**重庆大学Project报告**

题 目： 基于JAVA Socket的简单网络聊天命令行界面

学 院： 计算机学院

专业班级： 计算机科学与技术（卓越）02班

年 级： 2019级

姓 名： 李燕琴

学 号： 20195633

完成时间： 2021 年 12 月 31 日

成 绩：

指导教师： 李双庆

重庆大学教务处制

1. 需求分析
   1. 需求陈述

采用自己熟悉的编程语言，基于socket编写一个简单的网络聊天软件。

* 1. 需求理解

基于TCP/IP架构（如图 1所示），利用socket通信原理（如图 2所示），实现两个用户端之间的通信，其中每个用户即是发起聊天消息的客户端，也是需要接收对方聊天消息的服务段。

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。Socket把复杂的TCP/IP协议族隐藏在Socket接口后面，利用三元组（ip地址，协议，端口）就可以唯一标识网络中的进程，网络中的进程通信可以利用这个标志与其它进程进行交互。对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议即可。

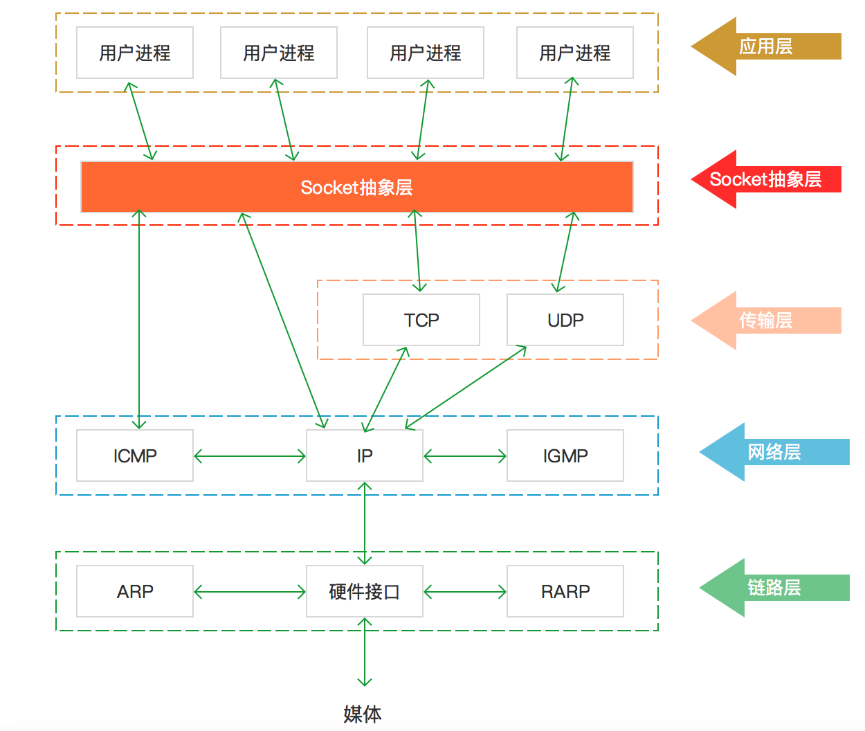


图 1 TCP/IP架构

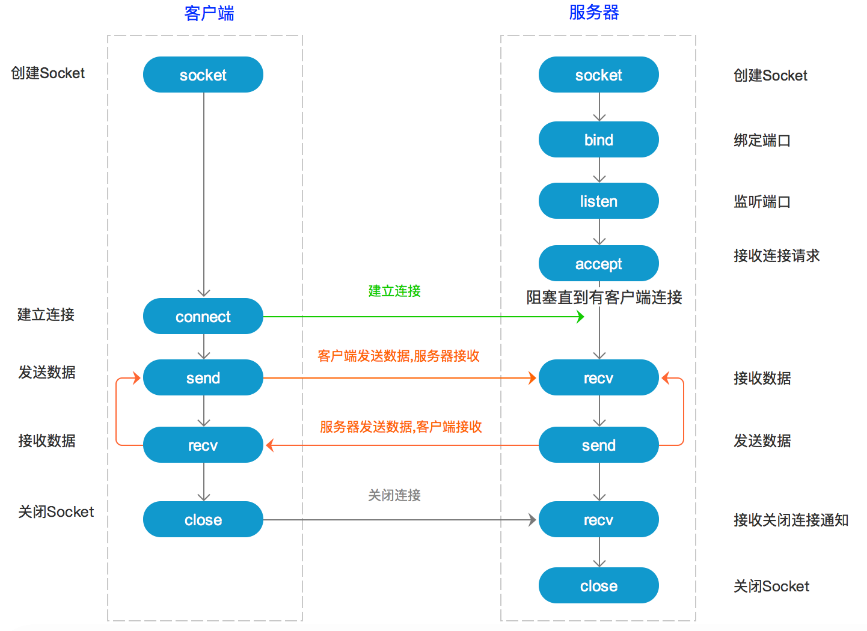


图 2 socket TCP通信原理（图源于网络）

1. 系统设计
   1. 系统逻辑

根据 **1.2需求理解**，本系统实现逻辑如图 3所示，通过创建两个Talker进程，由两个进程分别创建服务器线程（Receiver）、客户端线程（Sender），每个线程由socket管理进程间的通信。其中Sender，收到用户输入的消息，封装为UDP数据报，由socket发送给Receiver所在进程；Receiver收到数据报，对数据报进行解析。

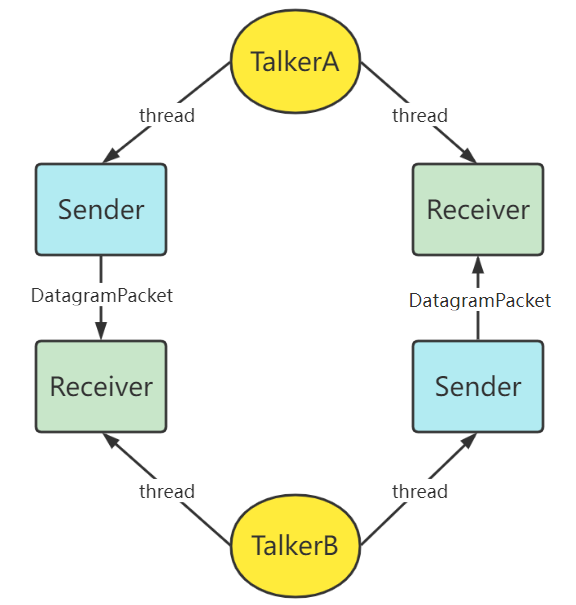


图 3 系统逻辑设计

* 1. 类的设计
     1. Receiver

用户接收线程（或C/S体系中的服务端），基于java.lang. Runnable实现，用于监听端口，接收数据并解析数据。类图如图 4所示：

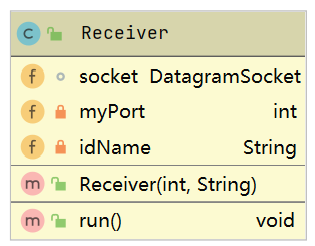


图 4 Receiver类

（1）成员变量

|  |  |  |
| --- | --- | --- |
| 变量 | 类型 | 备注 |
| socket | java.net.DatagramSocket | 接收数据报 |
| myPort | int | 该线程的端口号 |
| idName | String | 该线程所在进程用户的名字 |

（2）成员函数

|  |  |
| --- | --- |
| 成员函数定义 | |
| Receiver(int myPort,String idName) | Receiver的构造函数 |
| void run(); | 线程run()函数重载，监听接口并解析数据 |

* + 1. Sender

用户发送线程（或C/S体系中的客户端），基于java.lang. Runnable实现，用接收用户的输入数据，并构造数据报，向接收端发送数据。类图如图 5所示：

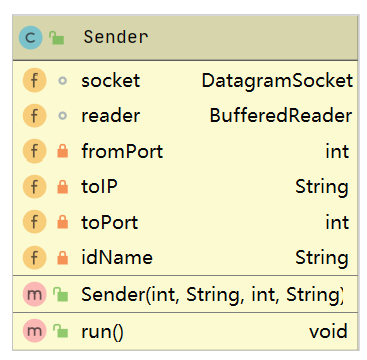


图 5 Sender类

（1）成员变量

|  |  |  |
| --- | --- | --- |
| 变量 | 类型 | 备注 |
| socket | java.net.DatagramSocket | 发送数据报 |
| Reader | java.io.BufferedReader | 接收用户的输入 |
| fromPort | int | 发送端端口号 |
| toIp | string | 接收端的ip地址 |
| toPort | int | 接收端端口号 |
| idName | String | 该线程所在进程用户的名字 |

（2）成员函数

|  |  |
| --- | --- |
| 成员函数定义 | |
| Sender(int fromPort, String toIP, int toPort,String idName) | Sender的构造函数 |
| void run(); | 线程run()函数重载，接收用户数据，构造数据报并发送 |

* + 1. Talker

根据该类创建用户，管理接收消息线程和发送消息进程，类图如图 6所示。

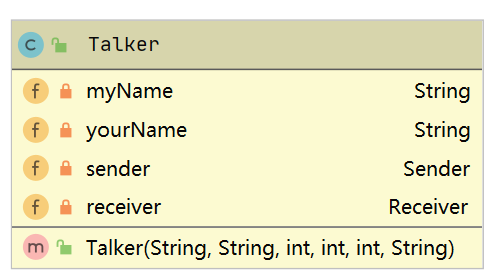


图 6 Talker 类

（1）成员变量

|  |  |  |
| --- | --- | --- |
| 变量 | 类型 | 备注 |
| myName | String | 用户自己的姓名 |
| yourName | String | 好友的姓名 |
| sender | Sender | 发送消息的线程 |
| receiver | Receiver | 接收消息的线程 |

（2）成员函数

|  |  |
| --- | --- |
| 成员函数定义 | |
| Talker(String myName, String yourName, int receivePort, int sendPort, int toPort, String toIp) | Talker的构造函数 |

* 1. 测试逻辑

本系统主要基于UDP报文原理实现，故主要端口逻辑设计如下：

通过创建talkerA进程，并指定自己的接收端口和对方的接收端口，具体实现如图 7所示；再创建talkerB进程，其接收端口与A中指定的接收端口一致，其发送端口可以任意指定，具体如图 8所示。其中TalkerA和TalkerB即Talker类的对象实现。



图 7 TalkerA进程



图 8 TalkerB进程

1. 系统实现

运行TalkerA.java和TalkerB.java两个程序，如图 9所示，其中TalkerA向TalkerB发送数据后，TalkerB的聊天框中仍然同步显示了TalkerA发送的数据；同样地，TalkerB发送的数据，TalkerA的聊天框中仍然同步显示，具体如图 10，图 11所示。

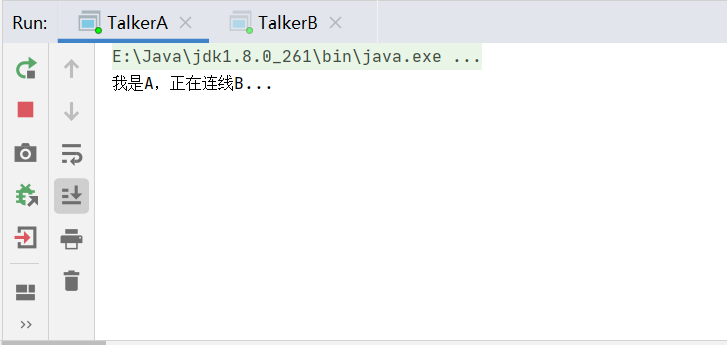


图 9 进程执行

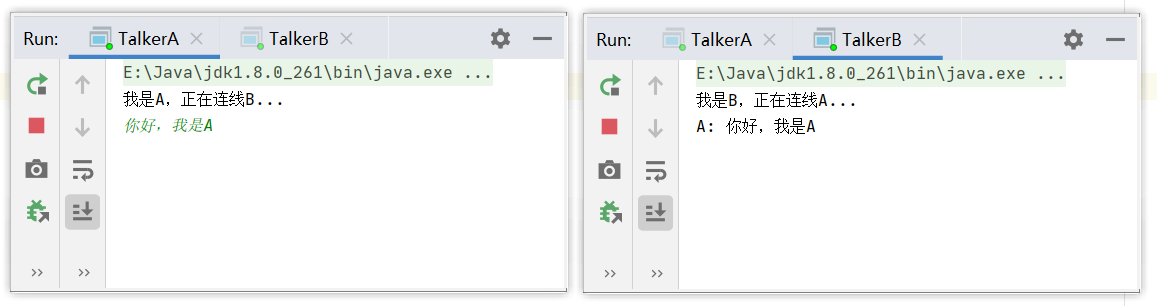


图 10 A向B发送数据

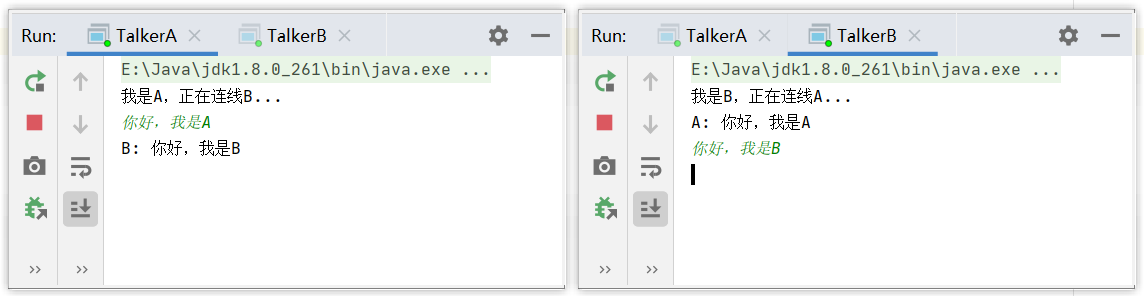


图 11 B向A发送数据

1. 总结

通过本次项目设计，更加深入地了解了通过高级语言实现UDP通信过程。通过Sender和Receiver两个类实现接受和发送消息的socket，通过Talker这个类链接这两个socket，组成网络聊天系统设计。总体封装性较好。

但是由于java封装性比较好，接触到的socket通信原理性的知识尚少。

1. 全部源代码
   1. Sender
2. **import** java.io.BufferedReader;
3. **import** java.io.InputStreamReader;
4. **import** java.net.DatagramPacket;
5. **import** java.net.DatagramSocket;
6. **import** java.net.InetSocketAddress;
8. /\*\*
9. \* @author Maxpicca
10. \* @Date 2021-11-21 19:44
11. \* @Description 发送方将读入的数据发送出去
12. \*/
13. **public** **class** Sender **implements** Runnable {
14. DatagramSocket socket=**null**;
15. BufferedReader reader=**null**;
17. **private** **int** fromPort;
18. **private** String toIP;
19. **private** **int** toPort;
20. **private** String idName;
22. /\*\*
23. \* 信息发送端
24. \* @param fromPort 己方发送端口
25. \* @param toIP 对方的ip
26. \* @param toPort 对方接收端口
27. \*/
28. **public** Sender(**int** fromPort, String toIP, **int** toPort,String idName) {
29. **this**.idName = idName;
30. **this**.fromPort = fromPort;
31. **this**.toIP = toIP;
32. **this**.toPort = toPort;
33. **try** {
34. // 基于己方接口，创建数据报 socket
35. socket = **new** DatagramSocket(**this**.fromPort);
36. // 创建输入消息的reader
37. reader = **new** BufferedReader(**new** InputStreamReader(System.in));
38. } **catch** (Exception e) {
39. e.printStackTrace();
40. }
41. }

44. @Override
45. **public** **void** run() {
46. // 进程一直维持运行状态
47. **while**(**true**) {
48. **try** {
49. // 输入的消息，按照字节解码
50. // System.out.print(this.idName +": ");
51. String msg = reader.readLine();
52. **byte**[] buffer = msg.getBytes();
53. // 创建UDP数据报，包括内容，内容长度，对方地址和对方接口
54. DatagramPacket packet = **new** DatagramPacket(buffer, 0, buffer.length, **new** InetSocketAddress(**this**.toIP, **this**.toPort));
55. socket.send(packet);
56. // 确定程序退出条件
57. **if**("exit".equals(msg)) {
58. **break**;
59. }
60. } **catch** (Exception e) {
61. e.printStackTrace();
62. }
63. }
64. // 退出后关闭当前发出的socket
65. socket.close();
66. }
68. }
    1. Receiver
69. **import** java.net.DatagramPacket;
70. **import** java.net.DatagramSocket;
71. **import** java.net.SocketException;
73. /\*\*
74. \* @author Maxpicca
75. \* @Date 2021-11-21 19:42
76. \* @Description 接收方将收到的数据输出
77. \*/
78. **public** **class** Receiver **implements** Runnable {
79. DatagramSocket socket=**null**;
80. **private** **int** myPort;
81. **private** String idName;
83. /\*\*
84. \* 信息接收端
85. \* @param myPort 己方接收端口
86. \* @param idName 对方信息
87. \*/
88. **public** Receiver(**int** myPort,String idName) {
89. **this**.myPort = myPort;
90. **this**.idName = idName;
91. **try** {
92. // 创建己方接收的数据报socket
93. socket = **new** DatagramSocket(**this**.myPort);
94. } **catch** (SocketException e) {
95. e.printStackTrace();
96. }
97. }
99. @Override
100. **public** **void** run() {
101. **while**(**true**) {
102. **try** {
103. **byte**[] buffer = **new** **byte**[1024];
104. DatagramPacket packet = **new** DatagramPacket(buffer, 0, buffer.length);
105. // 获取数据报
106. socket.receive(packet);
107. // 数据报的数据字段(byte格式)
108. **byte**[] data = packet.getData();
109. // byte转为字符串
110. String msg = **new** String(data,0,data.length);
111. // 接收消息
112. System.out.println(**this**.idName+": "+msg);
114. **if**("exit".equals(msg)) {
115. **break**;
116. }
117. } **catch** (Exception e) {
118. e.printStackTrace();
119. }
120. }
121. // 关闭接收窗口
122. socket.close();
123. }
125. }
     1. Talker
126. /\*\*
127. \* @author Maxpicca
128. \* @Date 2022-01-01 17:35
129. \* @Description Talker类实现
130. \*/
131. **public** **class** Talker {
132. **private** String myName;
133. **private** String yourName;
134. **private** Sender sender;
135. **private** Receiver receiver;
137. **public** Talker(String myName, String yourName, **int** receivePort, **int** sendPort, **int** toPort, String toIp) {
138. **this**.myName = myName;
139. **this**.yourName = yourName;
140. System.out.print("我是"+**this**.myName+"，正在连线"+**this**.yourName+"...\n");
141. **this**.sender = **new** Sender(sendPort,toIp,toPort, myName);
142. **this**.receiver = **new** Receiver(receivePort, yourName);
143. **new** Thread(**this**.sender).start();
144. **new** Thread(**this**.receiver).start();
145. }
146. }
     1. TalkerA
147. **import** java.io.PrintStream;
149. /\*\*
150. \* @author Maxpicca
151. \* @Date 2021-11-21 19:45
152. \* @Description 创建执行者A
153. \*/
154. **public** **class** TalkerA {
156. **public** **static** **void** main(String[] args) {
157. **new** Talker("A","B",11000,12000,21000,"localhost");
158. }
160. }
     1. TalkerB
161. /\*\*
162. \* @author Maxpicca
163. \* @Date 2021-11-21 19:45
164. \* @Description 创建执行者B
165. \*/
166. **public** **class** TalkerB {
168. **public** **static** **void** main(String[] args) {
169. **new** Talker("B","A",21000,22000,11000,"localhost");
170. }
172. }