket = **new** Socket("localhost", 8088);
76. InetAddress inetAddress = socket.getInetAddress();
78. // 与服务端建立输入输出流
79. toServer = **new** ObjectOutputStream(socket.getOutputStream());
80. fromServer = **new** ObjectInputStream(socket.getInputStream());
81. }**catch** (Exception e){
82. e.printStackTrace();
83. }
85. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*从服务端（物理层）接收帧 R - receive\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
86. **new** Thread(()->{
87. **try**{
88. **while**(**true**){
89. Frame Rframe = (Frame) fromServer.readObject();
90. **if**(Rframe.kind.equals("data")){
91. **if**(Rframe.seq == frameInfo.frameExpected){
92. // 改变期待帧号
93. frameInfo.frameExpected = (frameInfo.frameExpected + 1) % (FrameInfo.MAX\_SEQ + 1);
94. }
95. // 设定捎带确认帧计时器
96. frameInfo.piggyBackingTimer = **new** Timer();
97. frameInfo.piggyBackingTimer.schedule(**new** sendAckFrame(frameInfo.piggyBackingTimer), 2500);
98. }

101. // 查看ack信息
102. checkAck(Rframe);
103. Platform.runLater(()->{
104. // 显示结果
105. infoTip.setText("从物理层接收帧：" + Rframe.kind);
106. infoTip.setFill(Color.BLUE);
107. // 输出接收帧的信息
108. messageText.appendText("---------------------从服务端接收帧-------------------\n");
109. showResult(messageText, Rframe);
110. });
111. }
112. }**catch** (Exception e){
113. e.printStackTrace();
114. }
115. }).start();

118. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*点击发送数据按钮向（数据链路层）发送数据\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
119. sendButton.setOnAction(e->{
120. **try**{
121. **if**(frameInfo.numBuffer < FrameInfo.MAX\_SEQ){
122. // 存储数据到缓冲区
123. frameInfo.packetBuffer[frameInfo.nextFrameSend] = packetData.getText().trim();
124. frameInfo.numBuffer += 1;
125. // 发送帧
126. sendFrame(packetData.getText().trim());
127. // 取消捎带确认计时器
128. frameInfo.piggyBackingTimer.cancel();
129. }**else** {
130. infoTip.setFill(Color.RED);
131. infoTip.setText("发送失败，缓冲区已满");
132. }
133. }**catch** (Exception dataE){
134. dataE.printStackTrace();
135. System.out.println("传输数据失败");
136. }
137. });
138. }
140. // 展示帧和数据链路层数据
141. **private** **void** showResult(TextArea messageText, Frame frame){
142. messageText.appendText("kind：" + frame.kind + "    ");
143. messageText.appendText("seq：" + frame.seq + "    ");
144. messageText.appendText("ack：" + frame.ack + "    ");
145. messageText.appendText("packet：" + frame.packet + "\n");
146. messageText.appendText("nextFrameSend：" + frameInfo.nextFrameSend + "    ");
147. messageText.appendText("ackExpected：" + frameInfo.ackExpected + "    ");
148. messageText.appendText("frameExpected：" + frameInfo.frameExpected + "\n");
149. messageText.appendText("numBuffer：" + frameInfo.numBuffer + "\n");
151. }
153. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*查看返回的确认帧是否正确\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
154. **private** **void** checkAck(Frame frame){
155. **while**(**true**){
156. **if**((frameInfo.ackExpected <= frame.ack) && (frame.ack < frameInfo.nextFrameSend) ||
157. (frameInfo.nextFrameSend < frameInfo.ackExpected) && (frameInfo.ackExpected <= frame.ack) ||
158. (frame.ack < frameInfo.nextFrameSend) && (frameInfo.nextFrameSend < frameInfo.ackExpected)){
159. // 释放缓冲区
160. frameInfo.numBuffer -= 1;
161. // 取消计时器
162. frameInfo.frameTimer[frameInfo.ackExpected].cancel();
163. // 改变期待帧的帧号
164. frameInfo.ackExpected = (frameInfo.ackExpected + 1) % (FrameInfo.MAX\_SEQ + 1);
165. }**else**{
166. **break**;
167. }
168. }
169. }
171. // 发送数据帧
172. **private** **void** sendFrame(String packetData){
173. **try**{
174. // 创建帧 S - send
175. Frame Sframe = **new** Frame("data", frameInfo.nextFrameSend, 0 , packetData);
176. Sframe.ack = (frameInfo.frameExpected + FrameInfo.MAX\_SEQ) % (FrameInfo.MAX\_SEQ + 1);
178. // 向服务端发送帧
179. toServer.writeObject(Sframe);
180. toServer.flush();
182. // 设置定时器,在5秒延时后重发
183. frameInfo.frameTimer[Sframe.seq] = **new** Timer();
184. frameInfo.frameTimer[Sframe.seq].schedule(**new** FrameTimeOut(Sframe.seq), 5000);
185. System.out.println(frameInfo.frameTimer[Sframe.seq]);
187. // 显示发送结果
188. Platform.runLater(()->{
189. messageText.appendText("---------------------向服务端发送帧-------------------\n");
190. showResult(messageText, Sframe);
191. **if**(!infoTip.getText().trim().contains("号帧计时器超时")){
192. infoTip.setFill(Color.GREEN);
193. infoTip.setText("发送成功");
194. }
195. });
197. // 改变下一发送帧号
198. frameInfo.nextFrameSend = (frameInfo.nextFrameSend + 1) % (FrameInfo.MAX\_SEQ + 1);
199. System.out.println("增加后" + frameInfo.nextFrameSend);

202. }**catch** (Exception e){
203. e.printStackTrace();
204. System.out.println("发送帧失败");
205. }
206. }

209. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*帧计时器超时重发\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
210. **class** FrameTimeOut **extends** TimerTask{
211. **int** frameSeq = 0;
212. **public** FrameTimeOut(**int** frameSeq){
213. **this**.frameSeq = frameSeq;
214. }
216. @Override
217. **public** **void** run()
218. {
219. **try**{
220. Platform.runLater(()->{
221. // 显示超时
222. infoTip.setText("第" + frameSeq + "号帧计时器超时");
223. messageText.appendText("-----------------------计时器超时---------------------\n");
224. infoTip.setFill(Color.RED);
225. });
226. //System.out.println("第" + frameSeq + "号帧计时器超时");
227. frameInfo.nextFrameSend = frameInfo.ackExpected;
228. // 回退取消计时器
229. **int** seq = frameInfo.nextFrameSend;
230. **for** (**int** i = 1; i <= frameInfo.numBuffer; i++) {
231. frameInfo.frameTimer[seq].cancel();
232. // System.out.println(frameInfo.frameTimer[seq] + "取消");
233. seq  = (seq + 1) % (FrameInfo.MAX\_SEQ + 1);
234. }
236. // 重新发送帧
237. **for** (**int** i = 1; i <= frameInfo.numBuffer; i++) {
238. sendFrame(frameInfo.packetBuffer[frameInfo.nextFrameSend]);
239. frameInfo.piggyBackingTimer.cancel();
240. Thread.sleep(100);
241. }
242. System.out.println();
243. }**catch** (Exception e){
244. e.printStackTrace();
245. System.out.println("超时重传发送错误");
246. }
247. }
248. }
250. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*发送确认帧\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
251. **class** sendAckFrame **extends** TimerTask{
252. Timer piggyBackingTimer = **null**;
253. **public** sendAckFrame(Timer piggyBackingTimer){
254. **this**.piggyBackingTimer = piggyBackingTimer;
255. }
257. @Override
258. **public** **void** run(){
259. **try**{
260. Frame ackFrame = **new** Frame("ack", -1, 0,"确认帧" );
261. ackFrame.ack = (frameInfo.frameExpected + FrameInfo.MAX\_SEQ) % (FrameInfo.MAX\_SEQ + 1);
262. // 向服务端发送帧
263. toServer.writeObject(ackFrame);
264. toServer.flush();
266. // 取消捎带确认帧计时器
267. piggyBackingTimer.cancel();
269. Platform.runLater(()->{
270. // 显示已发送确认帧
271. infoTip.setText("发送ack：" + ackFrame.ack);
272. messageText.appendText("-----------------------发送ACK帧---------------------\n");
273. infoTip.setFill(Color.PURPLE);
274. });
275. }**catch** (Exception e){
276. e.printStackTrace();
277. System.out.println("发送确认帧失败");
278. }
279. }
280. }
281. }
282. **Server类**
283. **import** javafx.application.Application;
284. **import** javafx.application.Platform;
285. **import** javafx.collections.FXCollections;
286. **import** javafx.geometry.Pos;
287. **import** javafx.scene.Scene;
288. **import** javafx.scene.control.\*;
289. **import** javafx.scene.layout.HBox;
290. **import** javafx.scene.layout.VBox;
291. **import** javafx.stage.Stage;
292. **import** javafx.stage.WindowEvent;
293. **import** javafx.event.EventHandler;
295. **import** java.io.IOException;
296. **import** java.io.ObjectInputStream;
297. **import** java.io.ObjectOutputStream;
298. **import** java.net.ServerSocket;
299. **import** java.net.Socket;
301. **public** **class** Server **extends** Application {
303. // 声明输入输出数据流
304. ObjectInputStream fromClient = **null**;
305. ObjectOutputStream toClient = **null**;
307. // 文本域,用于展示数据的发送结果
308. TextArea messageText = **new** TextArea();
309. @Override
310. **public** **void** start(Stage primaryStage){
312. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*服务端开始运行，展示界面\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
313. // 帧信息设置窗口
314. ChoiceBox cb = **new** ChoiceBox(FXCollections.observableArrayList(
315. "data", "ack"));
316. cb.getSelectionModel().select(0); // 默认选择第一个
317. TextField frameAck = **new** TextField();
318. TextField frameSeq = **new** TextField();
319. TextField framePacket = **new** TextField();
320. Label labelKind = **new** Label("kind: ");
321. Label labelSeq = **new** Label("seq: ");
322. Label labelAck = **new** Label("ack: ");
323. Label labelPacketInfo = **new** Label("PacketInfo: ");
325. HBox hbox1 = **new** HBox();
326. hbox1.setAlignment(Pos.CENTER);
327. hbox1.getChildren().addAll(labelKind, cb);
329. HBox hbox2 = **new** HBox();
330. hbox2.setAlignment(Pos.CENTER);
331. hbox2.getChildren().addAll(labelSeq, frameSeq );
333. HBox hbox3 = **new** HBox();
334. hbox3.setAlignment(Pos.CENTER);
335. hbox3.getChildren().addAll(labelAck, frameAck);
337. HBox hbox4 = **new** HBox();
338. hbox4.setAlignment(Pos.CENTER);
339. hbox4.getChildren().addAll(labelPacketInfo, framePacket);
341. // 文本域,用于展示帧的发送结果
342. messageText.setStyle("-fx-font-size: 20");
343. messageText.setPrefColumnCount(30);
344. messageText.setPrefRowCount(80);
346. // 发送按钮
347. Button sendButton = **new** Button();
348. sendButton.setText("发送数据帧");
350. VBox vBox = **new** VBox();
351. vBox.setAlignment(Pos.CENTER);
352. vBox.getChildren().addAll(hbox1, hbox2, hbox3, hbox4, sendButton, messageText);
354. Scene scene = **new** Scene(vBox, 510, 800);
355. primaryStage.setTitle("GoBackN服务端");
356. primaryStage.setScene(scene);
357. // 展示图形界面
358. primaryStage.show();
360. // 当退出窗口时，程序结束
361. primaryStage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {
362. @Override
363. **public** **void** handle(WindowEvent event) {
364. System.exit(0);
365. }
366. });
368. **new** Thread(()->{
369. **try**{
370. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*以本机作为服务器提供服务，端口为8088\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
371. ServerSocket serverSocket = **new** ServerSocket(8088);
372. System.out.println("服务器启动:\n");
373. // 收听客户端的连接请求
374. Socket socket = serverSocket.accept();
376. // 与客户端建立输入输出流
377. fromClient = **new** ObjectInputStream(socket.getInputStream());
378. toClient = **new** ObjectOutputStream(socket.getOutputStream());
379. System.out.println("客户端已连接");
381. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*检测帧客户端(数据链路层）是否发送帧\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
382. **while**(**true**)
383. {
384. Frame frame = (Frame) fromClient.readObject();
385. Platform.runLater(() -> {
386. messageText.appendText("---------------------从客户端接收帧-------------------\n");
387. showResult(messageText, frame);
388. });
389. }
390. }**catch** (ClassNotFoundException e){
391. e.printStackTrace();
392. System.out.println("接收数据出错！！");
393. }**catch** (IOException e){
394. System.out.println("客户端断开连接");
395. fromClient = **null**;
396. toClient = **null**;
398. }
399. }).start();
401. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*点击发送数据按钮向（数据链路层）发送数据\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
402. sendButton.setOnAction(e ->{
403. **try**{
404. Frame frame = **new** Frame();
405. // 读取帧的数据类型
406. frame.kind = cb.getSelectionModel().getSelectedItem().toString();
407. // 读取帧序号
408. **if** (frameSeq.getText().trim().isEmpty()){
409. frame.seq = -1;
410. }**else** {
411. frame.seq = Integer.parseInt(frameSeq.getText().trim());
412. }
413. // 读取确认帧号
414. **if** (frameAck.getText().trim().isEmpty()){
415. frame.ack = -1;
416. }**else** {
417. frame.ack = Integer.parseInt(frameAck.getText().trim());
418. }
419. // 获取帧的数据
420. **if** (framePacket.getText().trim().isEmpty()){
421. frame.packet = "";
422. }**else** {
423. frame.packet = framePacket.getText().trim();
424. }
426. // 显示发送的帧信息
427. messageText.appendText("---------------------向客户端发送帧-------------------\n");
428. showResult(messageText, frame);
430. // 向客户端发送帧
431. toClient.writeObject(frame);
432. toClient.flush();
433. }**catch** (Exception e1){
434. e1.printStackTrace();
435. messageText.appendText("------数据填写错误，发送失败-------\n");
436. }
437. });
438. }
440. **private** **void** showResult(TextArea messageText, Frame frame){
441. messageText.appendText("kind：" + frame.kind + "    ");
442. messageText.appendText("seq：" + frame.seq + "    ");
443. messageText.appendText("ack：" + frame.ack + "    ");
444. messageText.appendText("packet：" + frame.packet + "\n");
445. }
446. }
447. **Frame类**
448. **import** java.io.Serializable;
450. **public** **class** Frame **implements** Serializable {
451. // 数据帧类型
452. **public** String kind = "";
453. // 序号
454. **public** **int** seq = 0;
455. // ack值
456. **public** **int** ack = 0;
457. // 报数据
458. **public** String packet = "";
460. **public** Frame(String kind, **int** seq, **int** ack, String packet) {
461. **this**.kind = kind;
462. **this**.seq = seq;
463. **this**.ack = ack;
464. **this**.packet = packet;
465. }
466. **public** Frame(){
467. }
468. }
469. **FrameInfo类**
470. **import** java.util.Timer;
472. // 用于记录目前帧的信息
473. **public** **class** FrameInfo {
474. // 最大帧序号
475. **public** **static** **final** **int** MAX\_SEQ = 7;
476. // 数据包缓冲区
477. **public** String packetBuffer[] = **new** String[MAX\_SEQ + 1];
478. // 下一发送帧的帧号
479. **int** nextFrameSend = 0;
480. // 期待的确认帧号
481. **int** ackExpected = 0;
482. // 期待接收下一帧的帧号
483. **int** frameExpected = 0;
484. // 缓冲区帧的数量
485. **int** numBuffer = 0;
486. // 计时器数组
487. **public** Timer frameTimer[] = **new** Timer[MAX\_SEQ + 1];
488. // 捎带确认计时器
489. **public** Timer piggyBackingTimer = **new** Timer();
490. }