第22章

用 TCP 开发网络应用程序

从本章开始，将讲解网络编程，在网络编程框架内，我们主要针对比较重要的几种应用进行讲解，它们是：TCP 编程和 UDP 编程。

本章讲解 TCP 编程。TCP 编程是一种应用比较广泛的编程方式。我们将利用 TCP 编程，实现一个简单的聊天室。

本章术语

TCP _____

UDP _____

IP 地址 _____

端口 _____

ServerSocket _____

Socket _____

PrintStream _____

BufferedReader _____

22.1 认识网络编程

Note

22.1.1 什么是网络应用程序

在本书的前面几章中，我们的程序都是在一台单独的机器上运行，这一般称为单机版的软件。单机版软件不具备网络通信的功能；与此对应的是网络应用程序，能够通过网络，和另一台机器通信。比如下面的软件：

图 22-1

可以将另外机器上的一个文件，通过迅雷下载到自己的机器上。又如：

图 22-2

可以将一条聊天信息通过网络发到别人的机器。这些都属于网络应用程序。

本章讲解如何编写网络应用程序。

22.1.2 认识 IP 地址和端口

在编写网络应用程序之前，首先必须明白几个概念。

1. 通过什么来找到网络上的计算机。

要和别的机器通信，必须找到另一台机器在哪里。如何确定对方机器呢？很明显，用机器名称是不现实的，因为名称可能重复；实际上，我们是通过 IP 地址来确定一台计算机在网络上的位置的。

IP 地址被用来给 Internet 上的电脑一个编号。我们可以把一台机器比作一部手机，那么 IP 地址就相当于手机号码。

IP 地址是一个 32 位的二进制数，通常被分割为 4 个“8 位二进制数”。为了方便起见，IP 地址通常用“点分十进制”表示成（a.b.c.d）的形式，其中，a,b,c,d 都是 0~255 之间的十进制整数。比如，192.168.1.5，就是一个 IP 地址。

Note

问答

问：如何知道本机 IP 地址？

答：在 cmd 窗口中输入命令：ipconfig，即可显示 IP 地址：

图 22-3

或者右击“网上邻居”，选择“属性”，选取相应连接，右击，选择“属性”，出现如下界面：

图 22-4

双击“Internet 协议(TCP/IP)”，即可显示 IP 地址以及其他配置。

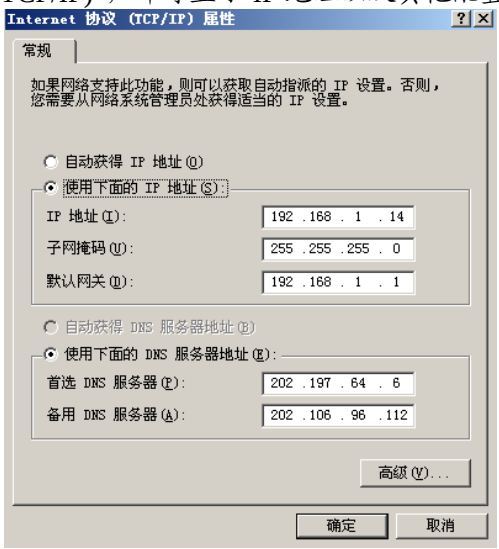

图 22-5

从上图可以看出，本机 IP 地址为 192.168.1.14，不过为了简便起见，统一可以用

127.0.0.1 表示本机 IP 地址。就好像每个人姓名不同，但是都可以自称“我”一样。

问：对方机器用 IP 地址确定，我们如何找到它？

答：某个 IP 地址的机器在网络的哪个地方，一般由路由器来判断，比如，我们要找 220.170.91.146，路由器会帮我们找到。具体找到过程，不是本课的内容。就好像我们要给对方打手机，不用关心移动公司怎样找到对方的一样。

Note

2. 通过什么来确定对方的网络通信程序。

找到计算机之后，就可以通信了吗？不一定。因为网络通信最终是软件之间的通信，还必须定位相应的软件。

我们首先来思考一个问题，一台联网的机器，只有一个网卡一根网线，为什么可以同时用多个程序上网？比如，我们可以用 FTP 下载文件，同时浏览网页，还可以 QQ 聊天，这些数据，是通过一根网线传过来的，为什么不会混淆呢？

实际上，我们是通过端口号(port)来确定一台计算机中特定的网络程序的。

我们可以将机器比作一栋办公楼，IP 就是这栋楼的地址，而端口就是办公楼中各个房间的房间号，虽然很多人都从大楼大门涌入，但是最后都进了不同的房间，每个房间负责完成不同的事情。

一台机器的端口号可以在 0-65535 之间。

这样，我们就可以理解，FTP 下载文件、浏览网页、QQ 聊天，这些程序应该对应不同的端口，信息传输到本机时，根据端口来进行分类，用不同的程序来处理数据。

问答

问：如何知道本机使用了哪些端口？

答：在 cmd 窗口中输入命令：netstat –an，即可显示本机使用了哪些端口：

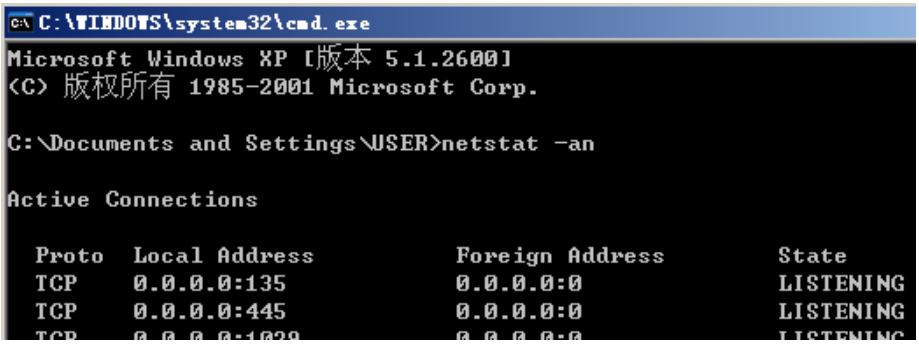

图 22-6

其中，IP 地址中，冒号后面你的数字(“0.0.0.0:135 中的 135”)，就是端口号。

问：常用应用程序的端口号有哪些？

答：21：FTP 协议；22：SSH 安全登录；23：Telnet；25：SMTP 协议；80：HTTP 协议。具体协议的意义，大家可以查看文档。

因此，在我们自己编写的网络应用程序中，要尽量避开这些常用的端口。一般情况下，0-1024 之间的端口最好不要使用。

3. 什么是 TCP？什么是 UDP？

TCP 和 UDP 是两种网络信息传输协议，都能够进行网络通信，你可以选择其中

的一种。

TCP 最重要的特点是面向连接，也就是说必须在服务器端和客户端连接上之后才能通信，它的安全性比较高。UDP 编程是面向非连接的，UDP 是数据报，只负责传输信息，并不能保证信息一定会被收到，虽然安全性不如 TCP，但是性能较好；TCP 基于连接，UDP 基于报文，具体大家可以参考计算机网络知识。

Note

你可以将 TCP 比喻成打电话，必须双方都拿起话机才能通话，并且连接要保持通畅。UDP 比喻成寄信，在寄信的时候，对方根本不知道有信要寄过去，信寄到哪里，靠信封上的地址。

没有必要讨论哪一种通信方式更好，这就像问打电话和寄信哪个好一样。

阶段性作业

上网搜索以下名词的全称：TCP、UDP、HTTP、FTP，它们有何区别？

22.1.3　客户端和服务器

客户端(Client)/服务器(Server)是一种最常见的网络应用程序的运行模式，简称 C/S。以网络聊天软件为例，在聊天程序中，各个聊天的界面叫做客户端，客户端之间如果要相互聊天，则可以将信息先发送到服务器端，然后由服务器端转发。因此，客户端先要连接到服务器端。

客户端连接到服务器端，需要知道一些什么信息呢？

显然，首先需要知道服务器端的 IP 地址，还要知道服务器端该程序的端口。如，知道服务器 IP 地址是 127.0.0.1，端口是 9999 等。

因此，服务器必须首先打开这个端口，等待客户端的连接，俗称打开并监听某个端口。

在客户端，必须要做到以下工作：根据服务器 IP，连接服务器的某个端口。

22.2　用客户端连到服务器

22.2.1　案例介绍

本节中我们开发一个聊天应用最基本的程序：客户端连接到服务器。首先，运行服务器，得到如图的界面：

图 22-7

Note

服务器运行完毕，界面上标题为：“服务器端，目前未见连接”。然后运行客户端，界面如图所示：

图 22-8

客户端运行完毕，界面上标题为：“客户端”。在上面有一个“连接”按钮，点击，连接到服务器端。服务器端界面变为如图所示的界面：

图 22-9

该界面上显示客户端的 IP 地址。

同时，客户端变为如图所示的界面，界面标题变为：“恭喜您，已经连上”：

图 22-10

22.2.2 如何实现客户端连接到服务器

前面说过，客户端连接到服务器端，首先需要知道服务器端的 IP 地址，还要知道服务器端该程序的端口。

服务器必须首先打开某个端口并监听，等待客户端的连接。客户端根据服务器 IP，连接服务器的某个端口。

本例中，服务器为本机，打开并监听的端口号是 9999。

1. 服务器端怎样打开并监听端口？

端口的监听是由 java.net.ServerSocket 进行管理的，打开 java.net.ServerSocket 的文档，这个类有很多构造函数，最常见的构造函数为：

public ServerSocket(int port) throws IOException

传入一个端口号，实例化 ServerSocket。

注意

实例化 ServerSocket，就已经打开了端口号并进行监听。

例如，如下代码就可以监听服务器上的 9999 端口，并返回 ServerSocket 对象 ss。

```
ServerSocket ss = new ServerSocket(9999);
```

2.客户端怎样连接到服务器端的某个端口？

客户端连接到服务器端的某个端口是由 java.net.Socket 进行管理的，打开 java.net.Socket 的文档，这个类有很多构造函数，最常见的构造函数为：

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

传入一个服务器 IP 地址和端口号，实例化 Socket。

注意

实例化 Socket，就已经请求连接到该 IP 地址对应的服务器。

例如，如下代码就可以连接服务器 218.197.118.80 上的 9999 端口，并返回连接 Socket 对象 socket。

```
Socket socket = new Socket("218.197.118.80",9999);
```

3. 服务器怎么知道客户端连上来了？

既然客户端用 Socket 来向服务器请求连接，如果连接上之后，Socket 对象自然成为连接的纽带。对于服务器端来说，就应该得到客户端的这个 Socket 对象，并以此为基础来进行通信。

怎样得到客户端的 Socket 对象？在前面的篇幅中我们知道，服务器端实例化 ServerSocket 对象，监听端口。打开 ServerSocket 文档，会发现里面有一个重要函数：

public Socket accept() throws IOException

该函数返回一个 Socket 对象，因此，在服务器端可以用如下代码得到客户端的 Socket 对象：

```
Socket socket = ss.accept();
```

注意

值得一提的是，accept 函数是一个“死等函数”，如果没有客户端请求连接，它会一直等待并阻塞程序。为了说明这个问题，我们编写如下代码进行测试：

AcceptTest.java

```
package chat1;
import java.net.ServerSocket;
import java.net.Socket;
public class AcceptTest {
    public static void main(String[] args) throws Exception {
        // 监听 9999 端口
        ServerSocket ss = new ServerSocket(9999);
        System.out.println("未连接");
        // 等待客户端连接,如果没有客户端连接，程序在这里阻塞
        Socket socket = ss.accept();
        System.out.println("连接");
```

```
    }
}
```

运行这个程序，控制台上打印：

```
未连接
```

图 22-11

Note

没有打印“连接”，说明程序在 accept 处阻塞。

当然，如果此时有另一个客户端进行连接，阻塞就可以解除：

AcceptTest_Client.java

```
package chat1;
import java.net.Socket;
public class AcceptTest_Client {
    public static void main(String[] args)throws Exception {
        Socket socket = new Socket("127.0.0.1",9999);
    }
}
```

运行客户端，服务器端打印：

```
未连接
连接
```

图 22-12

说明服务器阻塞被解除。

4. 如何从 Socket 得到一些连接的基本信息？

了解了客户端怎样连接到服务器端，很显然，客户端和服务器端用 Socket 对象来进行通信。那么，从 Socket 能否得到一些连接的基本信息呢？

打开 Socket 文档，会发现有如下函数：

```
public InetAddress getInetAddress()
```

返回 Socket 内连接客户端的地址。

该返回类型是 java.net.InetAddress，查找 java.net.InetAddress 文档，可以用以下方法得到 IP 地址：

```
public String getHostAddress()
```

返回 IP 地址字符串（以文本形式）。

22.2.3 代码编写

综合以上叙述，建立如下服务器端代码：

Server.java

```
package chat1;
import java.net.ServerSocket;
import java.net.Socket;
```

Note

```
import javax.swing.JFrame;
public class Server extends JFrame{
    private ServerSocket ss;
    private Socket socket;
    public Server(){
        super("服务器端，目前未见连接");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300,100);
        this.setVisible(true);
        try{
            //监听 9999 端口
            ss = new ServerSocket(9999);
            socket = ss.accept();
            String clientAddress = socket.getInetAddress().getHostAddress();
            this.setTitle("客户" + clientAddress + "连接");
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new Server();
    }
}
```

运行这个程序，就可以得到服务器的效果。

接下来是客户端程序，代码如下：

Client.java

```
package chat1;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.Socket;
import javax.swing.JButton;
import javax.swing.JFrame;
public class Client extends JFrame implements ActionListener{
    private JButton btConnect = new JButton("连接");
    private Socket socket;
    public Client(){
```

Note

```
        super("客户端");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.add(btConnect,BorderLayout.NORTH);
        btConnect.addActionListener(this);
        this.setSize(300,100);
        this.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        try{
            socket = new Socket("127.0.0.1",9999);
            this.setTitle("恭喜您，已经连上");
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new Client();
    }
}
```

运行，得到客户端界面，点击按钮，则可以连接到服务器。

注意

必须要先运行服务器端，再运行客户端。

阶段性作业

客户端连接到服务器，连接成功，双方的提示信息用消息框显示。

22.3 利用 TCP 实现双向聊天系统

22.3.1 案例介绍

上一节中已经讲述了客户端和服务器端的连接，接下来就可以让客户端和服务器端进行通信了。在本节中，服务器和客户端界面相同，都可以给对方发送信息，也能够自动收到对方发过来的信息。本节的效果如图所示：

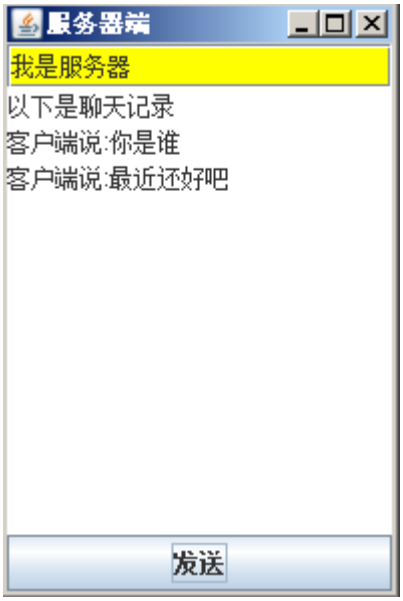

图 22-13

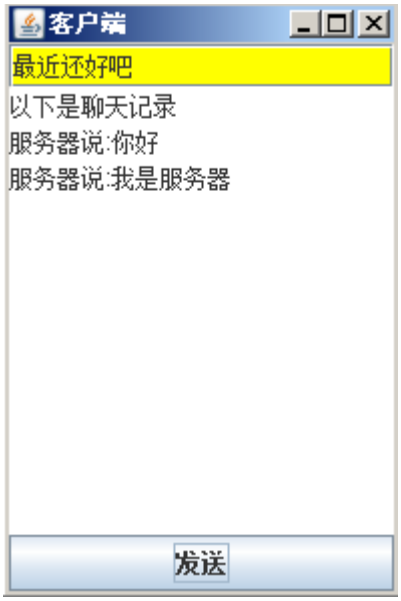

图 22-14

服务器端和客户端都有一个文本框，输入聊天信息。输入聊天信息之后，点击“发送”，就能够将信息发送给对方，对方也能够自动收到之后显示。

22.3.2　如何实现双向聊天

客户端与服务器端的通信过程，包括读信息和写信息，对于客户端和服务器端，如果将数据传给对方，就称为写，用到输出流；反之，如果从对方处得到数据，就为读，用到输入流。

TCP 编程中，客户端和服务器端之间的通信是通过 Socket 实现的。

1. 如何向对方发送信息？

从 java.net.Socket 文档中，会发现其中有一个重要函数：

public OutputStream getOutputStream() throws IOException

打开此 Socket 的输出流。

我们知道，OutputStream 功能并不强大，但是我们可以和 java.io.PrintStream 类配合使用，使之能够输出一行。如下代码：

```
Socket socket = new Socket("127.0.0.1",9999);
OutputStream os = socket.getOutputStream();
PrintStream ps = new PrintStream(os);
ps.println("消息内容");
```

就是用 Socket 向对方发出一个字符串。

2. 如何从对方处接收信息？

打开 java.net.Socket 文档，会发现其中有一个重要函数：

public InputStream getInputStream() throws IOException

打开此 Socket 的输入流。

我们知道，InputStream 功能并不强大，但是我们可以和 BufferedReader 函数配合使用，使之能够读取一行。如下代码：

Note

```
Socket socket = new Socket("127.0.0.1",9999);
InputStream is = socket.getInputStream(); // 得到输入流,InputStream 功能不强大
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String str = br.readLine();// 读
System.out.println(str);
```

就是从 Socket 的输入流中读入字符串，并打印。

很明显，在本例中，客户端和服务器端的通信，既要用到读操作，又要用到写操作。

为了对这个功能进行测试，我们在项目中建立一个服务器程序和客户端程序。让客户端发送给服务器端一个“服务器，你好”，然后服务器端收到之后打印。服务器端程序为：

Server.java

```
package chat2;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.InputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Server{
    public static void main(String[] args) throws Exception{
        ServerSocket ss = new ServerSocket(9999);
        Socket s = ss.accept();
        //获取对方传过来的信息，并打印
        InputStream is = s.getInputStream();
        BufferedReader br = new BufferedReader(new
InputStreamReader(is));
        String str = br.readLine();//读
        System.out.println(str);
    }
}
```

然后编写客户端，代码为：

Client.java

```
package chat2;
import java.net.Socket;
```

Note

```
import java.io.OutputStream;
import java.io.PrintStream;
public class Client{
    public static void main(String[] args) throws Exception{
        Socket s = new Socket("127.0.0.1",9999);//连接到服务器
        OutputStream os = s.getOutputStream();//os 只能发字节数组
        PrintStream ps = new PrintStream(os);    //ps 功能更强大
        ps.println("服务器，你好!");//信息发送出去
    }
}
```

首先运行服务器端，然后运行客户端，在服务器端的控制台上会打印：

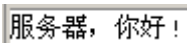

图 22-15

说明信息由客户端传输到了服务器端，并被服务器端收取。

注意

值得一提的是，在客户端与服务器端之间传递信息时，BufferedReader 的 readLine 函数也是一个“死等函数”，如果客户端连接上了，但是没有发送信息，readLine 函数会一直等待。为了说明这个问题，我们编写如下代码进行测试，服务器端代码为：

ReadLineTest.java

```
package chat2;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.InputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class ReadLineTest{
    public static void main(String[] args) throws Exception{
        ServerSocket ss = new ServerSocket(9999);
        Socket s = ss.accept();
        InputStream is = s.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(is));
        System.out.println("未收到信息");
        String str = br.readLine();//读
        System.out.println("收到信息");
        System.out.println(str);
    }
}
```

客户端代码为：

ReadLineTest_Client.java

```
package chat2;
import java.net.Socket;
public class ReadLineTest_Client {
    public static void main(String[] args)throws Exception {
        Socket socket = new Socket("127.0.0.1",9999);
    }
}
```

Note

运行服务器端，再运行客户端，服务器控制台上打印：

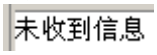

图 22-16

没有打印“收到信息”，说明程序在 readLine 处阻塞。

当然，如果客户端给服务器发送一条信息，阻塞就可以解除：

由以上情况可以看出，客户端和服务器端如果需要自动读取对方传来的信息，就不能将 readLine 函数放在主线程内，因为在不知道对方在什么时候会发出信息的情况下，readLine 函数的死等，可能会造成程序的阻塞。所以，最好的方法是将读取信息的代码写在线程内。

22.3.3 代码编写

综合以上叙述，建立如下服务器端代码：

Server.java

```
package chat3;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
public class Server extends JFrame implements ActionListener, Runnable {
    private JTextArea taMsg = new JTextArea("以下是聊天记录\n");
    private JTextField tfMsg = new JTextField("请您输入信息");
    private JButton btSend = new JButton("发送");
    private Socket s = null;
    public Server() {
        this.setTitle("服务器端");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Note

```
        this.add(taMsg,BorderLayout.CENTER);
        tfMsg.setBackground(Color.yellow);
        this.add(tfMsg,BorderLayout.NORTH);
        this.add(btSend,BorderLayout.SOUTH);
        btSend.addActionListener(this);
        this.setSize(200, 300);
        this.setVisible(true);
        try {
            ServerSocket ss = new ServerSocket(9999);
            s = ss.accept();
            new Thread(this).start();
        } catch (Exception ex) {
        }
    }
    public void run() {
        try {
            while (true) {
                InputStream is = s.getInputStream();
                BufferedReader br = new BufferedReader(
                        new InputStreamReader(is));
                String str = br.readLine();// 读
                taMsg.append(str + "\n");// 添加内容
            }
        } catch (Exception ex) {
        }
    }
    public void actionPerformed(ActionEvent e) {
        try {
            OutputStream os = s.getOutputStream();
            PrintStream ps = new PrintStream(os);
            ps.println("服务器说:"+tfMsg.getText());
        } catch (Exception ex) {
        }
    }
    public static void main(String[] args) throws Exception {
        Server server5 = new Server();
    }
```

```
}
```

运行这个程序，就可以得到服务器的效果。

接下来是客户端程序，代码如下：

Client.java

```
package chat3;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
public class Client extends JFrame implements ActionListener,Runnable{
    private JTextArea taMsg = new JTextArea("以下是聊天记录\n");
    private JTextField tfMsg = new JTextField("请您输入信息");
    private JButton btSend = new JButton("发送");
    private Socket s = null;
    public Client(){
        this.setTitle("客户端");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.add(taMsg,BorderLayout.CENTER);
        tfMsg.setBackground(Color.yellow);
        this.add(tfMsg,BorderLayout.NORTH);
        this.add(btSend,BorderLayout.SOUTH);
        btSend.addActionListener(this);
        this.setSize(200, 300);
        this.setVisible(true);
        try{
            s = new Socket("127.0.0.1",9999);
            new Thread(this).start();
        }catch(Exception ex){}
    }
    public void run(){
        try{
            while(true){
                InputStream is = s.getInputStream();
                BufferedReader br = new BufferedReader(
                        new InputStreamReader(is));
                String str = br.readLine();//读
```

Note

```
                    taMsg.append(str+"\n");//添加内容
                }
            }catch(Exception ex){}
        }
        public void actionPerformed(ActionEvent e){
            try{
                OutputStream os = s.getOutputStream();
                PrintStream ps = new PrintStream(os);
                ps.println("客户端说:"+tfMsg.getText());
            }catch(Exception ex){}
        }
        public static void main(String[] args) throws Exception{
            Client client5 = new Client();
        }
    }
```

运行，得到客户端界面，两者即可进行聊天。

注意

必须要先运行服务器端，再运行客户端。

阶段性作业

1. 将本例中的按钮去掉，改为：在文本框中回车，信息自动发出，文本框清空。

2. 完成一个网络远程控制系统，如果服务器给客户端发送的信息为：“关闭”二字，客户端能够自动关闭。

3. 完成一个简单的隐私窃取软件，如果客户端连到服务器，能够自动将其 C 盘下的所有文件名称传到服务器端显示。

22.4　利用 TCP 实现多客户聊天系统

22.4.1　案例介绍

上一节中已经讲述了客户端和服务器端的互相通信。但是，实际应用中，应该是客户端和客户端聊天，而不是客户端和服务器端聊天。客户端和客户端聊天的本质是信息由服务器端转发。因此，本节将开发一个支持多个客户端的程序。服务器端界面如图所示：

图 22-17

Note

以下是客户端界面，当客户端出现时，需要输入昵称：

图 22-18

点击“确定”，连接到服务器。如果连接成功，服务器回送一个信息：

图 22-19

确定，即可进行聊天。

为了体现多客户端效果，我们打开了 3 个客户端，如图所示：

图 22-20

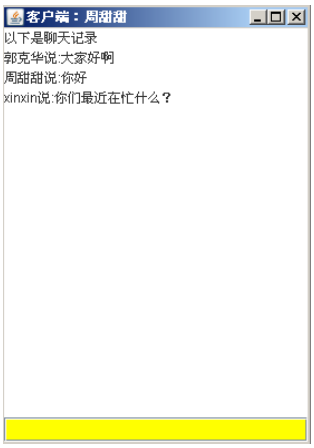

图 22-21

图 22-22

在界面下方可以输入消息，回车，消息发出。消息发出之后，文本框自动清空。消息发送之后，能够让各个客户端都收到聊天信息，聊天信息打印在界面上的多行文本框内，在打印聊天信息的同时，还能够打印这条聊天信息是谁说的。

22.4.2 编写服务器程序

在本例中，要让服务器端能够接受多个客户端的连接，需要注意以下几个问题：

1. 由于事先不知道客户端什么时候连过来，因此，服务器端必须首先有一个线程负责接受多个客户端连接；结构如下：

```
public class Server extends JFrame implements Runnable{
    public Server(){
        //服务器端打开端口
        //服务端开启线程，接受客户端连接
    }
    public void run(){
        //不断接受客户端连接
        while(true){
                //接收客户端连接
                 //开一个聊天线程给这个客户端
                //将该聊天线程对象添加进集合
               //聊天线程启动
        }
    }
}
```

Note

2．当客户端连接上之后，服务器端要等待这些客户端传送信息过来，而事先并不知道客户端什么时候会发信息过来。所以，每一个客户端连上之后，必须为这个客户端单独开一个线程，来读取它发过来的信息。因此，需要再编写一个线程类。

3. 服务器收到某个客户端信息之后，需要将其转发给各个客户端，这就需要在服务器端保存各个客户端的输入输出流的引用(实际上，这些引用可以保存在为客户端服务的线程中)。

因此，整个服务器端程序的基本结构如下：

```
public class Server extends JFrame implements Runnable{
    public Server(){
        //服务器端打开端口
        //服务端开启线程，接受客户端连接
    }
    public void run(){
        //不断接受客户端连接
        while(true){
                //接收客户端连接
                //开一个聊天线程给这个客户端
                //将该聊天线程对象添加进集合
                //聊天线程启动
        }
```

Note

```
        }
        /*聊天线程类，每连接上一个客户端，就为它开一个聊天线程*/
        class ChatThread extends Thread{
            //负责读取相应 SocketConnection 的信息
            public void run(){
                while(true){
                    //读取客户端发来的信息
                    //将该信息发送给所有其他客户端
                }
            }
        }
}
```

服务器端详细代码如下：

Server.java

```
package chat4;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import javax.swing.JFrame;
public class Server extends JFrame implements Runnable{
    private Socket s = null;
    private ServerSocket ss = null;
    private ArrayList clients = new ArrayList();//保存客户端的线程
    public Server() throws Exception{
        this.setTitle("服务器端");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBackground(Color.yellow);
        this.setSize(200,100);
        this.setVisible(true);
        ss = new ServerSocket(9999);//服务器端开辟端口，接受连接
        new Thread(this).start(); //接受客户连接的死循环开始运行
    }
    public void run(){
        try{
            while(true){
                s = ss.accept();
```

Note

```
                //s 就是当前的连接对应的 Socket，对应一个客户端
                //该客户端随时可能发信息过来，必须要接收
                //另外开辟一个线程，专门为这个 s 服务，负责接受信息
                ChatThread ct = new ChatThread(s);
                clients.add(ct);
                ct.start();
            }
        }catch(Exception ex){}
    }
    class ChatThread extends Thread{//为某个 Socket 负责接受信息
        private Socket s = null;
        private BufferedReader br = null;
        public PrintStream ps = null;
        public ChatThread(Socket s) throws Exception    {
            this.s = s;
            br = new BufferedReader(
                    new InputStreamReader(s.getInputStream()));
            ps = new PrintStream(s.getOutputStream());
        }
        public void run(){
            try{
                while(true){
                    String str = br.readLine();//读取该 Socket 传来的信息
                    sendMessage(str);  //将 str 转发给所有客户端
                }
            }catch(Exception ex){}
        }
    }
    public void sendMessage(String msg){//将信息发给所有客户端
        for(int i=0;i<clients.size();i++){
            ChatThread ct = (ChatThread)clients.get(i);
            //向 ct 内的 Socket 内写 msg
            ct.ps.println(msg);
        }
    }
    public static void main(String[] args) throws Exception{
        Server server = new Server();
```

```
        }
    }
```

Note

22.4.3 编写客户端程序

客户端编程相对简单，只需要编写发送信息、连接服务器、接受服务器端传输的信息即可。代码如下：

Client.java

```
package chat4;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import javax.swing.*;
public class Client extends JFrame implements ActionListener, Runnable {
    private JTextArea taMsg = new JTextArea("以下是聊天记录\n");
    private JTextField tfMsg = new JTextField();
    private Socket s = null;
    private String nickName = null;
    public Client() {
        this.setTitle("客户端");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.add(taMsg,BorderLayout.CENTER);
        tfMsg.setBackground(Color.yellow);
        this.add(tfMsg,BorderLayout.SOUTH);
        tfMsg.addActionListener(this);
        this.setSize(280, 400);
        this.setVisible(true);
        nickName = JOptionPane.showInputDialog("输入昵称");
        try {
            s = new Socket("127.0.0.1", 9999);
            JOptionPane.showMessageDialog(this,"连接成功");
            this.setTitle("客户端：" + nickName);
            new Thread(this).start();
        } catch (Exception ex) {}
    }
    public void run() {
        try {
```

Note

```
                while (true) {
                    InputStream is = s.getInputStream();
                    BufferedReader br = new BufferedReader(
                            new InputStreamReader(is));
                    String str = br.readLine();// 读
                    taMsg.append(str + "\n");// 添加内容
                }
            } catch (Exception ex) {
            }
        }
        public void actionPerformed(ActionEvent e) {
            try {
                OutputStream os = s.getOutputStream();
                PrintStream ps = new PrintStream(os);
                ps.println(nickName + "说:" + tfMsg.getText());
                tfMsg.setText("");
            } catch (Exception ex) {
            }
        }
        public static void main(String[] args) throws Exception {
            Client client = new Client();
        }
    }
```

运行服务器端和客户端，就可以得到本案例需求中的效果。

阶段性作业

1. 在客户端增加一个下拉列表框，显示每个在线客户的昵称，如果某个客户下线，可以通知其他客户进行刷新，如何实现？

2. 客户可以在下拉列表中选择自己要发送信息的人，进行私聊，如何实现？

本章知识体系

知识点	重要等级	难度等级
网络编程若干概念	★★★★	★★
ServerSocket	★★★	★★
Socket	★★★★	★★★
客户端连到服务器	★★★	★★

客户端和服务器通信	★★★★	★★★
基于线程双向通信	★★★	★★★★
多客户端	★★★	★★★★★

Note