 未标题-1

**软件体系结构**

**实验报告**

**实验题目: C/S架构的在线聊天客户端系统UML建模**

**C/S架构的在线聊天客户端系统详细设计**

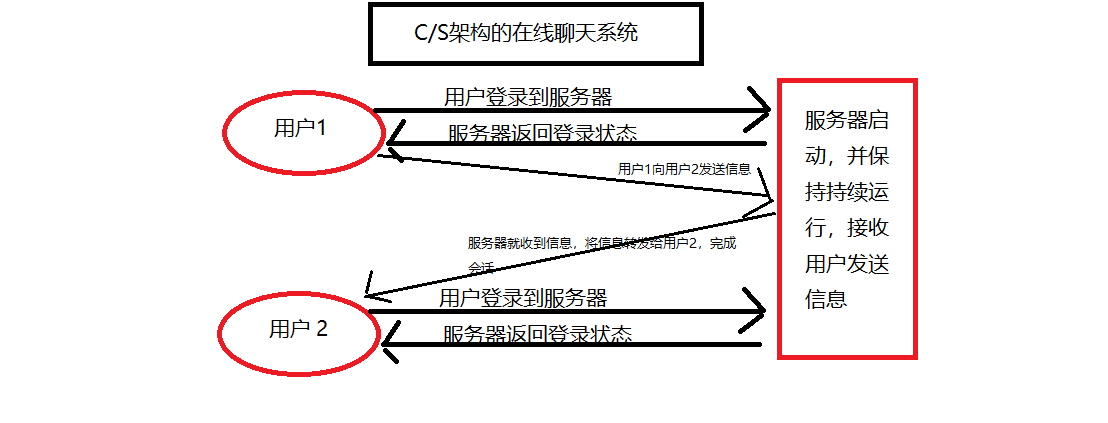
**院系名称： 信息科学与工程学院 专业班级：软件1702**

**学生姓名： 刘文博 学 号： 201716040224**

**指导教师： 刘 灿 日 期： 2020/4/11**

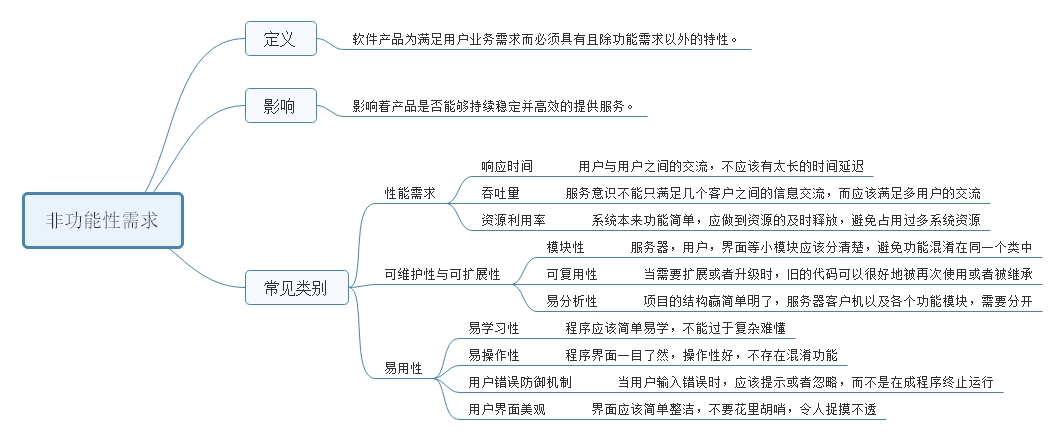
1. **需求**

**功能性需求**



1. 在线聊天系统需要服务器一直保持运行，时刻接受用户发起的登录以及用户间的会话信息，并正确的将用户的信息准确无误的转发出去。客户机启动，选择需要启动的端口，与服务器进行连接，并且可以向在线的用户发送信息，进行实时通信。

**非功能性需求**



1. **技术**

通常，我们写服务器处理模型的程序时，有以下几种模型：

（1）每收到一个请求，创建一个新的进程，来处理该请求；

（2）每收到一个请求，创建一个新的线程，来处理该请求；

（3）每收到一个请求，放入一个事件列表，让主进程通过非阻塞I/O方式来处理请求

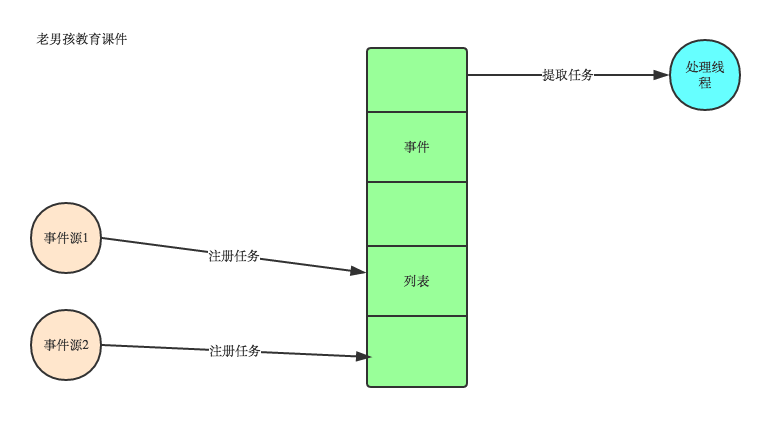
上面的几种方式，各有千秋，

第（1）中方法，由于创建新的进程的开销比较大，所以，会导致服务器性能比较差,但实现比较简单。

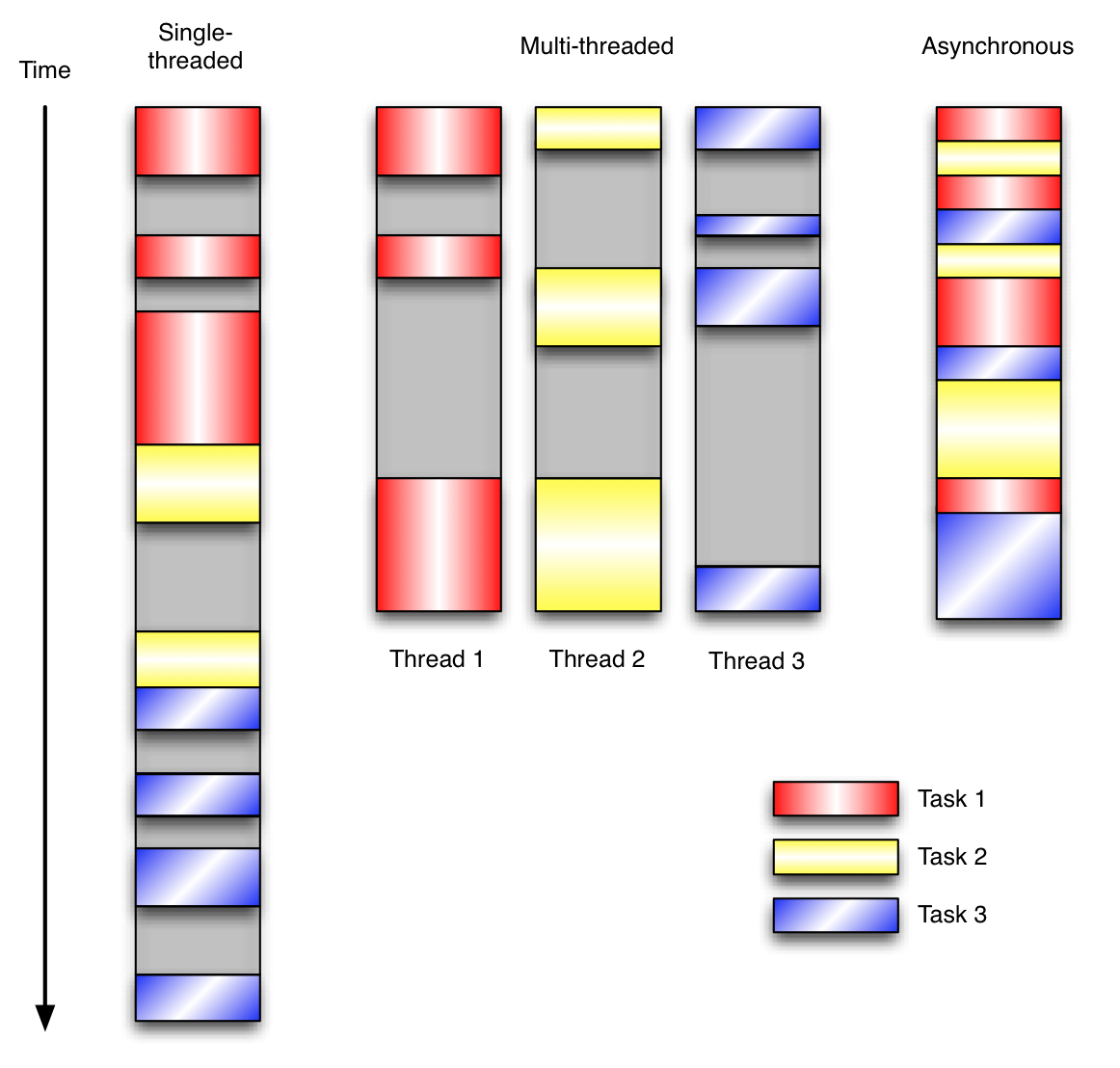
第（2）种方式，由于要涉及到线程的同步，有可能会面临死锁等问题。

第（3）种方式，在写应用程序代码时，逻辑比前面两种都复杂。

目前大部分的UI编程都是事件驱动模型，如很多UI平台都会提供onClick()事件，这个事件就代表鼠标按下事件。事件驱动模型大体思路如下：  
1. 有一个事件（消息）队列；  
2. 鼠标按下时，往这个队列中增加一个点击事件（消息）；  
3. 有个循环，不断从队列取出事件，根据不同的事件，调用不同的函数，如onClick()、onKeyDown()等；  
4. 事件（消息）一般都各自保存各自的处理函数指针，这样，每个消息都有独立的处理函数；

  
事件驱动编程是一种编程范式，这里程序的执行流由外部事件来决定。它的特点是包含一个事件循环，当外部事件发生时使用回调机制来触发相应的处理。另外两种常见的编程范式是（单线程）同步以及多线程编程。

让我们用例子来比较和对比一下单线程、多线程以及事件驱动编程模型。下图展示了随着时间的推移，这三种模式下程序所做的工作。这个程序有3个任务需要完成，每个任务都在等待I/O操作时阻塞自身。阻塞在I/O操作上所花费的时间已经用灰色框标示出来了。

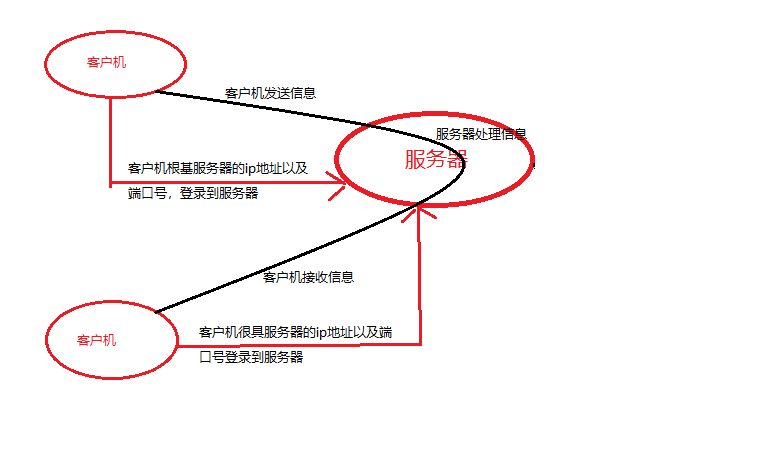


在单线程同步模型中，任务按照顺序执行。如果某个任务因为I/O而阻塞，其他所有的任务都必须等待，直到它完成之后它们才能依次执行。这种明确的执行顺序和串行化处理的行为是很容易推断得出的。如果任务之间并没有互相依赖的关系，但仍然需要互相等待的话这就使得程序不必要的降低了运行速度。

从上述图例中可以看出，才用事件驱动，并采用线程的方式，可以提高系统的效率，减少系统资源的浪费。

1. **架构**

最初的设想是：启动服务器，保持服务器一直运行，然后再启动客户机，客户机连接上服务器，就可以进行在线聊天了，但是发现此时，系统的IP地址以及端口号都未被指定，所以进行了一个管理员登录的过程，由管理员指定服务器的端口号IP地址等信息，然后启动服务器，并可以进行对用户的强制下线操作

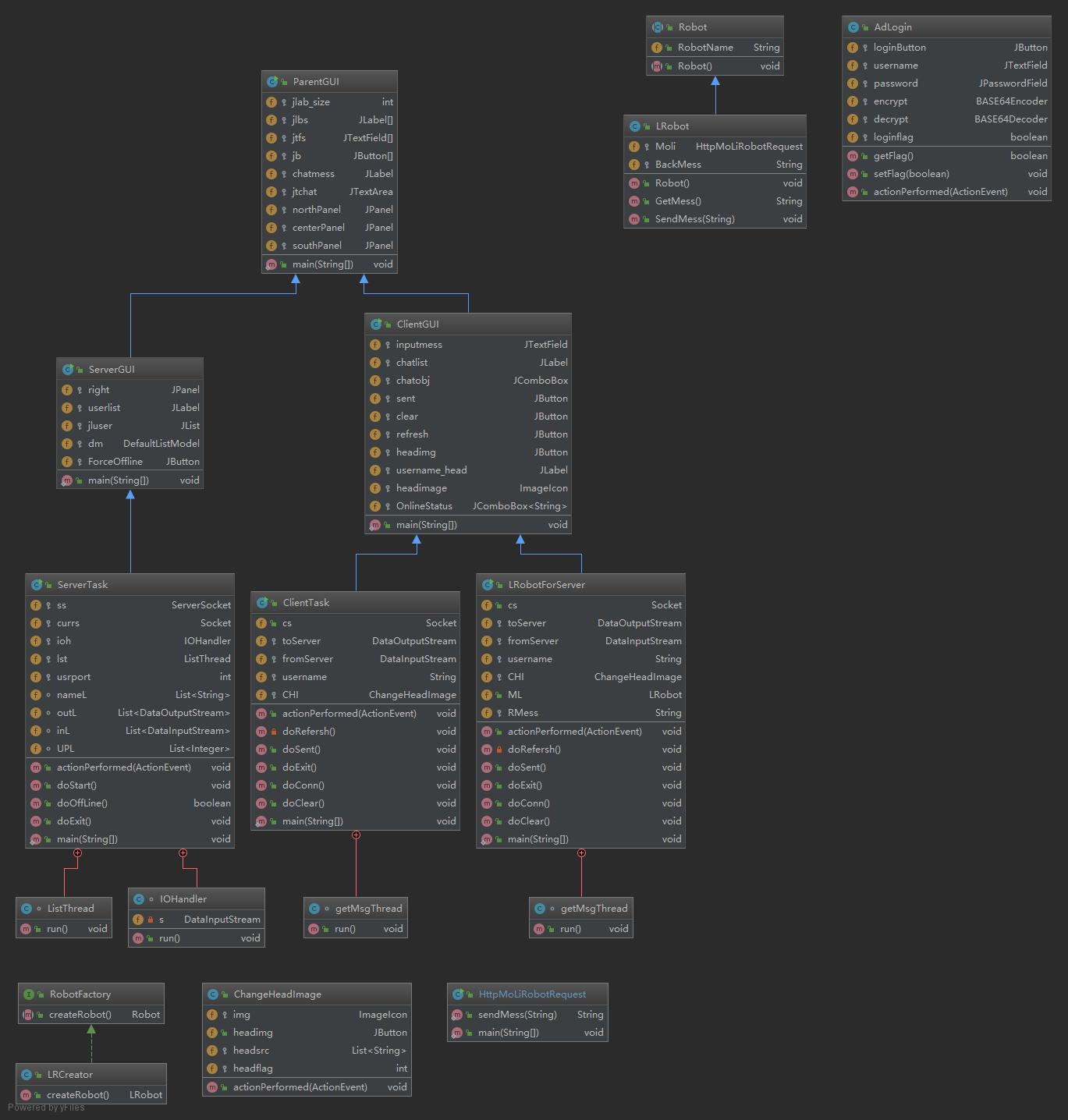
****

**服务器直接启动**

****

**由管理员指定端口启动服务器**

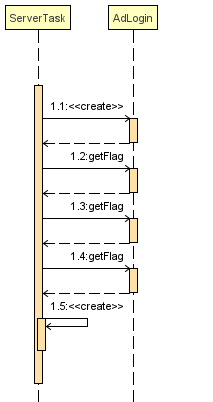
**程序的UML图：**

****

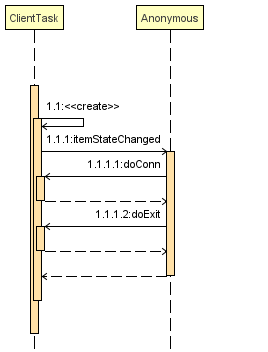
1. **设计**

考虑到在线聊天系统会有多个用户同时进行聊天，该系统用到了多线程技术，为每一个客户开辟一个线程，等待用户发送信息。

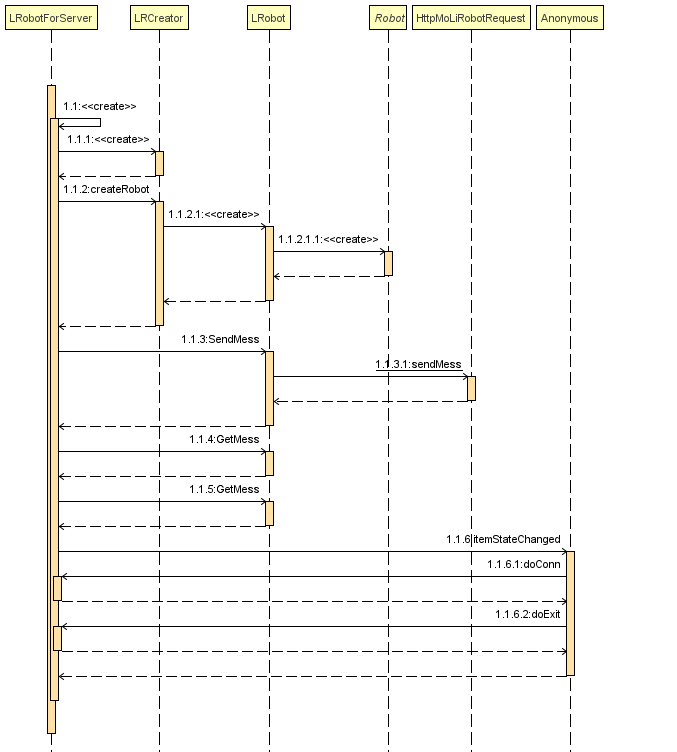
**服务器时序图如下所示：**

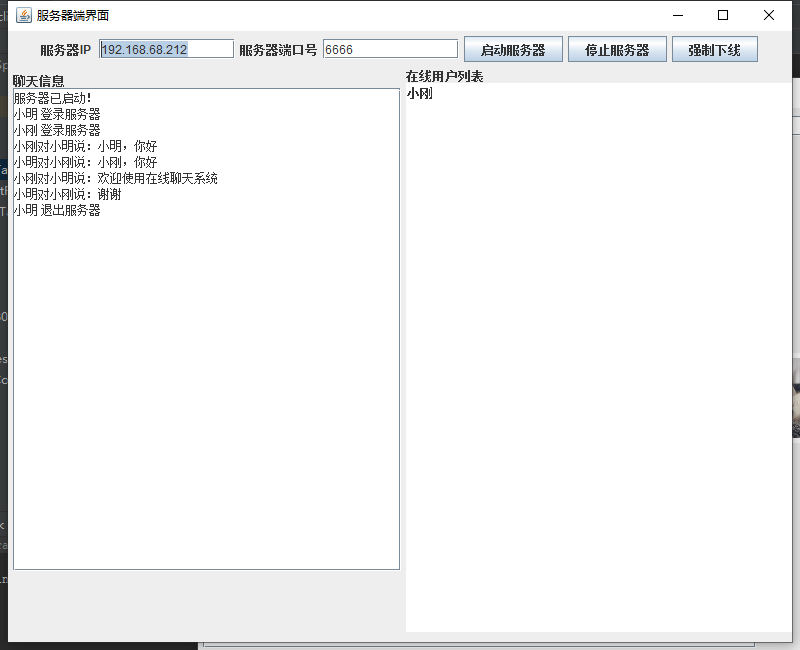
****

**客户端时序图如下所示：**

****

**机器人服务器时序图如下所示：**

****





可以看出，同一个主机可以支持多用户在线，这就体现除了多线程的优点，每个用户之间的通信互不干扰，解决了传统系统一直等待io阻塞的缺点，使得系统资源的利用率大大提高，使用户的体验更加良好

**五．实现**

**对于每一个用户，都采用多线程的方式，可以使多用户同时进行聊天。**

|  |
| --- |
| lass getMsgThread extends Thread{  public void run() {  while(true) {  try {  String jaStr = fromServer.readUTF();  JSONArray ja = new JSONArray(jaStr);  String flat = ja.getString(0);  if(flat.equals("3&&")){  chatobj.removeAllItems();  for(int i=1;i<ja.length();i++) {  String s = ja.getString(i);  if(s.equals(username)) continue;  chatobj.addItem(s);  }  chatobj.addItem("all");  }  else if(flat.equals("2&&")){  String s = ja.getString(1);  chatobj.removeItem(s);  }  else if(flat.equals("4&&"))  {  doExit();  jtchat.append("您已被踢出服务器！");  }  else {  // 发送信息用户的name不能是"3&&"，"2&&"  String fromServerName = ja.getString(0);  String content = ja.getString(1);  JSONObject messjson;  if(content.startsWith("{")){  messjson = new JSONObject(content);  content= messjson.getString("content");  //jtchat.append("茉莉机器人："+messjson.getString("content")+"\n");  }  if(!fromServerName.equals(username))  jtchat.append(fromServerName+"对我说："+content+"\n");  }  }catch(IOException e) {  }  }  }  } |

**服务器启动多线程，接受并转发每一位用户的信息。**

|  |
| --- |
| class ListThread extends Thread{  public void run(){  currs=null;  while(true){  try{  currs=ss.accept();  // System.out.println(currs.getRemoteSocketAddress());  DataInputStream dis = new DataInputStream(currs.getInputStream());  DataOutputStream dos = new DataOutputStream(currs.getOutputStream());  inL.add(dis);  outL.add(dos);  new IOHandler(dis).start();  }catch(IOException e){}  }  }  }  class IOHandler extends Thread{  private DataInputStream s;  public IOHandler(DataInputStream s){  this.s=s;  }  public void run(){  try{  while(true) {  String mess = s.readUTF();  //发送姓名  if(mess.startsWith("1&&")) {  String username = mess.substring(3);  nameL.add(username);  dm.addElement(username);  jtchat.append(username + " 登录服务器\n");  //发送用户信息  JSONArray ja = new JSONArray(nameL);  for(int i=0;i<outL.size();i++){  outL.get(i).writeUTF(ja.toString());  }  }  else if(mess.startsWith("2&&")) {  String username = mess.substring(3);  dm.removeElement(username);  int n = nameL.indexOf(username);  nameL.remove(n);  inL.remove(n-1);  outL.remove(n-1);  jtchat.append(username + " 退出服务器\n");    List<String> list = new ArrayList();  list.add("2&&");  list.add(username);  JSONArray ja = new JSONArray(list);  for(int i=0;i<outL.size();i++){  outL.get(i).writeUTF(ja.toString());  }  }  //处理群发消息  //处理一对一聊天信息  else {  JSONArray ja = new JSONArray(mess);  String fromName = ja.getString(0);  String content = ja.getString(1);  String toName = ja.getString(2);  if(toName.equals("all")) {  jtchat.append(fromName + "对所有人说："+content+"\n");  for(int i=0;i<outL.size();i++) {  DataOutputStream toOut = outL.get(i);  toOut.writeUTF(mess);  }  }else {  jtchat.append(fromName + "对"+toName+"说："+content+"\n");  int n = nameL.indexOf(toName);  DataOutputStream toOut = outL.get(n-1);  toOut.writeUTF(mess);  }  }  }  }catch(IOException e) {  e.printStackTrace();  }  }  } |

**机器人会联网根据你给他发送的内容，进行智能回复。**

|  |
| --- |
| public static String sendMess(String param) {  String result = "";  String enParam;  JSONObject CallBack;  BufferedReader in = null;  try {  enParam = URLEncoder.encode(param,"UTF-8");  //对编码进行URL转码  //在线获取人机对话内容  String urlNameString = "http://i.itpk.cn/api.php?question=" + enParam;  URL realUrl = new URL(urlNameString);  // 打开和URL之间的连接  URLConnection connection = realUrl.openConnection();  // 设置通用的请求属性  connection.setRequestProperty("accept", "\*/\*");  connection.setRequestProperty("connection", "Keep-Alive");  connection.setRequestProperty("user-agent","Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");  // 建立实际的连接伪装浏览器发起请求  connection.connect();  // 获取所有响应头字段  Map<String, List<String>> map = connection.getHeaderFields();  // 遍历所有的响应头字段  for (String key : map.keySet()) {  System.out.println(key + "--->" + map.get(key));  }  // 定义 BufferedReader输入流来读取URL的响应  in = new BufferedReader(new InputStreamReader(  connection.getInputStream(),"UTF-8"));  String tempmess = in.readLine();//服务器返回只有一行无需其他操作  if(tempmess.startsWith("?{")) {  CallBack = new JSONObject(in.readLine().substring(1));  result = CallBack.getString("content");  }  else  {  result = tempmess.substring(1);  }  //对服务器返回对象进行检测  //判断是否为JSON  } catch (Exception e) {  System.out.println("发送GET请求出现异常！" + e);  e.printStackTrace();  }  // 使用finally块来关闭输入流  finally {  try {  if (in != null) {  in.close();  }  } catch (Exception e2) {  e2.printStackTrace();  }  }  System.out.println(result);  return result;  } |

**六．总结**

在多线程版本中，这3个任务分别在独立的线程中执行。这些线程由操作系统来管理，在多处理器系统上可以并行处理，或者在单处理器系统上交错执行。这使得当某个线程阻塞在某个资源的同时其他线程得以继续执行。与完成类似功能的同步程序相比，这种方式更有效率，但程序员必须写代码来保护共享资源，防止其被多个线程同时访问。多线程程序更加难以推断，因为这类程序不得不通过线程同步机制如锁、可重入函数、线程局部存储或者其他机制来处理线程安全问题，如果实现不当就会导致出现微妙且令人痛不欲生的bug。

在事件驱动版本的程序中，3个任务交错执行，但仍然在一个单独的线程控制中。当处理I/O或者其他昂贵的操作时，注册一个回调到事件循环中，然后当I/O操作完成时继续执行。回调描述了该如何处理某个事件。事件循环轮询所有的事件，当事件到来时将它们分配给等待处理事件的回调函数。这种方式让程序尽可能的得以执行而不需要用到额外的线程。事件驱动型程序比多线程程序更容易推断出行为，因为程序员不需要关心线程安全问题。

作为客户端同时请求数给定的服务器的增加，服务器可以成为重载。对照到一个P2P网络，在其总带宽实际上增加为节点的添加，由于P2P网络的总带宽可以大致计算作为该网络中每个节点的带宽的总和。

客户机服务器模式缺乏一个良好的P2P网络的鲁棒性。在客户端服务器，如果一个重要的服务器失败，客户的要求不能得到满足。在P2P网络，资源通常分布在许多节点。即使一个或多个节点出发，放弃一个下载文件，例如，剩下的节点应该还是有必要来完成下载的数据。