Lab3：纯本文聊天服务器

0. 报告说明

由于课程要求中，得分项是针对压力测试，并且只针对服务器的，只需要提交服务端程序。

我之前实现的聊天室是做了服务器端和客户端，在客户端实现了一个图形化界面，较为直观，但是服务器端则只是简单为每一个客户端程序连接建立了一个线程，较为简便。为了能针对测试要求，我重写了一份利用线程池并且加上了安全性互斥访问的服务端程序。

1. 实验目的

- 学习Java的多线程编程，能够熟练写出多线程的程序，并且了解异常抛出的语法
- 学习Java的Socket编程的相关接口

2. 实验环境

Java版本：1.8.0_351

实验环境：Windows

3. 实验原理

3.1 实验设计

实现一个公共聊天室，聊天室内，每个用户可以发送消息，并接收其他用户的消息。服务器实现的功能为，接收客户端的连接，并且将客户端发送的消息转发给每个连接到自己的客户端。

3.2 Server设计

功能设计

- 利用线程池，对每一个请求连接的客户端进行处理
- 接收客户端发来的信息，并将此信息转发给每一个连接上的客户端程序

接收连接

对于客户端发来的请求，服务端将会循环不断的利用accept进行阻塞监听。当监听到时，创建一个线程进行管理，并放入线程池之中

核心代码设计如下：

```
    public void start(){
        try{
            //循环监听客户端的连接
            while(true){
                System.out.println("等待客户端连接...");
                Socket socket = serverSocket.accept();
                SocketAddress remoteSocketAddress =
socket.getRemoteSocketAddress();
                System.out.println("客户端"+remoteSocketAddress+"已连接！");
                ServerHandler handler = new ServerHandler(socket);
```

```
                //启动一个线程来完成针对该客户端的交互
                threadPool.execute(handler);
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

线程处理

对于每一个对客户端进行线程管理时，主要处理的是为每一个线程初始化的发送信息Printwriter和读取其他线程发送信息的BufferReader。接着，如果监听到客户端进行了消息写入，便将此消息群发给所有的writer。

```
        public void run() {
            PrintWriter writer=null;
            try {
                //初始化进行对客户端写入的writer
                writer = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream(), "UTF-8"), true);
                //安全添加到writers中
                addWriter(writer);
                //初始化进行对客户端读入的reader
                BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream(), "UTF-8"));
                String message = null;
                while((message = bufferedReader.readLine()) != null){
                    SocketAddress remoteSocketAddress =
socket.getRemoteSocketAddress();
                    sendMessage(remoteSocketAddress.toString() + ":" + message);
                }
            }catch (Exception e) {
                e.printStackTrace();
            }finally{
                //当客户端断开连接时，安全移除
                removeWriter(writer);
                if(socket != null){
                    try{
                        socket.close();
                    }catch(Exception e){
                        e.printStackTrace();
                    }
                }
            }
        }
```

安全保证

为了实现安全访问，即要对有互斥访问风险的地方进行互斥访问，这里我设置了三处，分别是对writer进行记录的结构进行添加和移除的时侯，避免了存在差异性结果；对于发送信息时，我也要求对这个writer的整体存储List进行互斥访问。

```
    private synchronized void addWriter(PrintWriter w){
        writers.add(w);
    }
    private synchronized void removeWriter(PrintWriter w){
        writers.remove(w);
    }
    private synchronized void sendMessage(String message){
        for(PrintWriter w:writers){
            w.println(message);
        }
    }
```

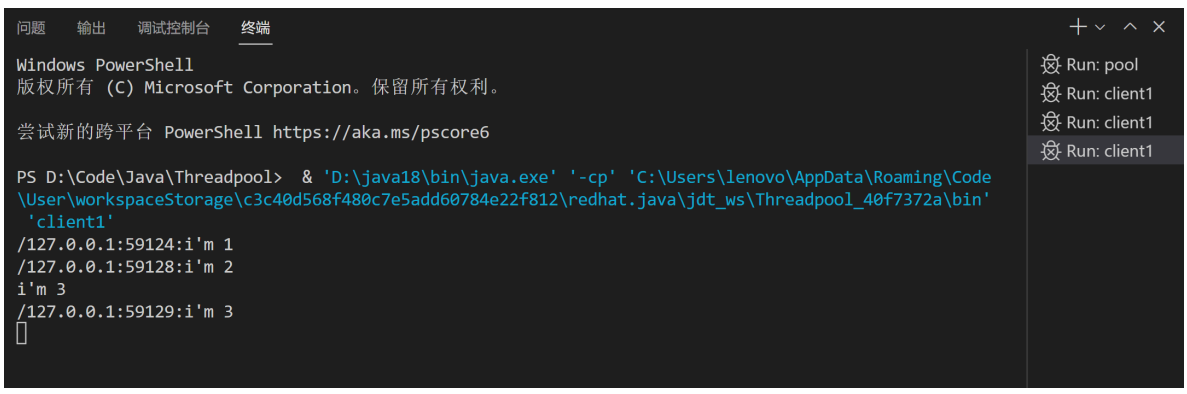
4. 实验测试

接收连接

成功接收连接，并且显示出每一个显示的用户

消息转发

如下，展示了每一个客户端进行消息接收，可以成功接收到信息。



5. 实验心得

在考试前紧急重写了一份应对压力测试的代码，用了我所知道可以较好处理的线程池和互斥访问的技术。

之前的那份代码实验了图形化界面，看起来更加的清晰美观，我本来想把那边的可视化界面也移动过来，但是奈何那里约定了好几个特殊的报文格式，如果要改的话这里这份程序就有太多需要修改的了。临近期末复习，没有来得及，希望期末能成功把这一份代码完善好，实现好。