Swing编写灵敏的图形用户界面 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/284/2021_2022_Swing_E7_BC_96_E5_86_c104_284981.htm 不灵敏的图形用户界面会降低应用程序的可用性。当以下现象出现的时候，我们通常说这个用户界面反应不灵敏： 不响应事件的现象； 没有更新的现象； 这些现象在很大程度上与事件的处理方法相关，而在编写Swing应用程序的时候，我们几乎必然要编写方法去响应鼠标点击按钮，键盘回车等事件。在这些方法中我们要编写一些代码，在运行时去触发一些动作。常见动作包括查找，更新数据库等。在这篇文章中通过对一个实例的分析，介绍了一些基本概念，常见的错误以及提出了一个解决方案。 event-dispatching thread 我们一定要记住，事件响应方法的代码都是在event-dispatching thread中执行的，除非你启用另一个线程。 那么，什么是event-dispatching thread呢？单一线程规则：一旦一个Swing组件被实现（realized），所有的有可能影响或依赖于这个组件的状态的代码都应该在event-dispatching thread中被执行。而实现一个组件有两种方式： 对顶层组件调用show(), pack(), 或者setVisible(true)； 将一个组件加到一个已经被实现的容器中。 单一线程规则的根源是由于Swing组件库的大部分方法是对多线程不安全的。 为了支持单一线程模型，Swing组件库提供了一个专门来完成这些与Swing组件相关的操作的线程，而这一线程就是event-dispatching thread。我们的事件响应方法通常都是由这一线程调用的，除非你自己编写代码来调用这些事件响应方法。在这里初学者经常犯的一个错误就是在事件响应方法中完成过多的与修改组件没有

直接联系的代码。其最有可能的效果就是导致组件反应缓慢。比如以下响应按钮事件的代码： String str = null. this.textArea.setText("Please wait..."). try { //do something that is really time consuming str = "Hello, world!". Thread.sleep(1000L). } catch (InterruptedException e) { e.printStackTrace(). } this.textArea.setText(str). 执行之后的效果就是按钮似乎定住了一段时间，直到Done.出现之后才弹起来。原因就是Swing组件的更新和事件的响应都是在event-dispatching thread中完成的，而事件响应的时候，event-dispatching thread被事件响应方法占据，所以组件不会被更新。而直到事件响应方法退出时才有可能去更新Swing组件。 为了解决这个问题，有人也许会试图通过调用repaint()方法来更新组件： final String[] str = new String[1]. this.jTextArea1.setText("Please wait..."). this.repaint(). try { Thread.sleep(1000L). }catch(InterruptedException e) { e.printStackTrace(). } str[0] = "Done.". jTextArea1.setText(str[0]). 但是这一个方法没有起到预期的作用，按钮仍然定住一段时间，在察看了repaint()方法的源代码之后就知道原因了。 PaintEvent e = new PaintEvent(this, PaintEvent.UPDATE, new Rectangle(x, y, width, height)). 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com