北京航空航天大學
BEIHANG UNIVERSITY

多客户数据流 Socket 网络程序说明文档

姓名： 冯飘飘

学号： 12061154

日期： 2015.06.09

目录

1 引言 5

1.1 编写目的 5

2 实验介绍 5

2.1 实验目的.....5

2.2 实验内容.....6

2.3 实验原理.....6

2.3.1 Socket 类和 ServerSocket 类.....6

2.3.2 Java 的多线程.....7

2.4 实验环境.....7

3 实验内容 8

3.1 任务概述 8

3.1.1 目标.....8

3.1.2 用户的特点 8

3.1.3 假定和约束 8

3.2 需求规定 9

3.2.1 对功能的规定 9

3.2.2 对性能的规定 9

3.2.3 输入输出要求 10

3.2.4 数据管理能力要求 10

3.2.5 故障处理要求 10

4 详细设计说明 11

4.1 程序系统的结构 11

4.2 服务器设计说明 13

4.2.1 程序描述 13

4.2.2 功能.....13

4.3 服务器端口监听程序设计说明 13

4.3.1 程序描述 13

4.3.2 功能.....13

4.3.3 输入/输出 14

4.3.4 流程逻辑 14

4.4 客户服务程序设计说明 14

4.4.1 程序描述 14

4.4.2 功能.....14

4.4.3 输入/输出 15

4.4.4 流程逻辑 15

4.5 客户登录界面设计说明 15

4.5.1 程序描述 15

4.5.2 功能.....15

4.5.3 输入/输出 16

4.6 聊天室界面设计说明 16

4.6.1 程序描述 16

4.6.2 功能.....17

4.6.3 输入/输出 17

4.7 服务器消息监听程序设计说明 18

4.7.1 程序描述 18

4.7.2 功能 18

4.7.3 输入/输出 18

4.7.4 流程逻辑 18

5 编码实现 19

5.1 界面类代码 19

5.1.1 用户登录界面 19

5.1.2 聊天窗口界面 20

5.2 SOCKET 通信代码 23

5.3 线程类代码 24

6 系统测试 27

6.1 用户登陆测试 27

6.2 界面内容更新测试 28

7 总结 29

1 引言

1.1 编写目的

本文档介绍了网络应用编程实验要求实现的通信程序——多客户数据流 Socket 网络编程，分别从需求、设计、编码、测试等角度对该程序做了详细的介绍。

本文档的对象是关注 Socket 通信程序实现，有基本的 Java 编程语言能力的读者。可以用过阅读本文档，了解程序实现的技术细节，对 Java 实现多线程 Socket 程序有一个直观的认识。

2 实验介绍

2.1 实验目的

- 了解 Java 语言实现 Socket 编程的方法
- 初步掌握多线程服务器的设计和开发方法

2.2 实验内容

采用 Java 语言提供的 Socket 类和 ServerSocket 类开发一个基于 Socket 的多客户 C/S 文本传输程序，客户端能够通过聊天界面输入信息发送到服务器端，服务器能够同时接受多个客户端发来的信息，并显示返回给各个客户端。相当于一个小型的聊天软件。

2.3 实验原理

本实验利用 Java 语言及其提供的类库编写一个多线程的 Socket 网络程序。Java 语言为 Java 网络提供了两种类：软件包 java.net 中的类和与流相关的类。

数据流套接字编程采用客户 /服务器方式，并使用 java.net 类库中的 Socket 类为客户端提供套接字， ServerSocket 类为服务器提供套接字。 Socket 类和 ServerSocket 类一起作用来建立客户端和服务器端之间的双向通信连接。

利用 Java 语言编写网络程序用到了 DataOutputStream、 DataInputStream 等与流相关的类。

本实验中服务器同时处理多个客户请求，使用了并发服务器模式。 Java 语言提供了多线程技术来支持并发服务器模式。

2.3.1 Socket 类和 ServerSocket 类

- Socket 类

Socket 类用于实现客户端套接字，用来初始化并创建一个客户端对象。Socket 类为客户端套接字提供了四个构造函数。本程序中只使用了其中一个

public Socket(String host,int Port) throws UnknownHostException,IOException

用于构造一个连接到指定主机(host)和端口(port)的 Socket 类，会产生 UnknownHostException,IOException,java.lang.Security Exception 异常。

- ServerSocket 类

ServerSocket 类用于实现服务器端套接字，负责监听和响应客户端的连接请求，并接受

客户端发送的数据。ServerSocket 类为客户端套接字提供了三个构造函数。本程序中只使用了其中一个

Public ServerSocket(int port) throws IOException

参数 port 指定服务器要绑定的端口(服务器要监听的端口)，当运行时无法绑定到指定端口时，Socket 构造方法会抛出 BindException 异常；当客户进程发出的连接请求被服务器拒绝时，Socket 构造方法会抛出 ConnectionException 异常。

2.3.2 Java 的多线程

多线程是在同一程序中可同时运行多个执行流程，也就是多个线程。与多任务不同的是，各线程共享同一数据空间；如果一个线程中的全局变量的值发生了变化，那么所有其他的线程都将能够观察到这个变化。多线程在 Java 网络编程中常被用于编写服务器。多线程服务器提供一个主服务器线程和多个处理线程，主服务器线程将专注于接受客户连接，为每一个客户连接启动处理线程，处理线程将服务于已接受的客户连接。 Java 语言提供了两种实现多线程的方法：一种是继承 Thread 类实现多线程程序，另一种是通过 Runnable 接口实现多线程。

2.4 实验环境

- Windows7
- Eclipse
- JDK1.7

3 实验内容

3.1 任务概述

3.1.1 目标

按照网络实验课程对该通信程序的要求，需要使用的 Socket 和多线程两项主要技术。完成的通信程序可以运行在学生自选的操作系统上。

本软件是一个可独立运行的通信程序，而不是一个更大系统的组成部分。其实现的功能是使用 Socket 和多线程技术实现一个聊天程序，只能支持多人同时聊天（聊天室功能）。

由于该程序是使用 Java 语言开发的，其运行时需要有 Java 虚拟机的支持，本程序开发过程中使用的是 JDK 1.7。

3.1.2 用户的特点

该软件的用户即是参与到聊天群组当中的个人。观察目前较流行的商业聊天软件，如 QQ、MSN 等可以发现，参与到聊天过程中的人具有不同的教育水平和技术专长，这就要求软件的使用界面应尽量简单，方便各种层次的用户使用。

该软件作为网络实验课程的实践部分，可以预见，其今后的应用范围和频度都比较小，基本上局限于同学之间内部使用，没有广泛的用户群，软件内部的容错、均衡机制可以暂不考虑。

软件维护人员需要具备 Java 通信程序的开发经验，了解该程序的执行过程，掌握相关调试工具，能够及时发现和排除程序中存在的 Bug。

3.1.3 假定和约束

进行本软件开发工作的假定和约束主要有：

- 相关程序和文档由个人独立完成；
- 程序开发过程中需要使用到 Socket 和多线程技术；
- 实现基本的多人聊天功能；

- 在 2015 年 6 月 11 日前完成相关程序的调试和文档撰写。

3.2 需求规定

3.2.1 对功能的规定

如 3.1.3 节所述，本软件需要提供的功能只有对多人聊天的支持。

图 3.1 展示的是多人聊天功能。登录到系统中的每一个用户都可以向同时在线的其他用户发送消息。每个登录到系统当中的用户都可以拥有这样一个图例描述的功能。

除了上述基本功能外，还需要考虑其周边的功能，如登录服务器、新用户登陆提示、新消息到达提示等。

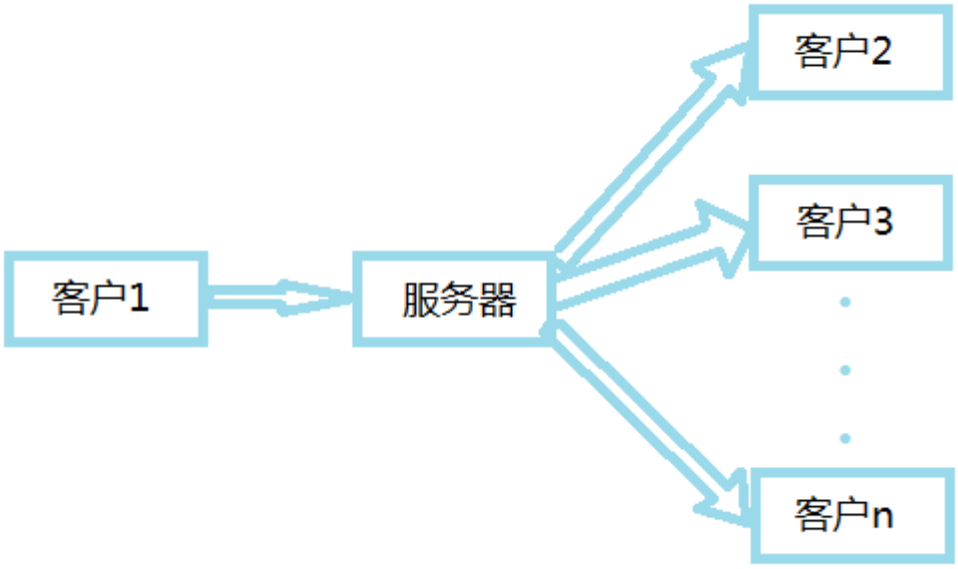

图 3.1 多人聊天功能

3.2.2 对性能的规定

该程序应具备较好的响应时间，其响应的事件主要有：

- 能够尽快发现新近登录/退出系统的其他用户，及时把这些信息反馈给当前用户；
- 数据传输速度：用户登陆客户端之后，通过一个公共的服务器端转发他们的通信数据。消息发送的效率一方面受网络情况的影响，另一方面也会受服务器端的影响。

3.2.3 输入输出要求

该软件完成的功能主体是网络聊天。其输入数据主要是是登陆到系统中的用户输入的文本信息；输出是把一个用户输入的文本信息传送，并输出到另一个用户的界面上。

对输入的要求是避免使用不规则字符，由于程序为了方便对用户输入信息进行解析，会自动在用户字符串中插入一些不常用字符作为分隔符，若用户输入的文本中包含同样的字符，有可能会导致系统运行出错。

由于输出数据只是输入数据在另一处的再现，所以没有特殊规定，只要是正常输入的数据都可以正常输出。

3.2.4 数据管理能力要求

该程序的用户范围和使用频率较低，一次处理的数据量也较小。数据处理主要发生在服务器端，服务器端需要首先解析客户端发来的消息类型（登录，退出，广播等），然后做出相应的动作。

由聊天的特点，每个用户一次发送的数据量较小，大多是一句或几句话，服务器端需要同时处理多个客户端发来的短句。

3.2.5 故障处理要求

聊天程序可能出现的故障通常有：无法连接到服务器，程序掉线，消息发送失败，程序崩溃等。其中无法连接到服务器，程序掉线，消息发送失败可能由网络状况引起（如网络拥塞），也可能是软件内部发生了逻辑错误；而程序崩溃则主要是由于程序自身的缺陷引起的。

面对可能出现的故障，需要程序能够及时给用户一个提示，让用户了解到当前发生地状况，用户可以把这些状况反馈给系统管理员，进行后续的分析与修复。

4 详细设计说明

4.1 程序系统的结构

按照事先的功能，可将程序可划分为两个部分：一部分是服务器，它负责监听端口，为每个连入的用户提供数据转发服务；另一部分是客户端，负责和服务器建立连接，监听服务器发送过来的数据，并提交给用户。

图 4.1 展示了服务器端运行的流程。服务器端采用了并发服务的模式，它是一种高效的服务器运行模式，允许服务器同时与多个客户端进行通信。其实现的方法是，服务器端一直处于端口监听状态，一旦有新的客户连接请求就创建一个线程，负责与这个客户进行通信，当这个客户端中断连接或者发生错误后结束相应的服务线程。

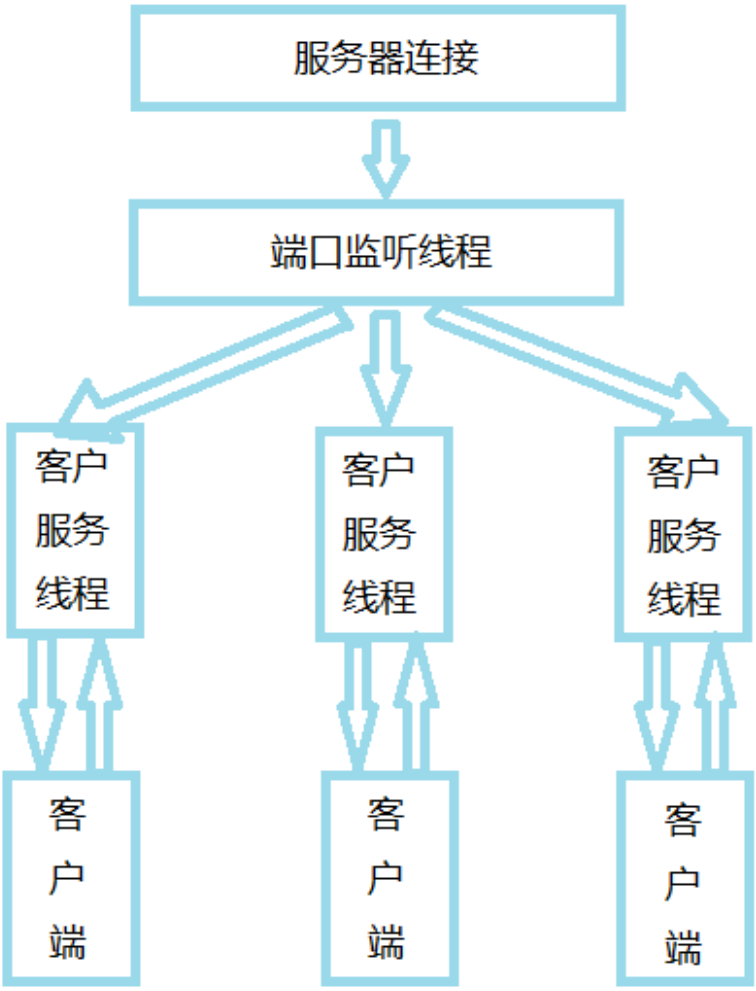

图 4.1 服务器端运行模式

客户端同样使用了线程机制。如图 4.2 所示，用户在使用本程序时首先要进行登陆，登录时用户需要提供所要连接的服务器地址和使用的用户名，一旦服务器收到该连接请求，将通过判断是否有重复用户名的方式来判断当前请求是否是一个重复连接请求。

登录后，用户将看到一个主界面，主界面展示了当前在线的用户，同时为用户提供了进行多人聊天或单人聊天的选择。用户看不到的是，该窗口在创建完成后会建立一个子进程，负责监听服务器端发来的数据，并把这些数据提交给相应的窗口展示出来。

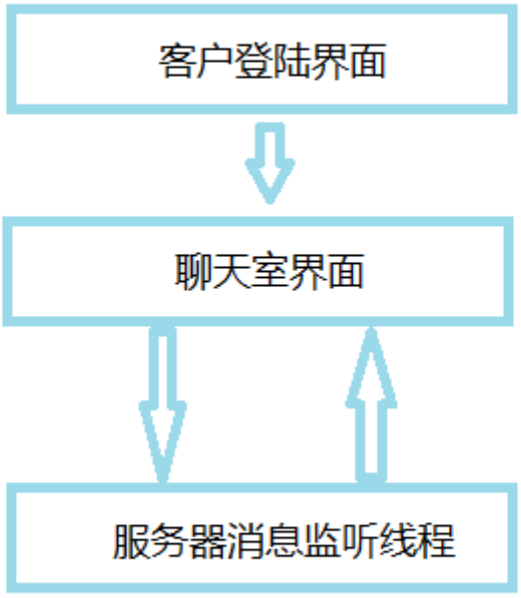

图 4.2 客户端运行模式

4.2 服务器设计说明

4.2.1 程序描述

该部分程序为用户提供了服务器功能。通过本程序启动服务器，将创建一个服务器端口监听程序，负责为每一个接入系统的合法用户创建一个服务线程。程序在退出时，将会杀掉后台的监听进程以及存在的用户服务线程。

4.2.2 功能

该程序提供了三个功能：

- 启动服务器，创建端口监听线程；
- 显示系统状态信息，如启动信息，用户进入、退出，当前用户等；
- 杀掉后台监听线程，Socket 连接。

4.3 服务器端口监听程序设计说明

4.3.1 程序描述

服务器端口监听程序，负责为每一个接入系统的合法用户创建一个服务线程。创建服务线程后，继续返回监听端口状态。

4.3.2 功能

该程序提供了四个功能：

- 监听服务器的一个固定端口（程序中是 8080）；
- 创建客户端服务线程；
- 为服务器界面程序返回状态信息（启动/停止信息）；
- 提供了一个用来记录所有在线用户的全局变量。

4.3.3 输入/输出

输入为客户端发来的连接请求。

输出为与当前客户端建立的 Socket 变量和客户端的用户名，监听线程利用这两个线程创建一个针对该客户的服务线程。

4.3.4 流程逻辑

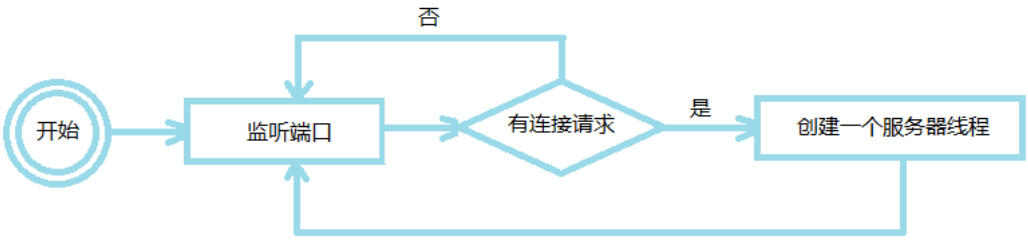

图 4.3 端口监听程序流程

4.4 客户服务程序设计说明

4.4.1 程序描述

客户服务程序是由服务器端口监听程序创建的线程，它负责为每个接入系统的用户提供数据转发的服务。这部分程序是服务器端程序的核心部分。

4.4.2 功能

该程序提供了三个主要功能：

- 用户登陆的合法性验证（有无重复用户名存在）；
- 当有用户登录或退出时，为所有用户提供更新信息；
- 支持多人聊天的广播功能；

4.4.3 输入/输出

输入为客户端通过套接字发送来的各种信息，这些信息描述了客户希望得到何种服务。通过解析客户端发来的信息，做出相应的动作，提供用户需要的服务。

4.4.4 流程逻辑

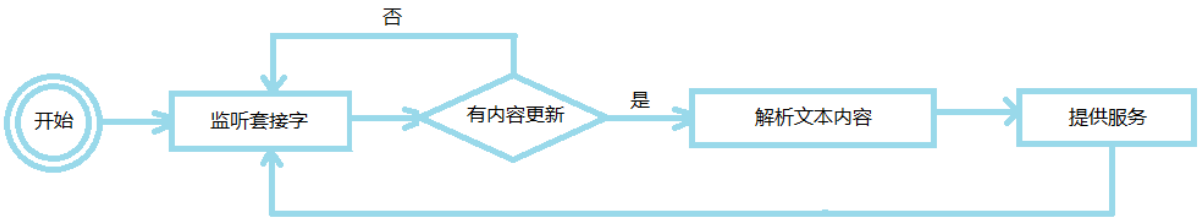

图 4.4 客户服务程序流程

4.5 客户登录界面设计说明

4.5.1 程序描述

该部分程序为用户提供了一个可视化的登录窗口，是聊天用户使用该程序的入口。该入口将检查用户的唯一性信息，若通过检查，则为用户创建一个自己的主界面。

4.5.2 功能

如图 4.5 所示，该程序提供了五个功能：

- 为用户提供用户名输入接口；
- 与指定服务器建立 Socket 连接，若连接失败给出提示信息；
- 为服务器端提供当前希望加入系统的用户名；
- 接收服务器端返回的登录验证结果；
- 根据验证结果或创建主界面，或通知用户更换用户名。

图 4.5 用户登陆界面

4.5.3 输入/输出

输入为用户使用的登录名。

输出或为一条出错提示（服务器无法连接，或用户名重复），或为登录成功确认信息。

4.6 聊天室界面设计说明

4.6.1 程序描述

聊天室对话框程序为用户提供了一个可视化的多人聊天界面。用户把自己希望发送的信息输入到图 4.6 左下角的文本框中，点击发送按钮可以将信息发送到服务器。

4.6.2 功能

如图 4.6 所示，该程序提供了三个功能：

- 为用户提供多人聊天的功能；
- 展示所有用户的群发信息；
- 为服务器消息监听进程提供更新展示界面内容的方法。

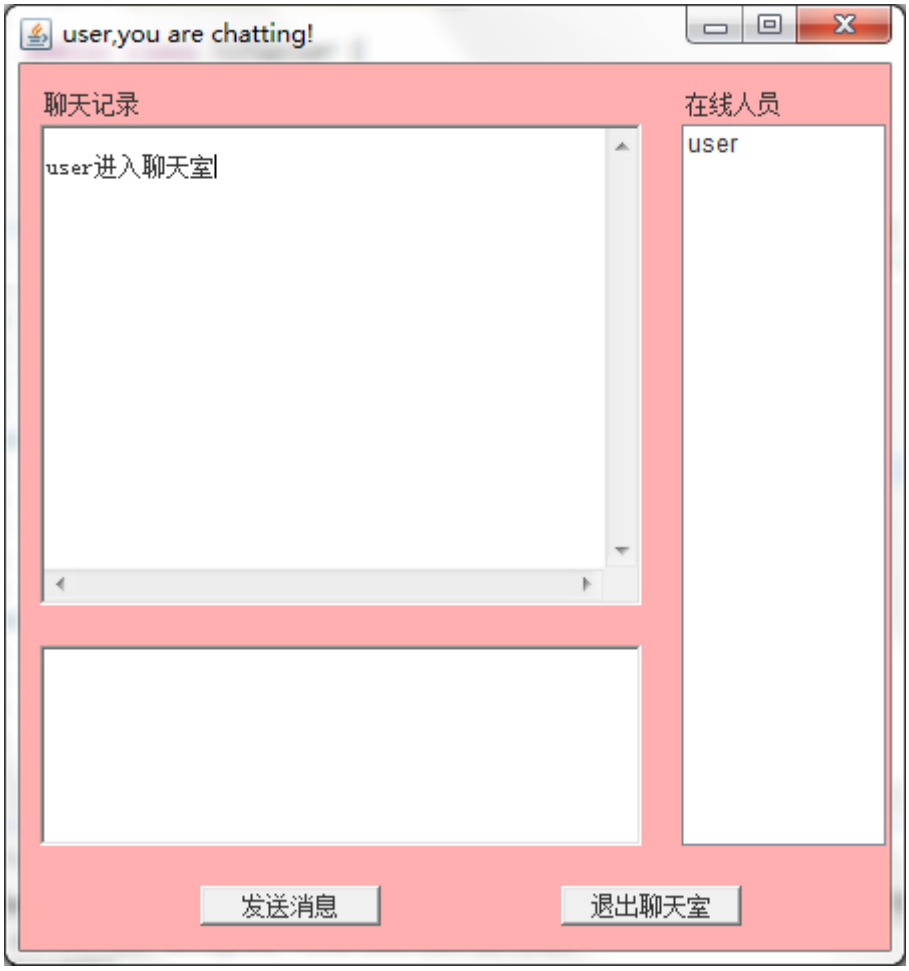

图 4.6 聊天室界面

4.6.3 输入/输出

输入为用户在图 4.6 左下角文本框内输入的文本信息。

输出为增加了功能标识头部的文本信息。

4.7 服务器消息监听程序设计说明

4.7.1 程序描述

该程序负责监听从服务器一端发送来的消息。这些消息包括：人员更新消息，广播消息。监听程序在收到消息后，将根据消息种类做相应的处理。

4.7.2 功能

服务器消息监听程序提供了四个功能：

- 监听从服务器发送的客户端的消息；
- 解析消息的内容；
- 若为更新消息，则通过主界面提供的方法更新聊天室在线列表；
- 若为广播消息，则通过聊天室界面提供的方法更新广播界面展示的内容；

4.7.3 输入/输出

输入为服务器一端的服务线程在 Socket 中写入的信息。

输出为相应界面内容更新的动作。

4.7.4 流程逻辑

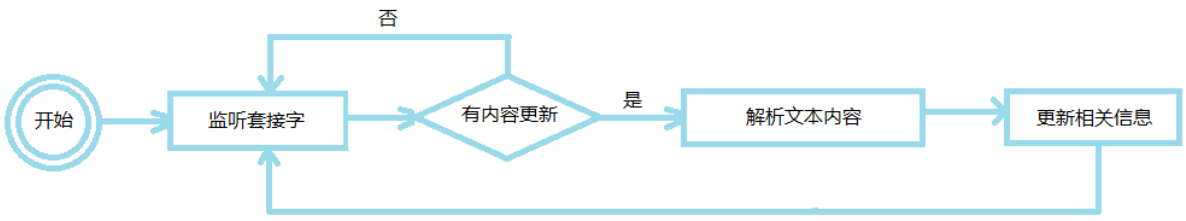

图 4.7 服务器消息监听程序流程

5 编码实现

程序中涉及的编码可以分为三种类型，分别是：界面类代码，Socket 类代码和线程创建/同步类代码。由于代码比较多，下面将分别做简要介绍。

5.1 界面类代码

如第 4 章所述，本程序共包含了 2 个窗口程序：用户登陆界面和聊天室界面。这些界面的编写方式属于同一种类型，其中都用到了 Java 提供的 AWT 组件和 Swing 组件.

窗口程序在初始化时，是按着先构造面板，然后逐个把空间加载到面板上的顺序来完成的。

5.1.1 用户登录界面

```
public Login(Hashtable listTable){
        jflogin = new JFrame("用户登录");
        inputusrname = new Label("请输入用户名:",Label.LEFT);
        usrname = new TextField(30);
        login = new Button("登录");
        cancel = new Button("取消");
        login.addActionListener(this);
        cancel.addActionListener(this);
        thread = new Thread(this);
        this.listTable = listTable;

        jp1 = new JPanel();
        jp2 = new JPanel();
        jp1.add(inputusrname);
        jp1.add(usrname);
        jp2.add(login);
        jp2.add(cancel);

        jflogin.setBounds(300, 400, 300, 200);

        jflogin.setLayout(new GridLayout(2, 1));

        jflogin.add(jp1);
        jflogin.add(jp2);
        jflogin.setVisible(true);
```

```
        jflogin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
```

实现界面上组件监听的程序结构如下列代码所示，其原理是通过 addActionListerner 方法增加了一个处理事件的代码。

```
public void actionPerformed(ActionEvent e) {
        int i;
        name = usrname.getText();
        if (e.getSource() == login) {
            i = JOptionPane.showConfirmDialog(null, "确认登录吗?","提示
",JOptionPane.YES_NO_OPTION);
            if (i == JOptionPane.YES_OPTION){
                if (name.length() == 0){
                    JOptionPane.showMessageDialog(null, "用户名不能为空","提示
", JOptionPane.INFORMATION_MESSAGE);
                }
                else {
                    if (socket != null) {
                        try {   // 将用户信息发送给服务器端
                            out.writeUTF("#昵称:" + name);
                        } catch (IOException e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                        }
                    }
                }
            }
        }
        else if (e.getSource() == cancel) {
            try {
                out.writeUTF("用户离开:");
            } catch (IOException ee) {
            }
            //尚未建立连接，故只需关闭窗口
            this.jflogin.setVisible(false);
            thread.stop();
        }
    }
```

5.1.2 聊天窗口界面

```
public ChatArea(String name, Hashtable listTable, int width, int height){
```

```
this.name = name;
this.listTable = listTable;
this.width = width;
this.height = height;
this.setBounds(300, 400, width, height);

threadMessage = new Thread(this);
MsgRecord = new TextArea(10,10);
EditMsg = new TextField(28);


l1 = new Label("聊天记录",Label.LEFT);
l2 = new Label("在线人员",Label.LEFT);

send = new Button("发送消息");
send.addActionListener(this);
exit = new Button("退出聊天室");
exit.addActionListener(this);

listComponent = new List();
this.setTitle(name+",you are chatting!");
this.setBackground(Color.pink);

Container cont = this.getContentPane();
cont.setBackground(Color.pink);

jp1 = new JPanel();
jp1.setBackground(Color.pink);

jp1.setLocation(0, 0);
jp1.setSize(width, height*3/4 + 50);

jp1.add(l1);
l1.setBounds(10, 10, width*2/3, 20);

jp1.add(MsgRecord);
MsgRecord.setBounds(10, 30, width*2/3, height/2);

jp1.add(EditMsg);
EditMsg.setBounds(10, 50 + height/2, width*2/3, height/4-20);

jp1.add(l2);
l2.setBounds(30+width*2/3, 10, width/4 - 10, 20);
```

```
        jp1.add(listComponent);
        listComponent.setBounds(30 + width*2/3, 30, width/4 - 10, height*3/4);

        jp1.setLayout(null);

        jp3 = new JPanel();
        jp3.setBackground(Color.pink);
        jp3.setLocation(10, 60+height*3/4);
        jp3.setSize(width, height/4 - 50);

        jp3.add(send);
        send.setBounds(width/5, 50 + height*3/4, width/5, 20);

        jp3.add(exit);
        exit.setBounds(width*3/5, 50 + height*3/4, width/5, 20);

        jp3.setLayout(null);

        cont.add(jp1);
        cont.add(jp3);

        this.setVisible(true);
    }
```

事件监听代码：

```
public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
    if (e.getSource() == send){
        String msg = EditMsg.getText();
        if (msg.length() > 0){
            try {
                out.writeUTF("公共聊天内容:"+name +"说:"+"\n"+"   "+ msg);
                System.out.println("公共聊天内容:"+name+"说:"+"\n"+"   "+
msg);
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            EditMsg.setText("");
        }
    }
    else if (e.getSource() == exit){
        int i = JOptionPane.showConfirmDialog(null, "确认离开聊天室吗?", "提示
", JOptionPane.YES_NO_OPTION);
```

```
        if (i == JOptionPane.YES_OPTION){
            try {
                out.writeUTF("用户离线:");
                threadMessage.stop();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            this.dispose();
        }
    }
}
```

5.2 Socket 通信代码

Socket 和 ServerSocket 类库位于 java.net 包中。ServerSocket 用于服务器端，Socket 是建立网络连接时使用的。在连接成功时，应用程序两端都会产生一个 Socket 实例，操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别，不因为在服务器端或在客户端而产生不同级别。

相应的代码如下，客户端首先创建一个 Socket 对象连接到服务器上，然后绑定一个写通道 out 和一个读通道 in，通过调用 DataOutputStream 的 writeUTF() 方法向 Socket 对象写入数据，通过 in 的 readUTF( ) 方法读出套接字中的信息。

```
try{
    socket = new Socket(host,port);
    in = new DataInputStream(socket.getInputStream());
    out = new DataOutputStream(socket.getOutputStream());
}
catch(IOException e){
    e.printStackTrace();
}
```

服务器端程序通过创建 ServerSocket 对象实现建立网络服务器；使用 ServerSocket 类提供的 accept( ) 方法实现对本机端口的 Socket 连接监听。

```
    public TCPServer() throws IOException{
        serverSocket = new ServerSocket(port);
        System.out.println("服务器成功启动....");
    }
//处理客户链接，采用多线程的方式，每个客户链接启动一个工作线程
    public void sevice(){
        while(true){
```

```
            Socket socket = null;
            try{
                //接收客户连接
                socket = serverSocket.accept();
                //为每个客户连接创建一个工作线程
                Thread workThread = new serverThread(socket,peopleList);
                //启动工作线程
                workThread.start();
            }
            catch(IOException e){
                e.printStackTrace();
            }
        }
    }
```

5.3 线程类代码

程序中使用的线程继承自 Java 中的 Thread 类。可以通过编写相应的构造函数来创建线程时需要完成的工作。还需要重写线程的 run() 方法，这个方法描述了线程在启动之后需要完成的功能。

```
public class serverThread extends Thread{

    Socket socket;
    Hashtable peopleList;

    DataInputStream in = null;
    DataOutputStream out = null;

    String name = null;

    public serverThread(Socket socket,Hashtable peopleList){
        this.socket = socket;
        this.peopleList = peopleList;

        try {
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
    }

    // 服务器与客户端通信线程
    public void run(){
        while (true) {
            String s = null;
            try {
                s = in.readUTF();
                if (s.startsWith("#昵称:")){
                    name = s.substring(s.indexOf(":")+1);
                    if (peopleList.containsKey(name)){
                        out.writeUTF("昵称已存在:");
                    }
                    else {
                        out.writeUTF("可以开始聊天啦:");
                        // 显示新的socket连接及其IP地址和端口号
                        System.out.println("New connection accepted " +
socket.getInetAddress() + ":" + socket.getPort());

                        peopleList.put(name, this); // 将当前线程添加到散列表,
昵称作为关键字

                        Enumeration enum1 = peopleList.elements();

                        while (enum1.hasMoreElements()) // 获取所有的与客户端通
信的服务器线程
                        {
                            serverThread th = (serverThread)
enum1.nextElement();// 将当前聊天者的昵称和性别通知所有的用户
                            th.out.writeUTF("聊天者:" + name); // 也将其他聊天
者的姓名通知本线程（当前用户）
                            if (th != this) {
                                out.writeUTF("聊天者:" + th.name);
                            }
                        }

                    }
                }
                else if (s.startsWith("公共聊天内容:")){
                    String message = s.substring(s.indexOf(":") + 1);
                    Enumeration enum1 = peopleList.elements(); // 获取所有的与
客户端通信的服务器线程
                    while (enum1.hasMoreElements()) {
                        ((serverThread) enum1.nextElement()).out.writeUTF("公
```

```
共聊天内容:" + message);
                    }
                }
                else if (s.startsWith("用户离开:")){
                    socket.close();
                    System.out.println("未登录客户离开了");
                    break;
                }
                else if (s.startsWith("用户离线:")){
                    Enumeration enum1 = peopleList.elements(); // 获取所有的与
客户端通信的服务器线程
                    while (enum1.hasMoreElements()) // 通知其他聊天者，该用户已
离线
                    {
                        try {
                            serverThread th = (serverThread)
enum1.nextElement();
                            if (th != this && th.isAlive()) {
                                th.out.writeUTF("用户离线:" + name);
                            }
                        } catch (IOException eee) {
                            eee.printStackTrace();
                        }
                    }
                    if (name != null)
                        peopleList.remove(name);
                    socket.close(); // 关闭当前连接
                    System.out.println(name + "客户离开了");
                    break; // 结束本线程的工作，线程死亡
                }
            } catch (IOException e) {
                Enumeration enum1 = peopleList.elements(); // 获取所有的与客户
端通信的服务器线程

                while (enum1.hasMoreElements()) // 通知其他聊天者，该用户离线
                {
                    try {
                        serverThread th = (serverThread) enum1.nextElement();
                        if (th != this && th.isAlive()) {
                            th.out.writeUTF("用户离线:" + name);
                        }
                    } catch (IOException eee) {
                    }
                }
```

```
                if (name != null)
                    peopleList.remove(name);
                try // 关闭当前连接
                {
                    socket.close();
                } catch (IOException eee) {
                }
                System.out.println(name + "用户离开了");
                break; // 结束本线程的工作，线程死亡
            }/* finally {
                try {
                    if (socket != null)
                        socket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }*/
        }
        }
    }
```

创建和启动线程的代码如下：

```
Thread workThread = new serverThread(socket,peopleList);
//启动工作线程
workThread.start();
```

6 系统测试

6.1 用户登陆测试

测试用户登陆功能是否有效，且能够正常工作。由 3.5 节，登陆部分实现的功能包括：向服务器发送连接请求，接收服务器返回信息，启动主界面，启动服务器消息监听线程。

测试结果如下：

	出错提示	启动聊天界面	启动监听线程
服务器地址设置为不可达 IP	√		
用户登录名为空	√		
服务器 IP、登录名正常		√	√

使用重复登录名登录	√		
使用不同登录名登录		√	√

测试结果表明，登陆程序工作情况良好，达到了预期目标。

6.2 界面内容更新测试

界面内容更新既涉及到服务器端界面内容的更新，也涉及到客户端界面内容的更新。测试内容主要包括：有用户加入或退出时服务器端界面的信息更新是否正确对应，有用户加入或退出时客户端主界面是否与当前真实在线用户保持一致，用户进入聊天室或单人聊天界面聊天时，是否无遗漏地把客户信息展示在相应的位置上。

测试结果如下：

	出错提示	**消息提示**
一个用户登陆		√
一个用户退出		√
进入聊天室发送一条群发消息 （其他用户没有开启聊天室界面）		√
进入聊天室发送一条群发消息 （其他用户已经开启聊天室界面）		√

测试结果表明，各窗口程序展示内容更新正常，达到了预期目标。

7 总结

通过编写这个聊天程序，我基本掌握了使用 Java 语言进行 C/S 模式程序开发的方法。通过对 Socket 通信和多线程的代码编写，使我对这两种技术也有了更加深入的理解。

在实验所给的需求中对于 Socket 通信的基本代码已经给出，但只能实现特别简单的客户-服务器的通信功能，于是我自己打算写一个可以多人聊天的程序。通过对给出代码的研究，我了解了客户端和服务器之间交换信息的方式，在此基础上进行了相关的拓展，完善了多人聊天中客户端与服务器之间交换的信息。

在写程序的过程中，要将聊天界面做出来，这对于从未用 Java 写过界面的我真是一个挑战。在这种情况下，我查阅了相关的书籍，在网上搜索资料，对着实例分析学习，终于写出了一个虽然很简陋但是对于新手的我并不容易的聊天界面，各种组件无法很好控制各自的位置真真困扰了好久。这为我今后使用 Java 语言进行可视化程序的开发打下了一个比较好的基础。

这次的编程练习还让我对于线程有了更深刻的理解，以前上面向对象的课程时有关于线程的作业，但是当时我始终没有搞懂线程的写法和用法。这次的编写过程让我认识到线程的重要作用，也对于线程的使用有了初步的了解。

通过编写这个通信程序，使我从程序的级别了解了网络通信的基本方法。同时，希望我以上总结的内容能够对正在或将要进行 Socket 程序探索的人们提供帮助。