线程池的使用

线程池的作用

线程池作用就是限制系统中执行线程的数量。

根据系统的环境情况，可以自动或手动设置线程数量，达到运行的最佳效果；少了浪费资源，多了造成系统拥挤效率不高。用线程池控制线程数量，其他线程排队等候。

一个任务执行完毕，再从队列中取最前面的任务开始执行。若队列中没有等待线程，线程池的这一资源处于等待。当一个新任务需要运行时，如果线程池中有等待的工作线程，就可以开始运行了；否则进入等待队列。

实验介绍

Knock Knock 游戏是一个非常流行的语言类游戏，是训练孩童、小学生语言表达能力和现象能力的有趣途径，因此这个游戏常见于家庭娱乐与同伴之间的玩乐中。这个游戏有两个玩家，一个扮演敲门人，一个扮演开门人，分别称为“门外人”和“门内人”。

门外人：发起会话，模仿敲门

门内人：总是问 Who’s there?

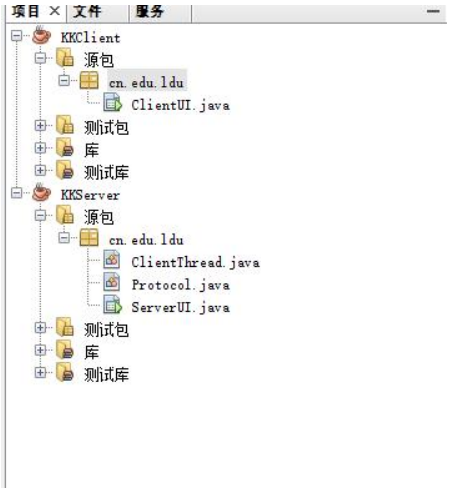
门外人：说一个自己想说的单词，例如 Eye

门内人：接着问 Eye who?

门外人：游戏最关键的地方，例如接着前面的 Eye 说 Ice-cream，发音很相似，但意思不相干，充满幽默感。

程序结构和功能

服务器扮演门外人，客户端扮演门内人。



ClientUI：包含消息面板，连接服务器和发送消息事件

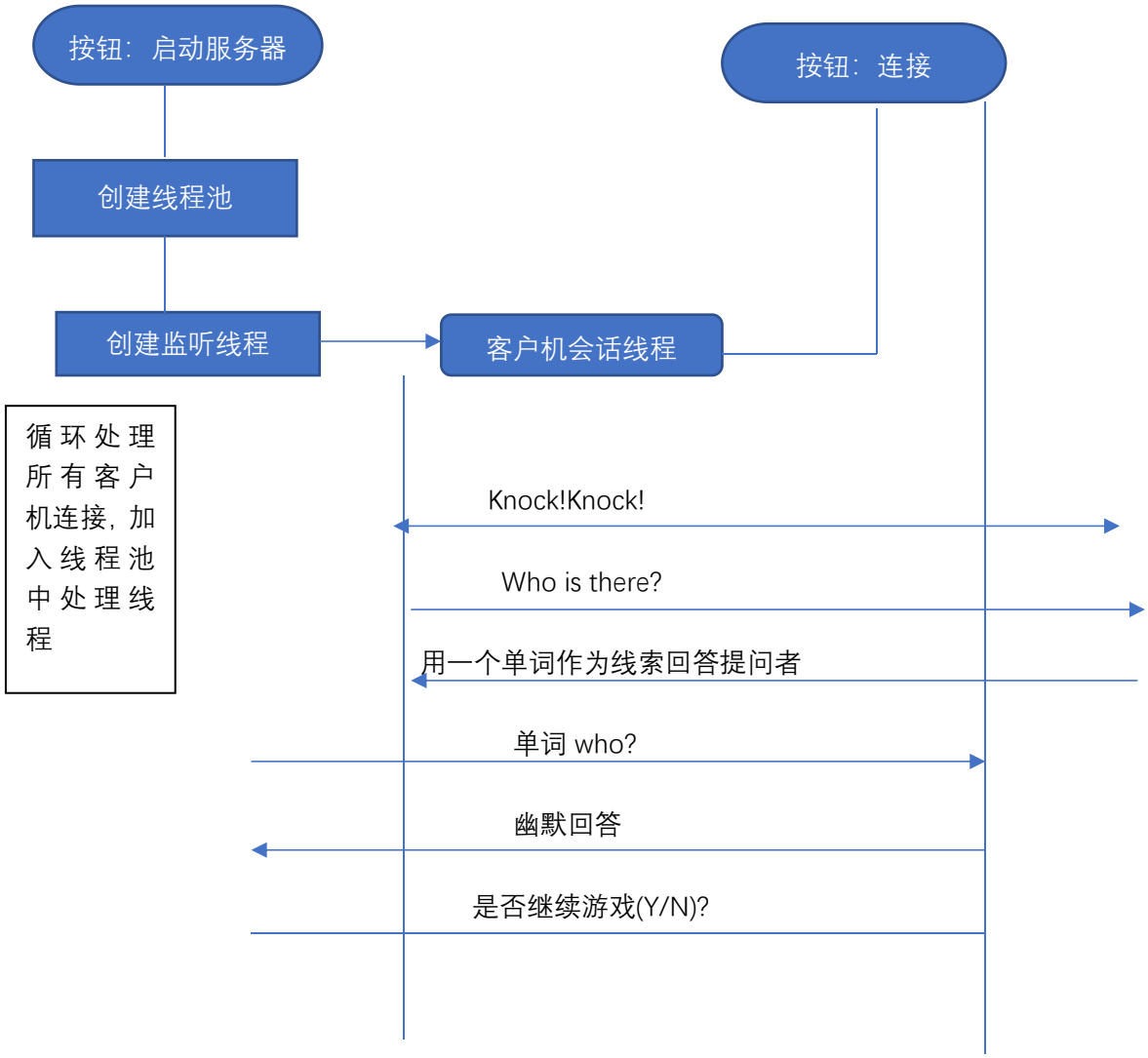
ServerUI：服务器面板，包括服务器启动和创建线程池，处理客户机连接线程。

ClientThread：客户线程，与服务器进行会话

Protocol：固定门外人和门内人的问话和回答的格式，使其按照一定的协议来交流

程序流程图

服务器窗口:TCP 连接　　客户机连接窗口



程序代码流程

启动服务器

```
private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        btnStart.setEnabled(false);
        txtHostName.setEnabled(false);
        txtHostPort.setEnabled(false);
        //获取启动参数
        String hostName=txtHostName.getText();
        int hostPort=Integer.parseInt(txtHostPort.getText());
        //构建套接字格式的地址
        SocketAddress serverAddr=new InetSocketAddress(InetAddress.getByName(hostName),hostPort);
        listenSocket=new ServerSocket(); //创建侦听套接字
        listenSocket.bind(serverAddr); //绑定到工作地址
        int processors=Runtime.getRuntime().availableProcessors();//CPU数
        fixedPool=Executors.newFixedThreadPool(2);//创建固定大小线程池
        long currentId=Thread.currentThread().getId();
        txtArea.append("服务器CPU数："+processors+",固定线程池大小："+processors*2+",当前侦听线程ID："+currentId+"，服务器正等待客户机连接.
```

```
new Thread(new Runnable() {
    public void run() {
      try {
        while (true) { //处理所有客户机连接
            toClientSocket=listenSocket.accept();//如果无连接，则阻塞，否则接受连接并创建新的会话套接字
            clientCounts++;
            txtArea.append(toClientSocket.getRemoteSocketAddress()+" 客户机编号："+clientCounts+" 连接到服务器，会话开始...\n")
            //客户会话线程为SwingWorker类型的后台工作线程
            Thread clientThread=new ClientThread(toClientSocket,clientCounts); //创建客户线程
            fixedPool.execute(clientThread); //用线程池调度客户线程运行
            //clientThread.start(); //这样做就是一客户一线程
        }//end while
      } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "错误提示", JOptionPane.ERROR_MESSAGE);
      }//end try catch
    }//end run()
}).start();
```

点击“启动”按钮，创建侦听套接字，绑定到工作地址，创建固定大小的线程池。另外开辟一个线程用于处理所有客户机连接，启动 SwingWorker 类型的客户会话后台线程，“一会话一线程”。

会话协议

```
private int state=WAITING; //会话状态
private int currentJoke=0; //计数
//以下两个数组分别存储敲门人的两次回答
private String[] clues={"Buster","Orange","Ice cream","Tunis","Old lady","Yah","Dishes","Amish"};
private String[] answers={"Buster Cherry! Is your daughter home?","Orange you going to answer the door?","Ice cream if you don't
public String protocolWorking(String question) { //question门内人的问话
    String answer=null; //敲门人的回答
    switch (state) {
        case WAITING: //开始敲门
            answer="Knock! Knock!";
            state=SENTKNOCKKNOCK;
            break;
        case SENTKNOCKKNOCK: //谁在敲门? 问答
            if (question.equalsIgnoreCase("Who's there?")) {
                answer=clues[currentJoke];
                state=SENTCLUE;
            }else {
                answer="你应该问：\"Who's there?\""+"重新开始：Knock! Knock!";
            }
            break;
        case SENTCLUE: //追问敲门人问答
            if (question.equalsIgnoreCase(clues[currentJoke]+" Who?")) {
                answer=answers[currentJoke]+" 是否继续 ? (y / n ?)";
                state=SENTANSWER;
            }else {
```

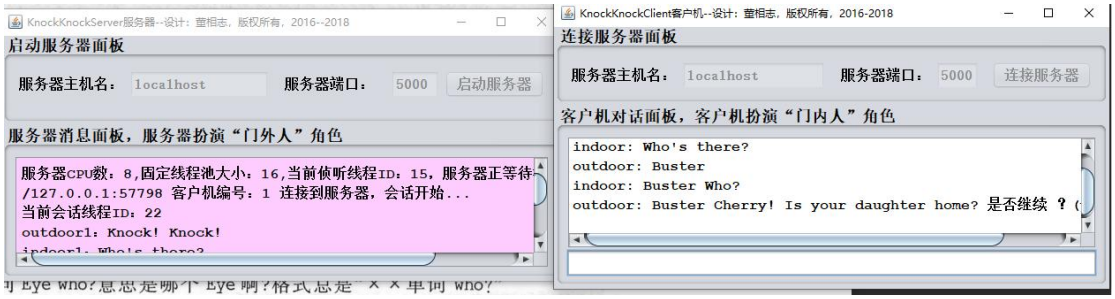
记录当前的会话状态和计数器，用两个数组分别存储敲门人的两次回答，根据当前的状态给出相应的答案，相应地也要更新当前状态，SENTKNOCKKNOCK、SENTCLUE、SENTANSWER。

客户机会话线程

```java
@Override
public void run() {
    try {
        //创建绑定到套接字toClientSocket上的网络输入流与输出流
        in=new BufferedReader(new InputStreamReader(toClientSocket.getInputStream(),"UTF-8"));
        out=new PrintWriter(new OutputStreamWriter(toClientSocket.getOutputStream(),"UTF-8"),true);
        long currentId=Thread.currentThread().getId();
        ServerUI.txtArea.append("当前会话线程ID: "+currentId+"\n"); //发布到process
        //根据服务器协议，在网络流上进行读写操作
        protocol=new Protocol(); //生成协议对象
        String outdoorStr; //门外人的回答
        String indoorStr; //门内人的问话
        outdoorStr=protocol.protocolWorking(null); //根据协议生成门外人的问话
        out.println(outdoorStr); //向客户机发起会话
        ServerUI.txtArea.append("outdoor"+clientCounts+": "+outdoorStr+"\n"); //发布门外人的话到process处理
        while ((indoorStr=in.readLine())!=null) { //只要客户机不断开连接则反复读
            ServerUI.txtArea.append("indoor"+clientCounts+": "+indoorStr+"\n");  //发布门内人的话到process处理
            //根据协议生成回答消息
            outdoorStr=protocol.protocolWorking(indoorStr);
            out.println(outdoorStr); //向客户机发送回答
            ServerUI.txtArea.append("outdoor"+clientCounts+": "+outdoorStr+"\n"); //发布门外人的话到process处理
            if (outdoorStr.endsWith("Goodbye!")) //结束游戏
                break;
        }//end while
        ServerUI.clientCounts--;  //客户机总数减1
        //因为客户机断开了连接，所以释放资源
        if (in!=null) in.close();
        if (out!=null) out.close();
        if (toClientSocket!=null) toClientSocket.close();
```

创建绑定到套接字 toClientSocket 上的网络输入流与输出流，当前 tcp 连接的 socket 套接字被当成参数传入函数。再根据 Protocal 协议函数调用生成门外人的问话和门内人的回答。

实验运行结果



当线程池的大小设置为 2 时，创建第 3 个线程时此线程发生阻塞，是因为此线程当前处在等待队列。