河南理工大学

《计算机网络课程设计》

课程设计报告

（2016—2017 学年第二学期）

题　　目　简单的即时通信软件

学生姓名　　刘 壮 飞

专业班级　　软件 15-1

学生学号　　311509060128

教师姓名　　李 莹 莹

成　　绩：

评　　语：

教师签名：

日期：

目 录

一.设计任务书

1.1 设计任务

本文设计的是一个简单的即时通信软件，利用 Java Socket 进行点到点通信，其工作机制模仿即时通信软件的基本功能，已实现的功能有：

1）客户端登录

2）客户端退出

3）群组成员之间传输文字或图片信息

该软件分为客户端与服务器端，客户端负责与服务器建立连接，且执行收发消息的操作，服务器端负责等待客户端连接并保存用户的昵称与客户端 Socket 的输出流的对应关系。

1.2 技术指标

本程序使用的是 TCP 协议实现的即时通信软件，程序是基于 Java 语言开发的，主要用到的技术有：

1）Socket 编程

2）自定义协议

如果使用普通的方法来标记一条消息的结束，如换行符，那么程序就不易扩展，只能发送纯文本消息，所以需要自己定义一种消息的格式，并且我们还需要提供发送消息与解析消息的方法。

服务器端创建一个 ServerSocket，循环等待客户端的连接，每当有客户端连接，就获取到客户端的 Socket 对象，并将该对象交付给一个服务器端线程处理，服务器端线程会不断从 Socket 的输入流中解析出消息类型、长度及消息内容，然后根据类型执行不同的操作。

客户端与服务器建立连接，同时开启一个客户端线程接收服务器端发送的消息，客户端登录是向服务器发送一条登录命令，客户端向服务器发送一条消息首先需要包装成定义的消息格式，然后再发送给服务器。

不管是发送消息还是发送命令其实本质都是一条消息，向服务器发送的消息都必须按照定义的格式来。

1.3 论证结果

经论证，这个任务是可行的。TCP 协议的实现细节 Java Socket 已经帮我们做好了，我们需要做的是定义一个协议工具类，实现发送消息与接收消息的方法，

然后客户端与服务器端都利用这两个方法来进行消息的发送与解析。

二.实现原理

2.1 基于 TCP 协议的即时通信

TCP 协议是一种端到端协议，当一台计算机要与远程的另一台计算机连接时，TCP 协议会让他们建立一个用于发送和接收数据的虚拟链路。TCP 要负责收集数据信息包，并将其按照适当的次序放好传送，接收端收到后再正确的还原，TCP 协议使用了重发机制，当一个通信实体发送一个消息到另一个通信实体后，需要接收到另一个通信实体的确认消息，如果没有收到确认消息，则会重发消息。所以 TCP 协议保证了数据包在传输中不发生错误。通信示意图如图 1 所示。

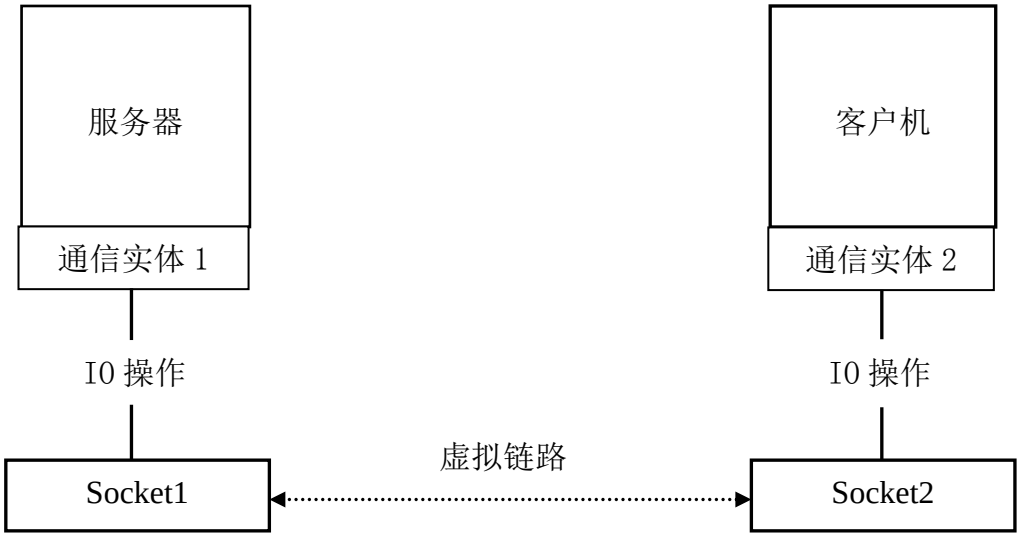

图 1. 通信实体之间通信示意图

在通信实体 1 与通信实体 2 建立虚拟链路前，必须有一方主动来接收其他通信实体的连接请求，作出“主动”的通信实体称为服务器，发出连接请求的通信实体称为客户机。

2.2 自定义协议的定义

2.2.1 通信原理

客户端与服务器端相互通信，首先要建立 Socket 连接，连接建立好后双方都会拿到一个 Socket 对象，通过 Socket 对象拿到输入、输出流可以实现写、读的功能。服务器端接收到客户端的连接，将客户端的 Socket 对象交付给一个线程，该线程负责维护该客户端，在线程体中需要使用死循环不断的获取客户端发给服务器的消息。

2.2.2 存在的问题

那么问题来了：怎么标志客户端发送的消息的结尾？如果不对结尾标志，服务器端将不知道客户端本次客户端发送的消息到哪里。

2.2.3 文本消息的解决办法

对文本消息的一般做法是将‘\n’作为结尾标记，操作过程如下：

1）客户端与服务器端建立连接，服务器端将客户端的 Socket 加入集合中保存，并将客户端的 Socket 交付给一个服务器端线程处理，服务器线程初始化套接字、输入输出流，然后一直循环等待客户端发送消息

2）客户端向服务器发送消息“Hello World！\n”，服务器线程获取到客户端发送的消息，然后使用输入流读取一行消息，读取到的消息是“Hello World!”，然后遍历服务器端的那个集合，获取到集合中每个 Socket 的输出流，向每个输出流中写入消息。

以上是一般意义上群聊的实现原理：客户端向服务器发送消息，服务器获取到消息后转发给群组中的所有成员。

2.2.4 依然存在的问题

问题一：如果发送的是图片、文档应该怎么标记消息的结束？

问题二：在实际应用中，客户端向服务器端发送消息并不像刚才的例子那么简单，还需要登录、注销、登录成功等命令，怎么来区别这些命令？

2.2.5 自定义协议的内容

为解决以上问题，我们规定：消息的发送与解析都必须使用以下格式：

表 1.自定义协议的格式

type	totalLen	bytes[]
1 字节	4 字节	长度不定

由表 1 知，数据分为三个部分

type：1 字节，表示发送的消息类型，所以可以表示 65535 种消息类型。

totalLen:4 字节,整型数据,表示发送的消息的总长度,包含 type、totalLen 的长度以及消息内容的长度，totalLen 占用 4 字节，所以最大可以发送 2G 的数据。

bytes[]：字节数组，表示发送的消息的内容，大小没有限制。

2.2.6 使用自定义协议

制定消息的规范后以上两个问题都会迎刃而解了，客户端向服务器端发送消息的过程如下：

例 1：发送纯文本

客户端：

1）客户端的视图与用户交互获取到用户要发送的文本内容“你好啊”

2）客户端将获取到的文本内容转化为字节数组 bytes

3）客户端将消息包装成自定义的消息格式，如表 2 所示

表 2.发送纯文本的格式

TYPE_TEXT	bytes.length+5	bytes

4）客户端往输出流中写入消息

服务器端：

1）服务器端线程一直等待接收客户端的消息

2）服务器端线程获取到客户端发送的消息，按照格式解析出消息的类型、长度以及消息内容

3）服务器端线程获取到的消息类型是文本类型 TYPE_TEXT，那么需要遍历服务器端的集合，获取到集合中每个 Socket 的输出流，使用这个输出流对消息转发，在转发前同样需要包装成定义的消息格式。

例 2：登录功能

客户端：

1）客户端与服务器端建立连接之后，将用户输入的昵称包装成一条消息，如表 3 所示，消息类型是 TYPE_LOAD，字节数组 bytes 是用户昵称。

表 3.发送登录命令的格式

TYPE_LOAD	bytes.length+5	bytes

2）在建立连接的同时客户端会开启一个线程等待接收服务器端的消息。

3）将消息发给服务器端。

4）客户端线程如果收到服务器端反馈的信息，就将信息告知用户。

服务器端：

1）接收到消息后获取到消息类型为 TYPE_LOAD，服务器端就可以知道这条消息是登录请求，然后 bytes[]数组里的数据就是用户昵称，将用户昵称、该客户端的 Socket 对象的输出流先保存到 Map 中，然后将该 Map 保存到集合中。

2）服务器端线程对客户端的登录请求处理完成后，向客户端反馈一条标记着是否操作成功的消息，类型是 TYPE_LOADSUCCESS 和 TYPE_LOADFAIL，bytes 数组是服务器端反馈的消息。

如果登录成功，服务器反馈的消息格式如表 4 所示。

表 4.反馈登录成功命令的格式

TYPE_LOADSUCCESS	bytes.length+5	bytes

如果登录失败，服务器反馈的消息格式如表 5。

表 5.反馈登录失败命令的格式

TYPE_LOADFAIL	bytes.length+5	bytes

2.2.7 小结

其实不管客户端发送的消息是哪种类型，客户端只需要负责将要发送的消息转化为字节数组，然后对数据包装后发送给服务器。对于一些非命令类的消息，服务器接收到消息后只需要根据数据类型对数据进行解析、包装、转发即可。对于一些命令类的消息，如登录，退出等功能，则需要服务器端执行相应的操作，服务器端不需要对消息转发，可能需要对一些命令给予反馈。

通过自定义协议可以解决上述的两个问题，并给出了客户端与服务器端使用自定义协议发送消息的两个详细步骤

2.3 自定义协议的实现

这个自定义协议就是自己定义的一个发送消息、解析消息的规范，无论是发消息还是收消息都必须按照这个规范来，实现这个协议无非需要考虑三个问题：

问题一：如何发送消息？

问题二：如何解析消息？

问题三：如何表示解析消息后的结果？

我们只需要定义两个类，协议工具类 Protocol 负责消息的发送与解析，消息结果类 Result 封装了一个消息的三个部分：type、totalLen、bytes，协议工具类对消息解析后会返回一个 Result 对象表示一次解析的结果。所以这两个类结合起来使用就可以解决以上三个问题。

2.3.1 协议工具类的实现

协议工具类 Protocol 负责消息的发送与解析，内部需要定义消息的格式，协议工具类的设计如图 2 所示。

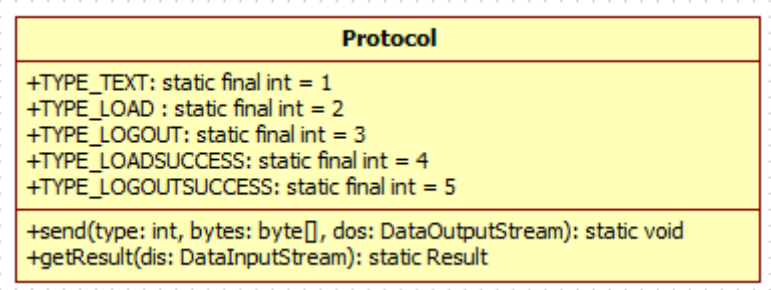

图 2. Protocol 类类图

1）消息类型

消息类型是协议工具类的静态的公共的整型常量，这样的设置为程序后期的扩展提供了方便，提供的消息类型如表 6 所示。

表 6.消息类型

静态常量	含义
TPYE_TEXT	文本类型
TPYE_LOAD	登录命令
TYPE_LOGOUT	退出命令
TYPE_LOGOUTSUCCESS	退出成功
TYPE_LOADSUCCESS	登录成功

2）发送消息

发送消息就是按照定义的格式往输出流中写入数据，我们首先要做的是包装数据

定义方法签名，如表 7 所示。

表 7.发送消息的方法签名

返回值类型	方法名	方法说明
void	send(int type, byte[] bytes, DataOutputStream dos)	发送消息。type 是消息类型，bytes[]是字节数组型的消息内容，dos 是数据输出流。

1）包装数据

一条数据有三部分，消息类型、消息内容已经通过参数获取到了，消息的长度还要程序计算：消息长度=消息的内容的长度+5 字节。

```
int totalLen = 1 + 4 + bytes.length;
```

2）按格式写入输出流，先写入消息类型，然后写入消息的总长度，最后再写入消息的内容。

```
dos.writeByte(type);
dos.writeInt(totalLen);
dos.write(bytes);
dos.flush();
```

3）解析消息

解析消息是指将从输入流中读取到一条消息，然后按照格式转化为一个结果对象 `Result`。

定义方法签名如表 8 所示。

表 8.解析消息的方法签名

返回值类型	方法名	方法说明
Result	getResult(DataInputStream dis)	解析消息。dis 是输入流，本方法从输入流 dis 中解析出一条数据，返回一个结果对象

1）消息提取

从输入流中依次读取三部分：type、totalLen、bytes[]，dis 是方法的参数，调用方法时需要传入输入流

```
byte type = dis.readByte();
int totalLen = dis.readInt();
byte[] bytes = new byte[totalLen - 4 - 1];
dis.readFully(bytes);
```

2）结果返回

将提取出来的数据的三个部分封装成一个结果对象作为方法的返回值，注意第一个参数：type & 0xFF，因为 type 是字节，需要与 0xFF 进行“与”运算得到一个整型值。

```
return new Result(type & 0xFF, totalLen, bytes);
```

2.3.2 结果类的实现

结果类 Result 封装一条消息的三个部分，主要提供了 setter、getter 方法来设置或者获取消息的三个组成部分，结果类的设计如图 3 所示。

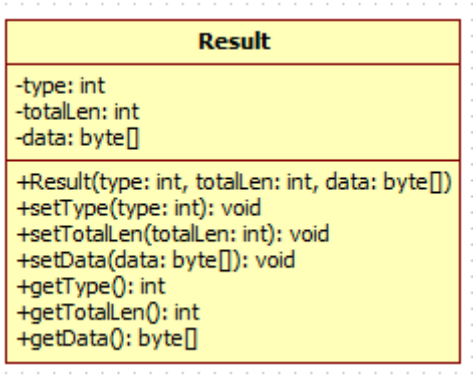

图 3. Result 类类图

1）消息格式

Result 类定义了消息的格式，消息的组成如表 9 所示。

表 9. 消息格式

类型	属性名	含义
int	type	消息类型
int	totalLen	消息总长度
byte[]	data	消息内容

2）方法

Result 类的方法签名如表 10 所示。

表 10. Result 类的方法签名

返回值类型	方法签名	含义
	Result(int type,int totalLen,byte[] data)	构造方法，构造一条消息
void	setType(int type)	设置消息的类型
void	setTotalLen(int totalLen)	设置消息总长度
void	setData(byte[] data)	设置消息的内容
int	getType()	获取消息类型
int	getTotalLen()	获取消息总长度
byte[]	getData()	获取消息内容

2.4 服务器端的实现

2.4.1 服务器类

Server 类负责等待客户端连接并将连接上的客户端交付给服务器线程类。Server 类的设计如图 4 所示。

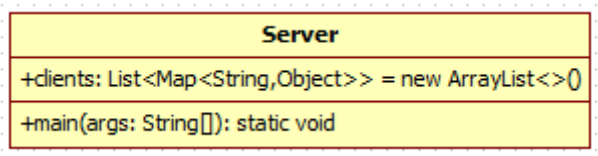

图 4. Server 类类图

clients 维护一个 List 集合，集合中每个元素都是 Map，每个 Map 中都有两个 item，保存着客户端的昵称和对应的输出流。

main 方法中要实现的是等待客户端连接，使用 ServerSocket，有客户端连接时开启一个线程来处理。代码如下：

```
ServerSocket serverSocket = new ServerSocket(30000);
while (true) {
        Socket socket = serverSocket.accept();
        new Thread(new ServerThread(socket)).start();
        }
```

2.4.2 服务器线程类

ServerThread 类负责接收客户端的消息并对消息进行相应的处理，ServerThread 类的设计如图 5 所示。

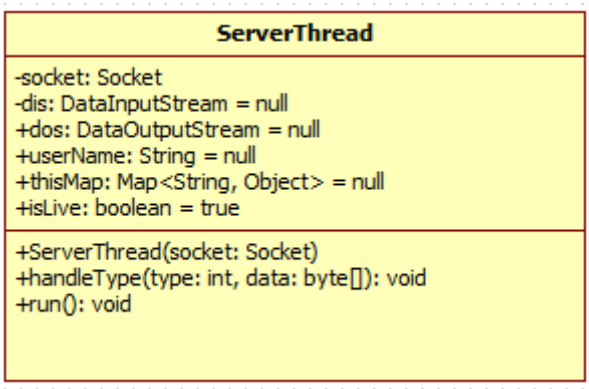

图 5. ServerThread 类类图

1）变量

ServerThread 类的变量以及其含义如表 11 所示。

表 11. ServerThread 的变量表

类型	属性名	含义
Socket	socket	客户端的套接字
DataInputStream	dis	客户端的输入流
DataOutputStream	dos	客户端的输出流
String	userName	客户端昵称
Map<String,Object>	thisMap	客户端的 map
boolean	isLive	该线程是否生存

2）方法签名

ServerThread 类的方法签名以及含义如表 12 所示。

表 12. ServerThread 类的方法签名

返回值类型	方法签名	含义
	ServerThread(Socket socket)	构造方法
void	run()	线程体
void	handleType(int type, byte[] data)	根据消息类型处理客户端消息

2.5 客户端的实现

2.5.1 客户端界面

界面的元素有：登录输入框、聊天内容文本域、消息输入文本域、发送按钮。客户端界面初始化时会调用 Client 的方法执行客户端与服务器的连接请求，连接成功后客户端与服务器端会形成一个虚拟链路，当用户输入用户名后回车，客户端通过该虚拟链路向服务器端发送一条登录命令。View 类的设计如图 6 所示。

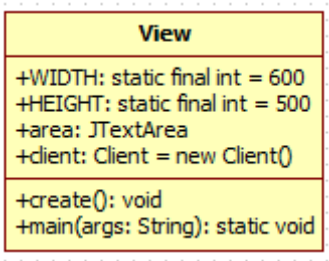

图 6. View 类类图

2.5.2 客户端

客户端 Client 负责处理客户端连接、客户端发送消息的任务，Client 类的设计如图 7 所示。

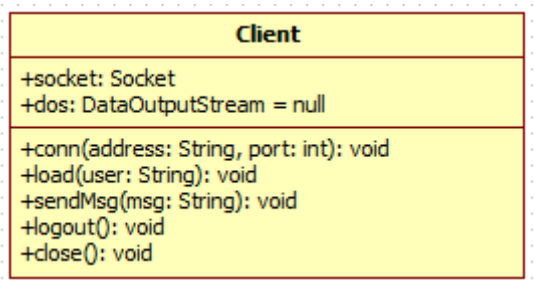

图 7. Client 类类图

1）建立连接

```
socket = new Socket(address, port);
dos = new DataOutputStream(socket.getOutputStream());
// 监听服务器发回的消息
new ClientThread(socket).start();
```

2）登录

```
public void load(String user) {
        Protocol.send(Protocol.TYPE_LOAD,user.getBytes(), dos);
    }
```

3）发送消息

```
public void sendMsg(String msg) {
        Protocol.send(Protocol.TYPE_TEXT, msg.getBytes(), dos);
    }
```

4）退出

```
public void logout(){
        Protocol.send(Protocol.TYPE_LOGOUT, "logout".getBytes(),
dos);
    }
```

客户端线程负责接收服务器端发的消息，其对消息的处理方法与服务器端线程的处理方法类似，都是先解析消息，然后根据消息类型执行相应的操作。

2.5.2 客户端线程

客户端线程主要负责接收消息，并对接收到的消息进行显示。ClientThread 类的设计如图 8 所示。

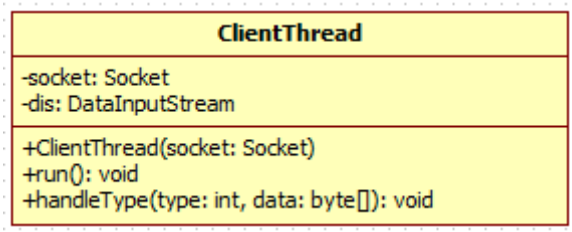

图 8.ClientThread 类类图

ClientThread 类维护的是客户端的套接字以及输入流，那三个方法的作用和服务器端线程类似，这里不再细说。

三.实验结果

3.1 运行结果

1）服务器端启动后是没有运行界面的，运行结果如图 9 所示。

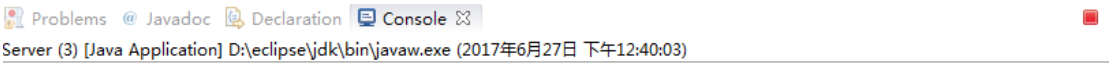

图 9. 服务器启动后效果图

2）客户端启动后初始界面如图 10 所示。

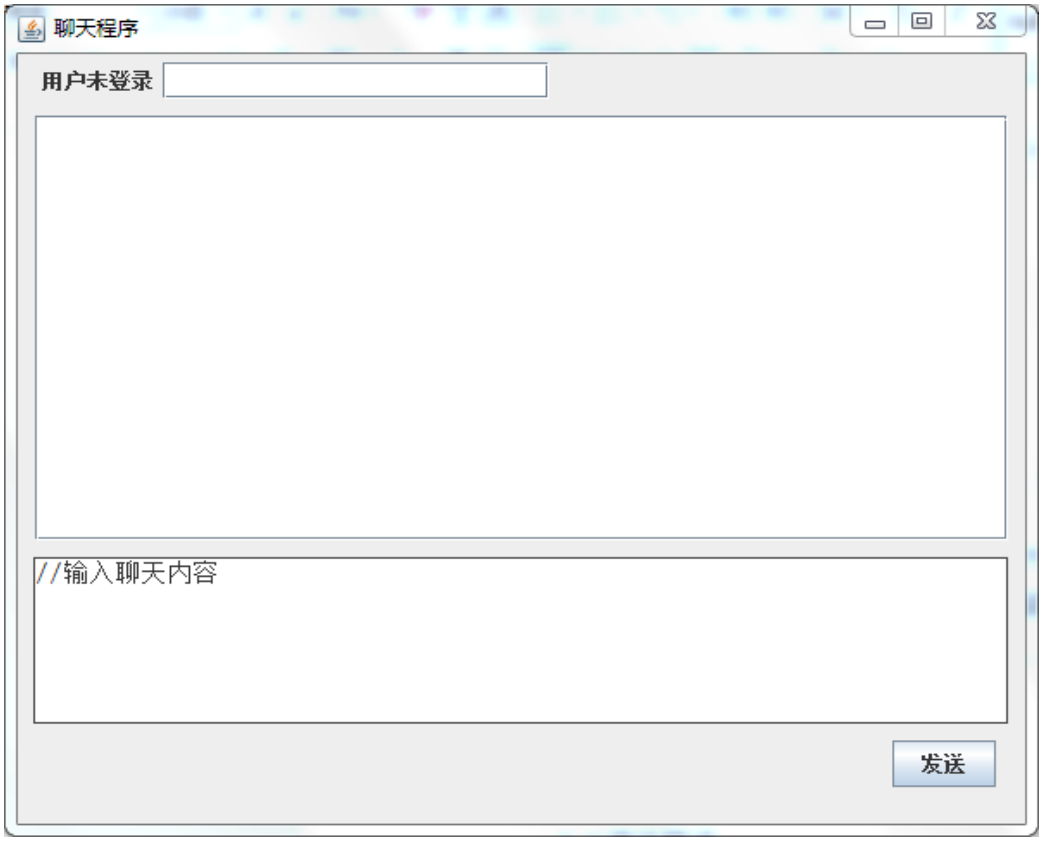

图 10. 客户端初始化界面图

3）输入用户名后回车登录，登录后如图 11 所示。

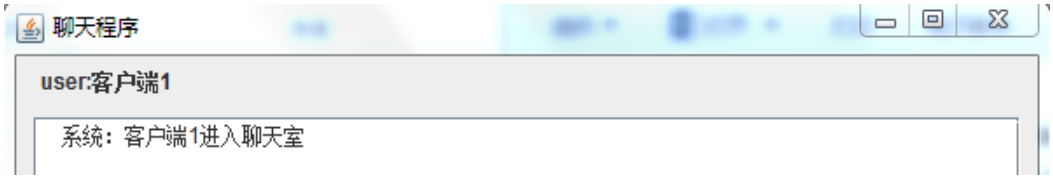

图 11. 客户端登录后效果图

4）单个客户端退出

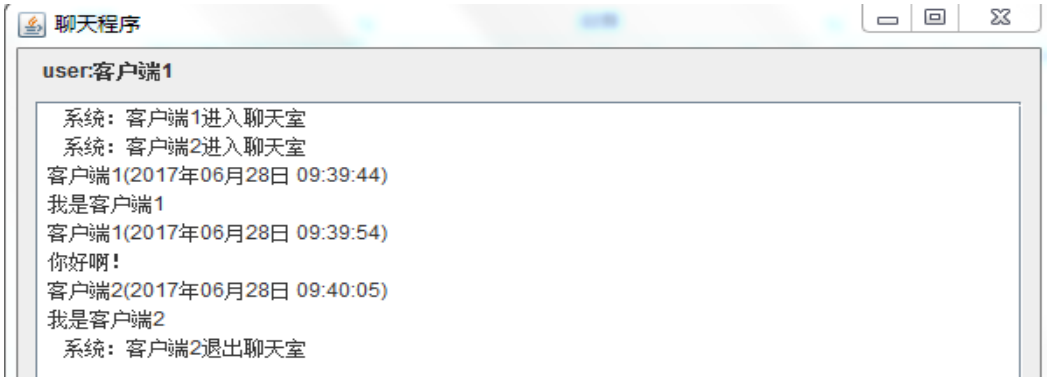

图 13. 客户端退出

5）两个客户端互发消息，如图 12 所示。

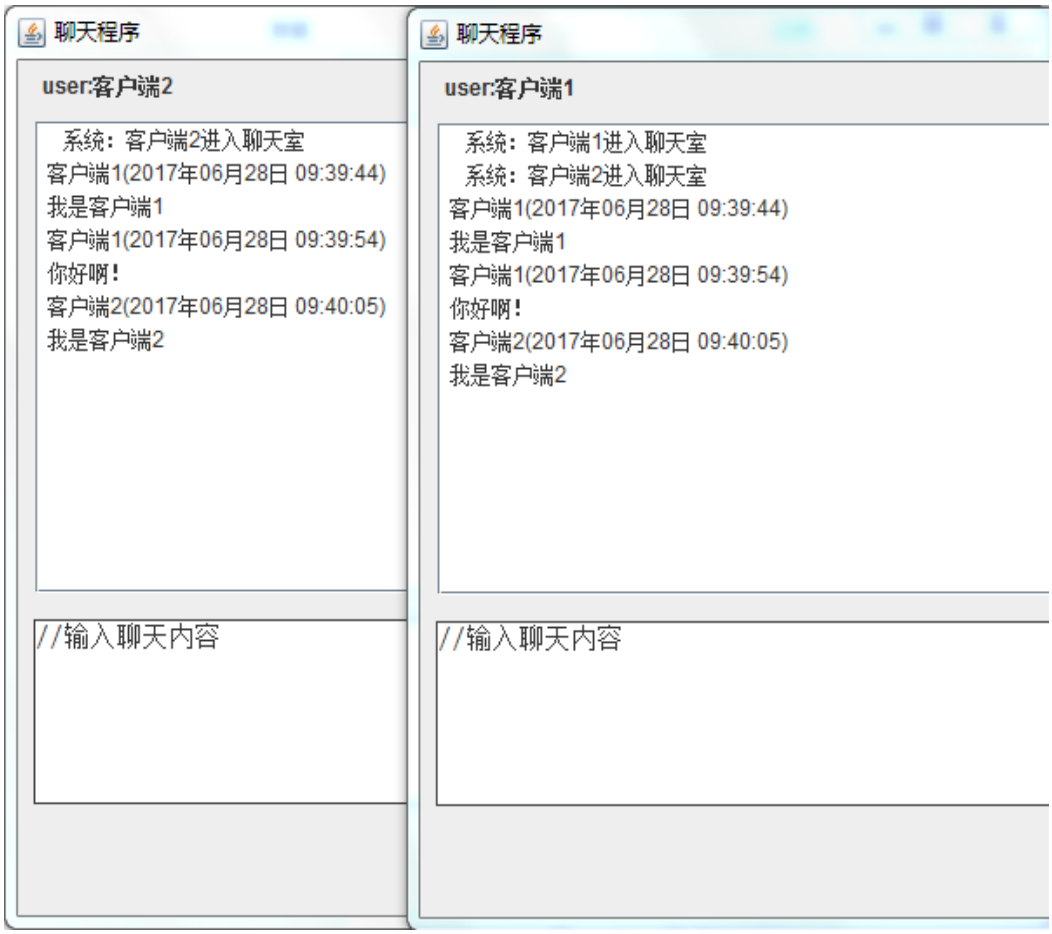

图 12. 两个客户端相互通信

3.2 主要问题及故障分析

3.2.1 主要问题

1）不知道如何标记一条消息的结尾

2）界面问题

3.2.2 故障分析

对于第一个问题：如果只是发送一条文本消息的话，是没有这个问题的。但是为了使程序拥有更好的扩展性，使其可以发送图片以及文档，这个问题还是值得思考的。定义一种消息的格式，无论是发送还是接收消息都按照这个标准来，这个就是我们定义的协议工具类的作用。

对于第二个问题：本程序是基于 Java 语言开发的，AWT 和 SWING 是 Java 语言开发 GUI 的工具包。SWING 和 AWT 写界面都不是很方便，所以本程序的界面有点粗糙。

3.3 设计结论

由于之前写过这类的程序，所以在程序层次上的实现并不难，本次实验不仅巩固了编写程序的功底，还加深了对 Socket 通信底层理论的理解，可以说是收获非常大。至此，本论文已经接近尾声，所研究的是一个简单的即时通信软件的实现过程以及实现原理。

四.附录一:实验相关

4.1 实验数据

客户端与客户端 2 发送的消息数据，[]内部表示的是发送方的昵称，昵称外部是发送的消息内容，具体实验数据如下:

[客户端 1]我是客户端 1

[客户端 1]你好啊!

[客户端 2]我是客户端 2

4.2 系统软硬件环境

4.2.1 硬件环境

系统：Window7 旗舰版

系统类型：64 位操作系统

处理器：i5-4210U

安装内存：4.0GB

4.2.2 软件环境

已安装 JRE、JDK 并配置好环境变量

4.3 使用说明

本程序分为客户端与服务器端，首先需要启动服务器端，然后可以打开多个客户端，客户端打开后可以进行聊天。

4.4 参考资料

[1] 李刚，疯狂 Java 讲义第 3 版，北京：电子工业出版社，2014.7。

[2] James F.Kurose，Keith W.Ross，计算机网络-自顶向下方法上册（第 5 版），北京：高等教育出版社。

五.附录二:程序源代码

5.1 客户端类

```
package client;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

import myutil.Protocol;

/**
 * 封装客户端与服务器通信的细节
 */
public class Client {

    //套接字
    Socket socket;

    //输出流
    DataOutputStream dos = null;

    /**
     * 连接服务器并初始化输出流
     * 开启客户端线程负责消息的接收
     * @param address 服务器IP地址
     * @param port 服务器端口号
     */
    public void conn(String address, int port) {
        try {
            socket = new Socket(address, port);
            dos = new DataOutputStream(socket.getOutputStream());
            new ClientThread(socket).start();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 登录
     * @param user 用户昵称
```

```
     */
    public void load(String user) {
        Protocol.send(Protocol.TYPE_LOAD,user.getBytes(), dos);
    }

    /**
     * 发送消息
     * @param msg 消息内容
     */
    public void sendMsg(String msg) {
        Protocol.send(Protocol.TYPE_TEXT, msg.getBytes(), dos);
    }

    /**
     * 退出
     */
    public void logout(){
        Protocol.send(Protocol.TYPE_LOGOUT, "logout".getBytes(), dos);
    }

    /**
     * 关闭客户端，释放掉资源
     */
    public void close() {
        // 向服务器发送退出命令
        Protocol.send(Protocol.TYPE_LOGOUT, new String("logout").getBytes(),
dos);
        // 关闭资源
        try {
            if (dos != null)
                dos.close();
            if (socket != null)
                socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5.2 客户端线程端

```
package client;
import java.io.DataInputStream;
import java.io.IOException;
import java.net.Socket;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;

import myutil.Protocol;
import myutil.Result;

/**
 * 客户端消息线程
 * 用以接收服务器消息
 * @author Administrator
 *
 */
public class ClientThread extends Thread {
    private Socket socket;//套接字
    private DataInputStream dis;//输入流

    //初始化套接字与输入流
    public ClientThread(Socket socket) {
        this.socket=socket;
        try {
            dis=new DataInputStream(socket.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        while(true){
            //解析消息
            Result result =    Protocol.getResult(dis);
            if(result!=null)
            //根据消息类型处理
            handleType(result.getType(),result.getData());
        }
    }

    /**
     * 根据消息的类型对消息处理
     * @param type  消息类型
     * @param data  消息内容
     */
    private void handleType(int type, byte[] data) {
        SimpleDateFormat df=new SimpleDateFormat("yyyy年MM月dd日
```

```
hh:mm:ss");
            String time=df.format(new Date());
            switch (type) {
            case 1:
                //文本
                String[] args=new String(data).split("说：");

                View.area.append("    "+args[0]+"("+time+")\n    "+args[1]+"\n");
                break;
            case 4:
                View.area.append("    "+new String(data)+"\n");
                break;
            case 5:
                View.area.append("    "+new String(data)+"\n");
            default:
                break;
            }
            View.area.select(View.area.getText().length(),
View.area.getText().length());
        }
}
```

5.3 视图类

```
package client;

import java.awt.*;
import java.awt.event.*;
import java.util.List;

import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

/**
 * 聊天视图
 *
 * @author Administrator
 *
 */
public class View {

    // 窗口属性值
    private final int WIDTH = 600;
    private final int HEIGHT = 500;
```

```
// 聊天记录文本域
public static JTextArea area;

// 客户端实体对象
Client client=new Client();

/**
 * 创建一个视图
 */
public void create() {
    // 连接服务器
    client.conn("127.0.0.1", 30000);

    //窗口
    JFrame frame = new JFrame("聊天程序");

    // 登录面板
    JPanel loadPanel = new JPanel();
    loadPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
    frame.add(loadPanel, BorderLayout.NORTH);

    // 标签以及输入框
    final JLabel userLabel = new JLabel("      用户未登录");
    final JTextField userTextField = new JTextField(20);
    //添加
    loadPanel.add(userLabel);
    loadPanel.add(userTextField);

    //设置回车登录事件
    userTextField.addKeyListener(new KeyAdapter() {
        @Override
        public void keyReleased(KeyEvent e) {
            if (e.getKeyCode() == 10) {
                String user = userTextField.getText();
                if (user != null && !user.equals("")) {
                    client.load(user);
                    userLabel.setText("      user:" + user);
                    userTextField.setText("");
                    userTextField.setVisible(false);
                }
            }
        }
    });
```

```
        // 聊天记录面板
        JPanel topPanel = new JPanel();
        loadPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        // 聊天记录文本域
        area = new JTextArea(14, 51);
        area.setEditable(false);
        // 滚动条
        JScrollPane jsp = new JScrollPane(area,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        //添加
        frame.add(topPanel);
        topPanel.add(jsp);

        // 底部输入面板
        JPanel bottomPanel = new JPanel();
        frame.add(bottomPanel, BorderLayout.SOUTH);
        bottomPanel.setPreferredSize(new Dimension(WIDTH, 165));
        // 文本域
        final JTextArea ta = new JTextArea();
        ta.setBorder(BorderFactory.createLineBorder(Color.darkGray));
        ta.setFont(new Font("宋体", Font.PLAIN, 15));
        ta.setPreferredSize(new Dimension(WIDTH - 35, 100));
        ta.setText("//输入聊天内容");
        ta.select(0, 0);
        ta.setLineWrap(true);
        //设置回车发送消息
        ta.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                if (e.getKeyChar() == KeyEvent.VK_ENTER) {
                    if (!ta.getText().equals("") &&
userLabel.getText().indexOf("user:") != -1) {
                        client.sendMsg(ta.getText());
                        ta.setText("");
                    } else {
                        System.out.println("用户未登录或内容为空");
                    }
                    e.consume();
                }
            }
        });

        //输入聊天输入框随鼠标的动态效果
```

```
        ta.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseEntered(MouseEvent e) {
                if (ta.getText().equals("") || ta.getText().equals("//输入聊天内容"))
                    ta.setText("");
            }

            @Override
            public void mouseExited(MouseEvent e) {
                if (ta.getText().equals(""))
                    ta.setText("//输入聊天内容");
            }
        });

        // 按钮面板
        JPanel buttonPanel = new JPanel();
        buttonPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 20));
        buttonPanel.setPreferredSize(new Dimension(WIDTH, 50));
        buttonPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));

        // 按钮
        JButton sendButton = new JButton("发送");
        buttonPanel.add(sendButton);
        sendButton.setFocusPainted(false);
        //添加按钮点击发送事件
        sendButton.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                if (ta.getText() != null && ta.getText().length() != 0 &&
userLabel.getText().indexOf("user:") != -1) {
                    client.sendMsg(ta.getText());
                    ta.setText("");
                } else {
                    System.out.println("用户未登录或内容为空");
                }
            }
        });

        // 底部面板添加控件
        bottomPanel.add(ta);
        bottomPanel.add(buttonPanel);
        //添加窗口关闭自动退出系统事件
        frame.addWindowListener(new WindowAdapter() {
```

```
            @Override
            public void windowClosing(WindowEvent e) {
                client.logout();
            }
        });

        // 窗口设置
        frame.setSize(WIDTH, HEIGHT);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    /**
     * 主函数，程序的入口
     * 执行视图的实例化
     * @param args
     */
    public static void main(String[] args) {
        View view=new View();
        view.create();
    }
}
```

5.4 协议工具类

```
package myutil;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

/**
 * 协议工具类
 * 封装了消息的类型以及发和收的方法
 * @author Administrator
 *
 */
public class Protocol {
    // text
    public static final int TYPE_TEXT = 1;

    // 登录
    public static final int TYPE_LOAD = 2;

    // 退出
```

```
public static final int TYPE_LOGOUT = 3;

//登录成功
public static final int TYPE_LOADSUCCESS = 4;

//退出成功
public static final int TYPE_LOGOUTSUCCESS = 5;

/**
 * 向输出流中发送消息
 * @param type 消息类型
 * @param bytes 消息内容
 * @param dos 输出流
 */
public static void send(int type, byte[] bytes, DataOutputStream dos){
    int totalLen = 1 + 4 + bytes.length;
    try {
        //依次读取消息的三个部分
        dos.writeByte(type);
        dos.writeInt(totalLen);
        dos.write(bytes);
        dos.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * 从输入流中解析消息
 * @param dis 输入流
 * @return 解析之后的结果
 */
public static Result getResult(DataInputStream dis) {
    byte type;
    try {
        //依次取出消息的三个部分
        type = dis.readByte();
        int totalLen = dis.readInt();
        byte[] bytes = new byte[totalLen - 4 - 1];
        dis.readFully(bytes);
        //返回解析结果
        return new Result(type & 0xFF, totalLen, bytes);
    } catch (IOException e) {
        e.printStackTrace();
```

```
        }
        return null;
    }
}
```

5.5 结果类

```
package myutil;

/**
 * 封装一个消息，亦是一次解析的结果
 */
public class Result {
        //消息类型
        private int type;

        //消息总长度
        private int totalLen;

        //消息内容
        private byte[] data;

        //以消息的三个部分构造一个消息实体
        public Result(int type, int totalLen, byte[] data) {
            super();
            this.type = type;
            this.totalLen = totalLen;
            this.data = data;
        }

        //以下是setter、getter方法
        public int getType() {
            return type;
        }
        public void setType(int type) {
            this.type = type;
        }
        public int getTotalLen() {
            return totalLen;
        }
        public void setTotalLen(int totalLen) {
            this.totalLen = totalLen;
        }
        public byte[] getData() {
            return data;
        }
```

```
        public void setData(byte[] data) {
            this.data = data;
        }
}
```

5.6 服务器类

```
package Server;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * 服务器类
 * 负责接受客户端的连接
 * 将客户端的连接交付给服务器端线程处理
 */
public class Server {
    //维护客户端的配置信息
    public static List<Map<String,Object>> clients=new ArrayList<>();

    //主方法
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(30000);
            while (true) {
                Socket socket = serverSocket.accept();
                new Thread(new ServerThread(socket)).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5.7 服务器线程类

```
package Server;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.HashMap;
```

```
import java.util.Map;
import myutil.Protocol;
import myutil.Result;

/**
 * 服务器端线程
 * 负责与客户端通信
 * @author Administrator
 *
 */
public class ServerThread implements Runnable{
    //套接字
    public Socket socket;

    //输入、输出流
    public DataInputStream dis=null;
    public DataOutputStream dos=null;

    //用户昵称
    public String userName=null;

    //用户配置信息的 Map
    public Map<String, Object> thisMap=null;

    //标志线程是否生存
    public boolean isLive=true;

    /**
     * 构造服务器端线程实体
     * 初始化输入、输出流
     * @param socket  客户端套接字
     */
    public ServerThread(Socket socket){
        this.socket=socket;
        try {
            dis=new DataInputStream(socket.getInputStream());
            dos=new DataOutputStream(socket.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 线程体
```

```
     */
    public void run() {
        while(isLive){
            //解析消息
            Result result = null;
            result = Protocol.getResult(dis);
            if(result!=null)
            //按类型处理
            handleType(result.getType(),result.getData());
        }
    }

    /**
     * 根据消息类型执行相应操作
     * @param type 类型
     * @param data 消息内容
     */
    public void handleType(int type, byte[] data) {
        switch (type) {
        case 1:
            //遍历集合，获取输出流
            //向所有用户转发消息
            for(int i=0;i<Server.clients.size();i++){
                System.out.println("message:"+new String(data));
                DataOutputStream dos2=(DataOutputStream)
Server.clients.get(i).get("dos");
                String msg=new String(data);
                Protocol.send(Protocol.TYPE_TEXT,(userName+"  说  :
"+msg).getBytes(),dos2);
            }
            break;
        case 2:
            //设置配置信息并添加至服务器端的集合中
            userName=new String(data);
            Map<String,Object> map=new HashMap<>();
            map.put("dos",dos);
            map.put("user",userName);
            Server.clients.add(map);

            //通知所有用户有人登陆聊天室
            thisMap=map;
            for(int i=0;i<Server.clients.size();i++){
                DataOutputStream dos2=(DataOutputStream)
Server.clients.get(i).get("dos");
```

```
                    Protocol.send(Protocol.TYPE_LOADSUCCESS, ("        系 统 ：
"+userName+"进入聊天室").getBytes(), dos2);
                }
                break;
            case 3:
                //告知所有用户有人要退出聊天室
                for(int i=0;i<Server.clients.size();i++){
                    DataOutputStream                    dos2=(DataOutputStream)
Server.clients.get(i).get("dos");
                    Protocol.send(Protocol.TYPE_LOGOUTSUCCESS, ("      系 统 ：
"+userName+"退出聊天室").getBytes(), dos2);
                }
                //删除集合中保存的该客户端信息
                Server.clients.remove(thisMap);
                isLive=false;
                break;
            default:
                break;
            }
        }
}
```