

多客户端聊天服务器

1、实验目的

实现一个多客户端的纯文本聊天服务器，能同时接受多个客户端的连接，并将任意一个客户端发送的文本向所有客户端（包括发送方）转发。

2、实验环境

系统	Ubuntu 20.04 on Windows 11
内核版本	5.10.60.1-microsoft-standard-WSL2
处理器	AMD Ryzen 5 5600G with Radeon
内存	32.0 GB
java版本	java 11 2018-09-25
jre版本	18.9 (build 11+28)

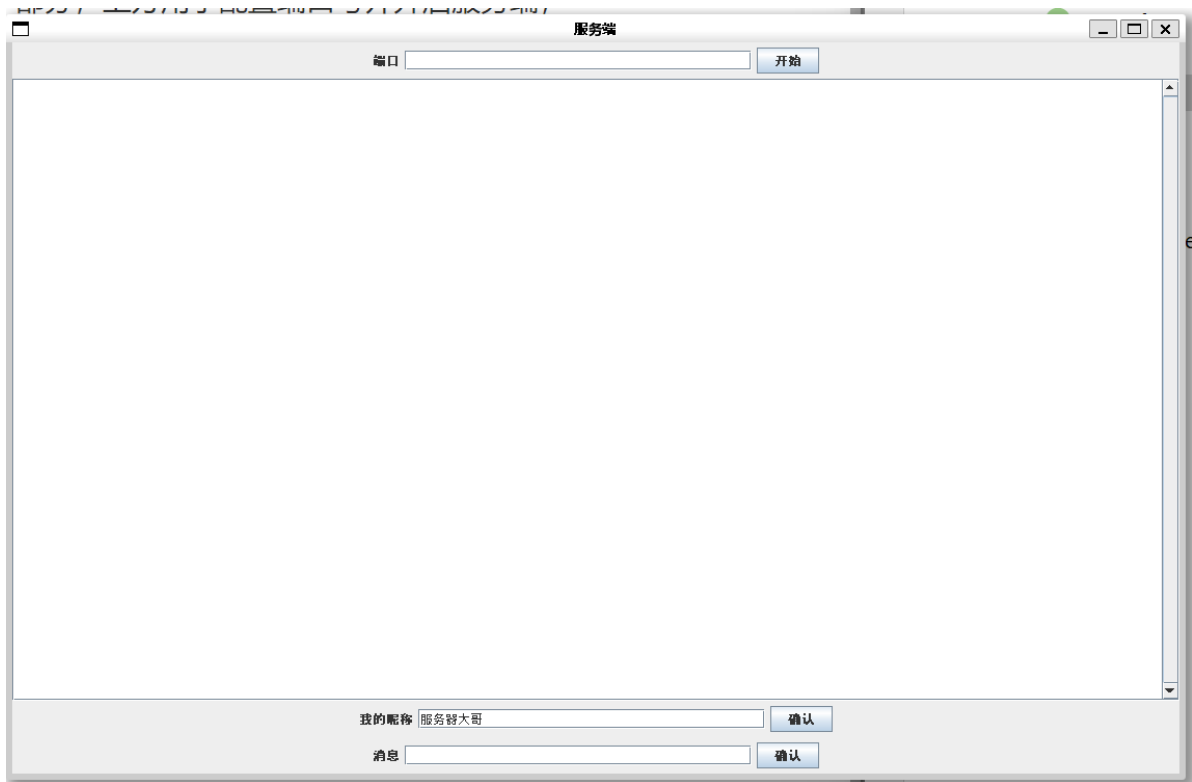
3、实验设计

3.1 功能介绍

本项目实现了多客户端的纯文本聊天服务器，能够同时接受多个客户端的连接，并将任意一个客户端发送的文本向所有客户端（包括发送方）转发，同时也实现了服务端的GUI窗口，用于清晰的配置服务端的端口，名称以及输入输出情况。

3.2 界面结构

在这里我们只实现了服务端的代码，功能包括开始连接，设置服务端名称以及给所有客户端发送消息。服务端整体GUI布局我们使用了Swing中的 `BorderLayout`，分为上中下三个部分，上方用于配置端口号并开启服务端，下方用于配置服务端的名字以及向所有客户端发送消息，中间部分是一个可以滚动的区域，用于显示所有的消息以及各种信息。整体布局如下所示



这样消息展示的也比较直观，同时整体布局简洁。具体代码不在这里进行展示。

3.3 功能设计

本次设计中我们将Server的逻辑和GUI的逻辑分割开来，这也是本次设计的亮点之一，我们需要额外考虑GUI和server两部分的通信与数据共享的方法，我们当然可以**简单粗暴的将 Server 直接放到 GUI 内部**，但是！这样会导致高度的耦合以及大量代码杂糅，无法方便的开发和维护，因此我们设计并实现了 `ServerIntf` 和 `GUIIntf` 两个接口，这两个组件分别实现该接口，同时拥有对方接口的一个实例（即接口的实例互相设置为对方的实例即可，两者公共父组件实现时进行设置），这样可以很方便的解耦合，如下是两个接口的设计：

ServerIntf:

```
public interface ServerIntf {
    //开始监听
    void onStart(int port);
    //修改nickname
    void onChangeNickName(String newName);
    //发送信息
    void onSendMsg(String msg);
}
```

GUIIntf:

```
public interface GUIIntf {
    //展示区中增加消息
    void appendMessage(String msg);
    //展示弹窗
    void showMessageDialog(String msg);
}
```

因此两者可以方便的进行数据共享

在服务端的设计上，我们使用了 线程池 + 消息队列 + bufferedIo 的设计模式，1个线程用于绑定 socket，1个线程用于消息中心不停的向所有客户端发送消息，剩下n个线程用于接受客户端的消息并处理。

- 线程池的设计上我们使用了java自带的 `ThreadPoolExecutor`，配置参数为

```
threadPool= new ThreadPoolExecutor(0,MaxThread,60L, TimeUnit.SECONDS,
                                     new
    SynchronousQueue<Runnable>());
```

这里我使用2000作为最大的连接数，最多可以同时接受处理1998个连接。

每次需要新的线程直接使用下方的代码提交一个新任务即可

```
threadPool.submit(new Runnable())
```

- 消息队列的设计上我们使用了synchronized+Queue的形式，首先建立发送中心类，之后每个 Client类都将自己注册到该中心中，发送线程使用 `this.wait()` 进行等待，而需要发送的消息则通过使用消息中心的 `sendToClient`，具体实现如下，可以到目前还是非常简陋的水平，但是基本实现了需要的功能。

需要注意的是，该类中基本所有的方法都必须使用synchronized进行同步

```
public void sendToClient(String msg){
    synchronized (this){
        messageQueue.add(msg);
        //唤醒等待的线程
        this.notify();
    }
}

@Override
public void run() {
    while(true){
        synchronized (this){
            try {
                //还有没有发的数据
                while (!messageQueue.isEmpty()){
                    String msg=messageQueue.peek();
                    messageQueue.remove();
                    sends.forEach(r->{
                        try {
                            r.sendMessage(msg);
                        }
                    });
                }
            }
        }
    }
}
```

```

        } catch (Exception ex){
            ex.printStackTrace();
        }
    });
}
//交出锁, 进行等待
this.wait();
} catch (InterruptedException e ){
    e.printStackTrace();
}
}
}
}
}

```

- 收发消息上，我们分别使用了 `PrintStream` 和 `BufferedReader` 进行消息的收发，在 `Client` 类创建的时候，使用传入的 `socket` 来建立发送和接受实例，并将发送实例 `printStream` 注册到消息中心中，之后开始监听并接收 `BufferedReader` 的消息，并在合适的时候（收到 "bye" 或者客户端断开连接）主动断开和 `client` 的连接，同时将消息中心中的注册删除即可。这里的实现比较简单，没有考虑实现更为复杂的 NIO

```

//客户端类, 实现了SendMsgIntf接口用来向客户端发送数据
public class Client implements Runnable, SendMsgIntf{
    //原始的socket
    Socket socket;
    //接收
    BufferedReader bufferedReader;
    //发送
    PrintStream printStream;
    //是否断开
    Boolean disconnect;
    //初始化各种变量
    public Client(Socket s) {
        this.socket = s;
        disconnect=false;
        try {
            bufferedReader=new BufferedReader(new
InputStreamReader(s.getInputStream()));
            printStream=new PrintStream(s.getOutputStream());
            guiIntf.appendMessage(String.format("%s connect to
Server!\n",getFullInfo()));
            sendCenter.addSend(this);

            printStream.println("Hello world!");
            printStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public String getFullInfo(){
        if(socket==null){
            return "连接为空";
        }
        return String.format("[remote]IP:%s,Port:%d [server]IP:%s,
Port:%d",socket.getInetAddress().toString(),socket.getPort(),socket.getLocal
Address(),socket.getLocalPort());
    }
}

```

```

        public String getRemoteInfo(){
            if(socket==null){
                return "连接为空";
            }
            return String.format("
[remote]IP:%s,Port:%d",socket.getInetAddress().toString(),socket.getPort());
        }
        @Override
        public void run() {
            //使用bufferedReader读取数据
            String newMessage;
            try {
                while ((newMessage=bufferedReader.readLine())!=null){
                    if(newMessage.equals(END)){
                        break;
                    }
                    guiIntf.appendMessage(String.format("New message
%s\n%s\n", getFullInfo(), newMessage));
                    sendCenter.sendToClient(String.format("New message
%s\n%s\n", getRemoteInfo(), newMessage));
                }
                //连接断开, 客户端主动
                this.disconnect=true;
                sendCenter.disconnect();
                guiIntf.appendMessage(String.format("Disconnect
%s\n",getFullInfo()));
                //关闭连接
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        @Override
        public void sendMsg(String msg) {
            printStream.println(msg);
            printStream.flush();
        }

        @Override
        public Boolean getStatus() {
            return disconnect;
        }
    }
}

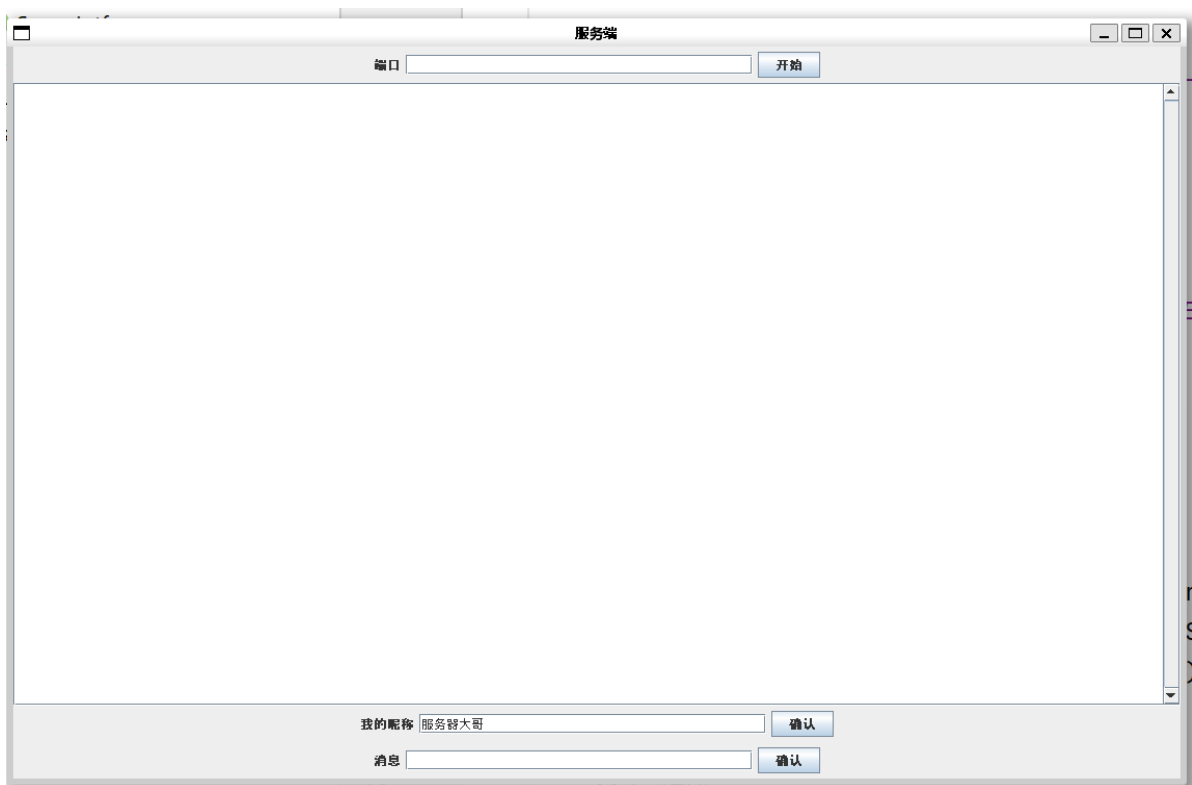
```

主函数也非常简单，我们分别New出来GUI和Server得类，并且将这两个类分别加入对方的类中去。

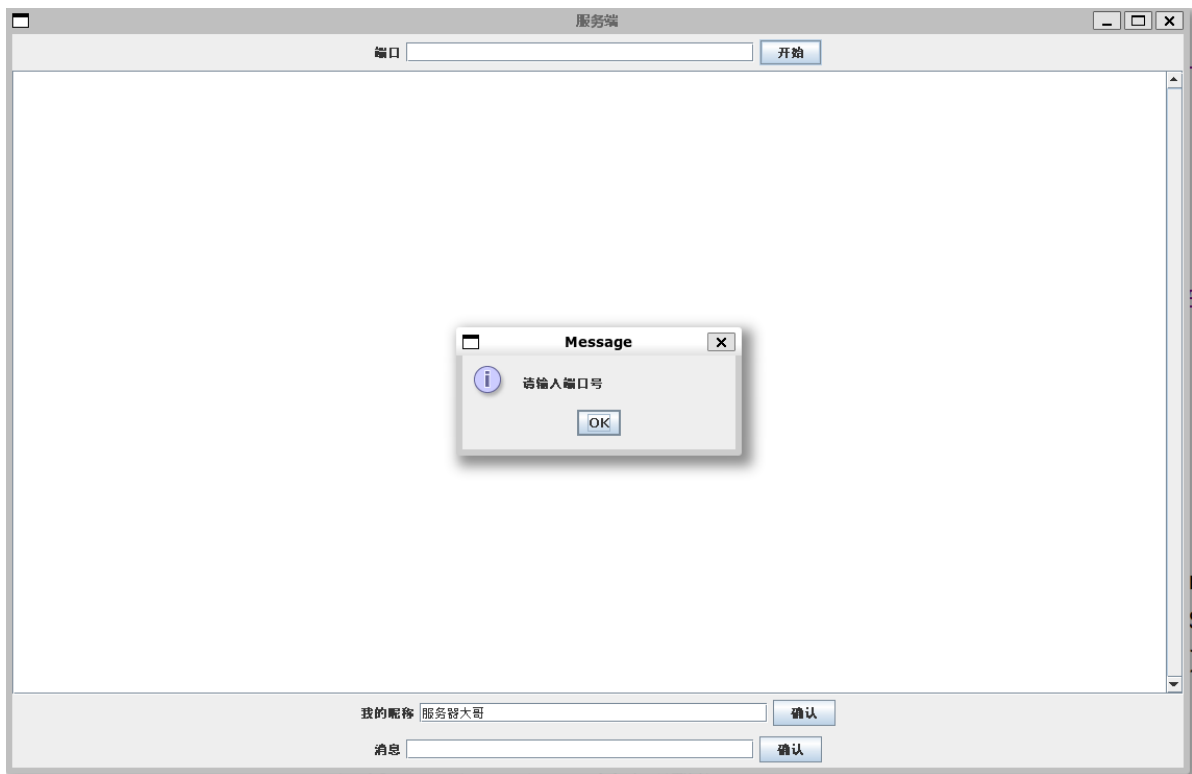
```
public class Main {  
  
    public static void main(String[] args) {  
        Gui gui=new Gui();  
        Server server=new Server();  
        gui.setServerIntf(server);  
        server.setGuiIntf(gui);  
    }  
}
```

4 功能展示和分析

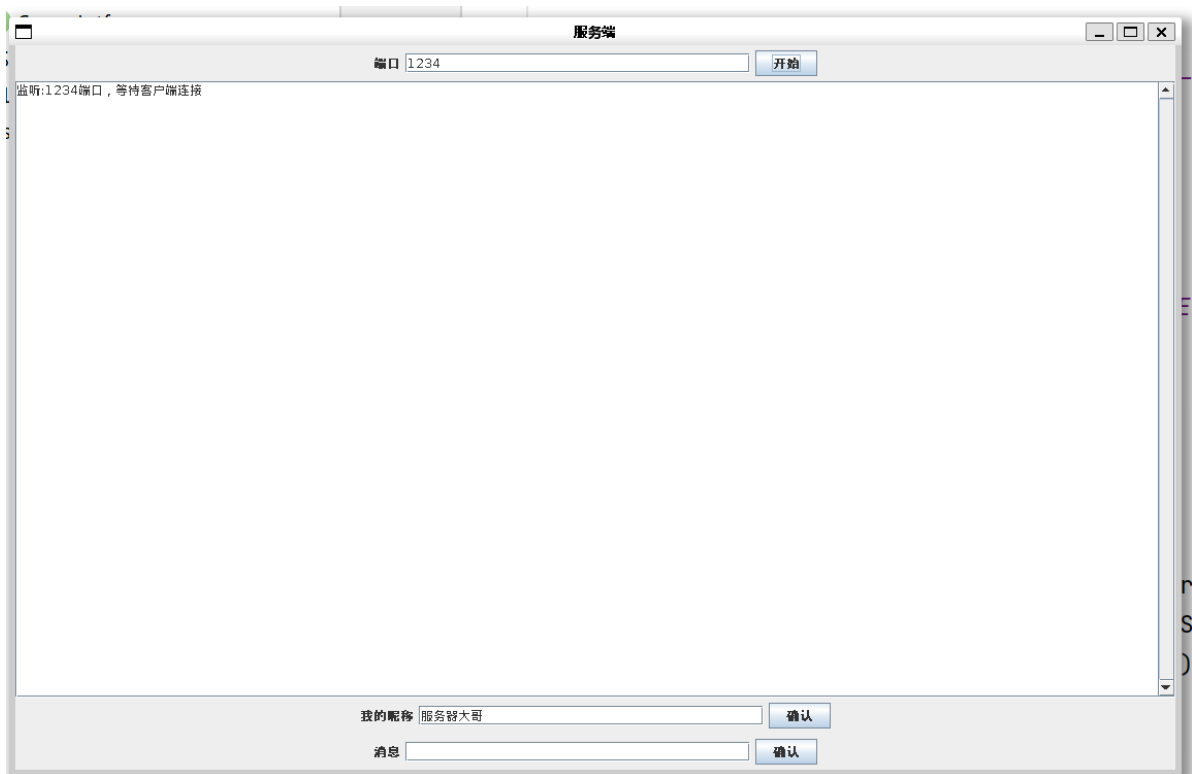
首先是服务端的启动，下图为初始页面



未输入端口号：



输入1234并点击确定，可以看到已经开始监听1234端口

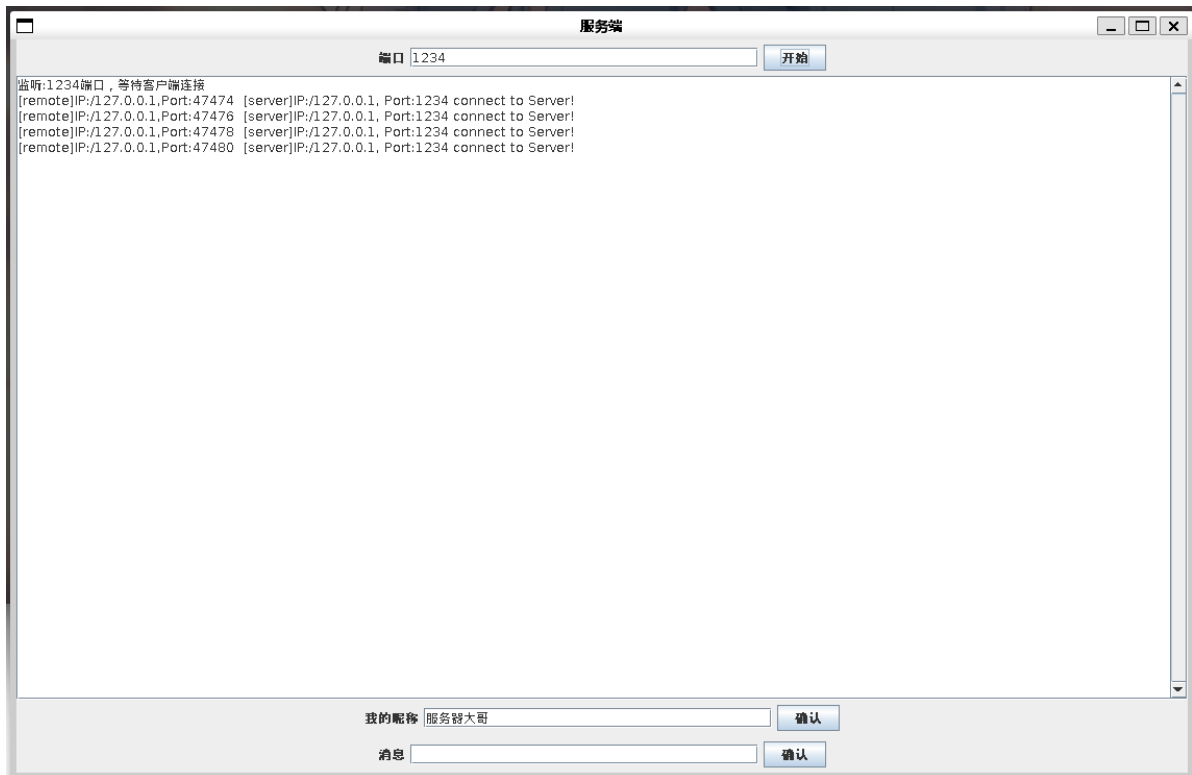


这里我开启4个终端，都使用telnet 127.0.0.1 1234 进行连接

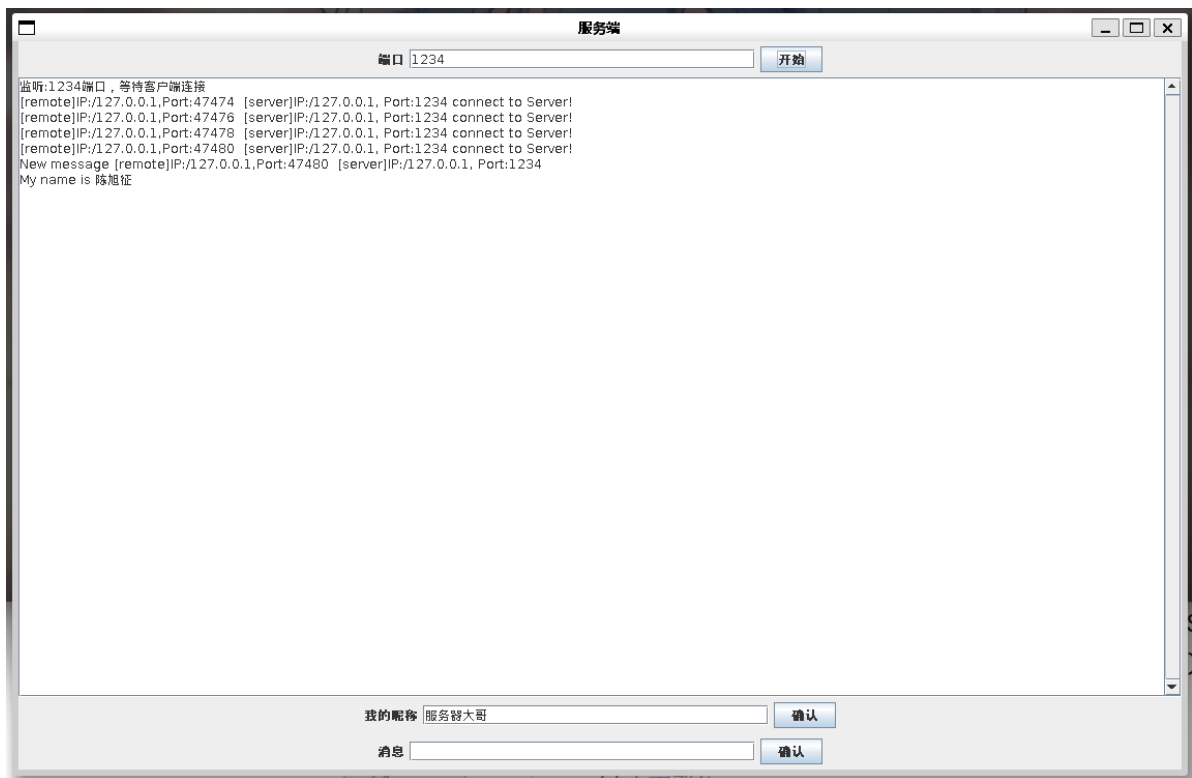
客户端：

```
> telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello World!
|
```

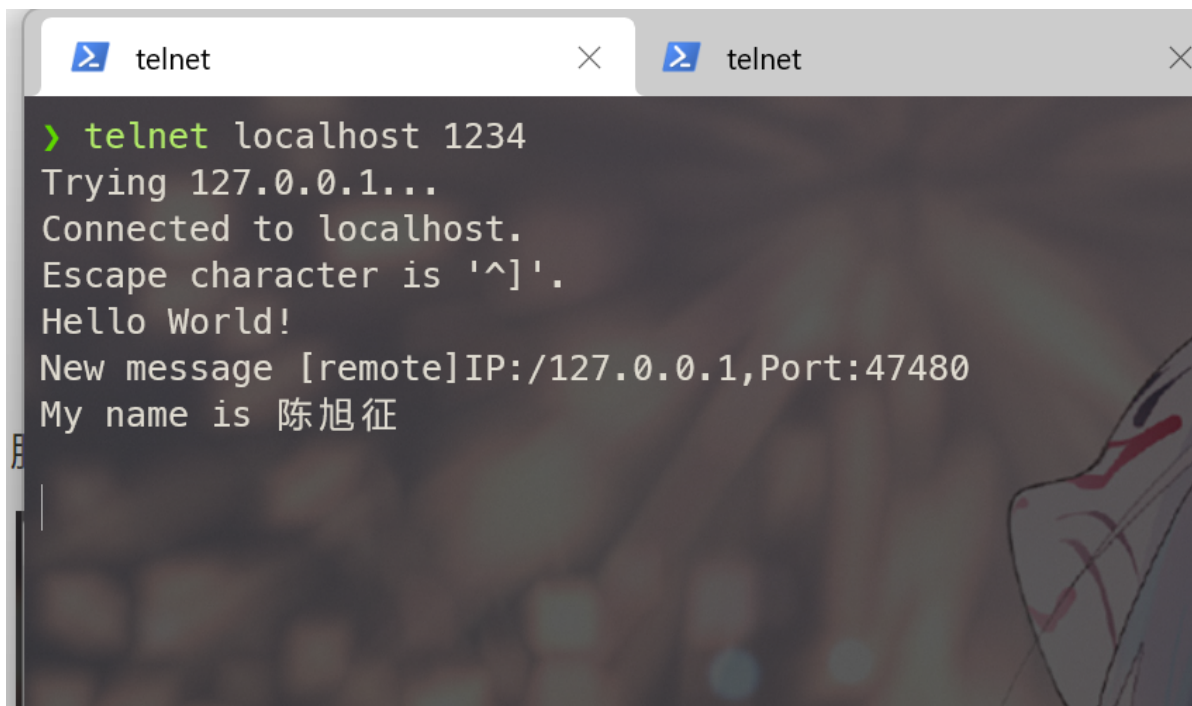
服务端：



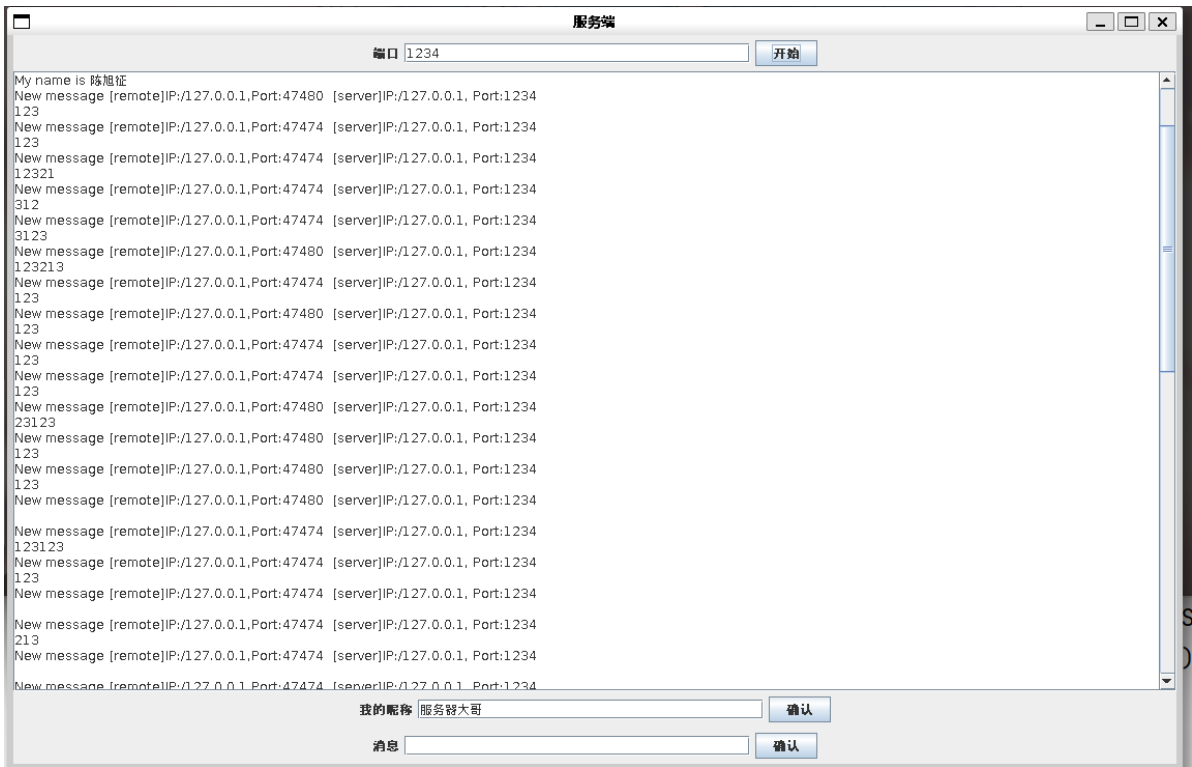
客户端之间发送消息：

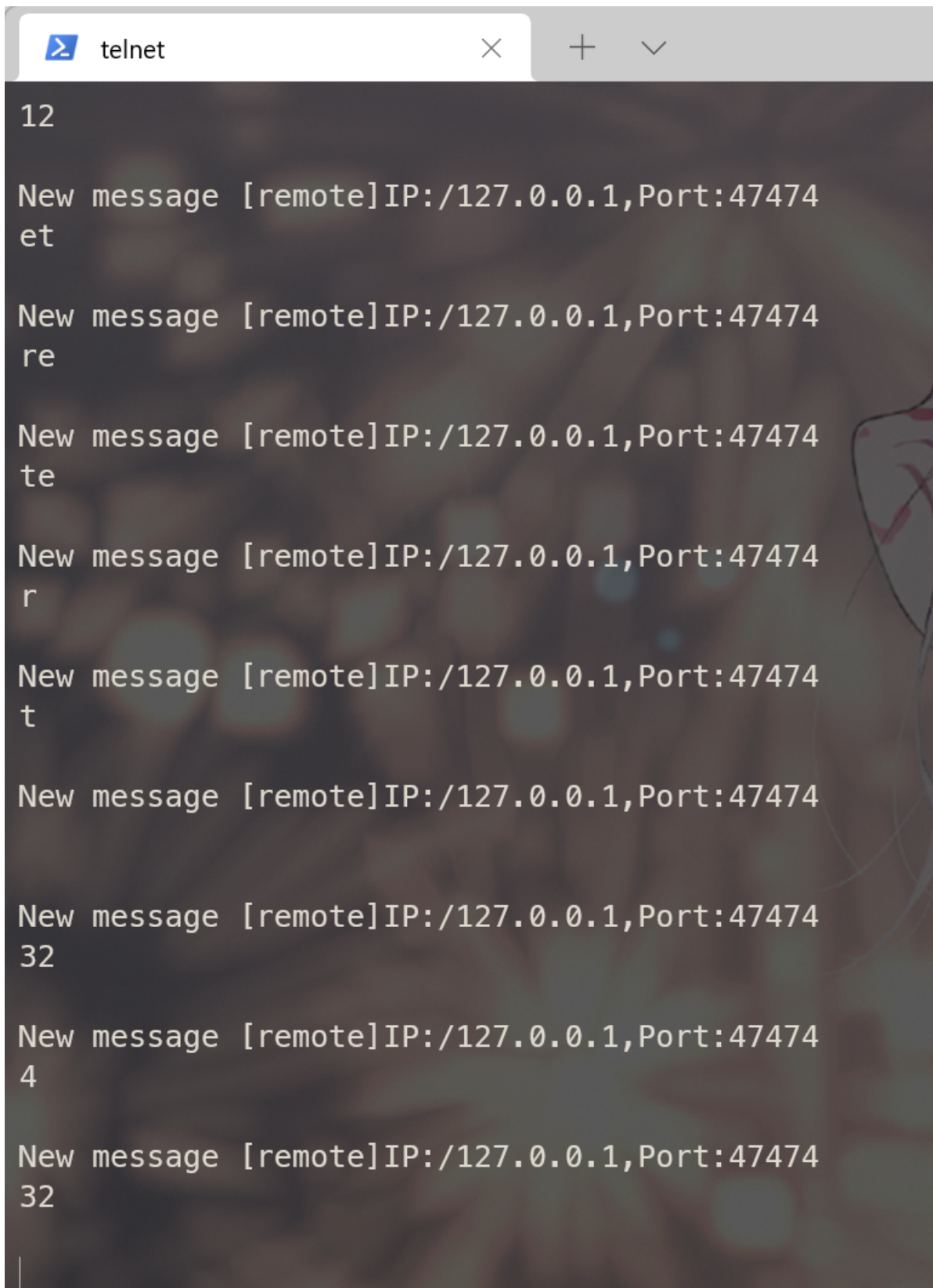


其他客户端上也收到了:



大量消息发送测试:



A screenshot of a telnet terminal window. The window has a title bar with a blue icon, the text 'telnet', and standard window controls (close, maximize, minimize). The terminal content shows a series of messages received from a remote IP address. The messages are: '12', 'New message [remote]IP:/127.0.0.1,Port:47474 et', 'New message [remote]IP:/127.0.0.1,Port:47474 re', 'New message [remote]IP:/127.0.0.1,Port:47474 te', 'New message [remote]IP:/127.0.0.1,Port:47474 r', 'New message [remote]IP:/127.0.0.1,Port:47474 t', 'New message [remote]IP:/127.0.0.1,Port:47474', 'New message [remote]IP:/127.0.0.1,Port:47474 32', 'New message [remote]IP:/127.0.0.1,Port:47474 4', and 'New message [remote]IP:/127.0.0.1,Port:47474 32'. The background of the terminal window is dark with a faint, stylized illustration of a person's face on the right side.

```
telnet

12

New message [remote]IP:/127.0.0.1,Port:47474
et

New message [remote]IP:/127.0.0.1,Port:47474
re

New message [remote]IP:/127.0.0.1,Port:47474
te

New message [remote]IP:/127.0.0.1,Port:47474
r

New message [remote]IP:/127.0.0.1,Port:47474
t

New message [remote]IP:/127.0.0.1,Port:47474

New message [remote]IP:/127.0.0.1,Port:47474
32

New message [remote]IP:/127.0.0.1,Port:47474
4

New message [remote]IP:/127.0.0.1,Port:47474
32

|
```

可以看到都完全正常。

服务端给用户端发送消息，可以看到接受正常：

```
New message [remote]IP:/127.0.0.1,Port:47474
32

New message [remote]IP:/127.0.0.1,Port:47474
4

New message [remote]IP:/127.0.0.1,Port:47474
32

Server 服务器大哥 send new message
Hello I am Server
```

客户端断开连接，输入bye或直接关闭terminal

输入bye:

```
32

Server 服务器大哥 send new message
Hello I am Server

bye
Connection closed by foreign host.

/mnt/c/U/ChenXuzheng
```

手动关闭terminal，可以看到服务端有相应的响应，因此处理错误正确:

```
4
New message [remote]IP:/127.0.0.1,Port:47474 [server]IP:/127.0.0.1, Port:1234
32
Server 服务器大哥 send new message
Hello I am Server
Disconnect [remote]IP:/127.0.0.1,Port:47474 [server]IP:/127.0.0.1, Port:1234
Disconnect [remote]IP:/127.0.0.1,Port:47476 [server]IP:/127.0.0.1, Port:1234
```

5.实验心得

由于实验2已经完成并使用了java的Swing GUI，此次实验再使用起来更加得心应手，同时使用了GUI和Server分离的设计模式，很方便的进行了代码解耦合，写起来也更加直观。

本次实验的难点在于更优雅的设计模式以及消息队列、线程池的使用，我都搜索了一些资料并按照自己的理解进行了简单的实现与使用，最终实现了一个比较优雅的服务端，但是很可惜的是由于时间原因没有将客户端的GUI一起实现了，这也是今后改进的一个方向。

值得一提的是我计网课LAB上也实现了一个更加多功能的[C/S消息服务器](#)，因此本次实验有一定的基础，做起来简单了不少，但是相比于计网课上的，这个更加不够规范，还存在很多可以优化的点，之后有时间会进一步改进。