再论JavaSwing线程Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E5_86_8D_E8_AE_BAJava_c104_644743.htm 不正确的Swing线程是运行缓慢、无响应和不稳定的Swing应用的主要原因之一。这是许多原因造成的，从开发人员对Swing单线程模型的误解，到保证正确的线程执行的困难。即使对Swing线程进行了很多努力，应用线程逻辑也是很难理解和维护的。本文阐述了如何在开发Swing应用中使用事件驱动编程，以大大简化开发、维护，并提供高灵活性。 背景 既然我们是要简化Swing应用的线程，首先让我们来看看Swing线程是怎么工作的，为什么它是必须的。Swing API是围绕单线程模型设计的。这意味着Swing组件必须总是通过同一个线程来修改和操纵。为什么采用单线程模型，这有很多原因，包括开发成本和同步 Swing的复杂性——这都会造成一个迟钝的API。为了达到单线程模型，有一个专门的线程用于和Swing组件交互。这个线程就是大家熟知的Swing 线程，AWT（有时也发音为“ ought ”）线程，或者事件分派线程。在本文的下面的部分，我选用Swing线程的叫法。 既然Swing线程是和Swing组件进行交互的唯一的线程，它就被赋予了很多责任。所有的绘制和图形，鼠标事件，组件事件，按钮事件，和所有其它事件都发生在Swing线程。因为Swing线程的工作已经非常沉重了，当太多其它工作在Swing线程中进行处理时就会发生问题。会引起这个问题的最常见的位置是在非Swing处理的地方，像发生在一个事件监听器方法中，比如JButton的ActionListener，的数据库查找。既然 ActionListener的actionPerformed()方法自动在Swing线程中

执行，那么，数据库查找也将在Swing线程中执行。这将占用了Swing的工作，阻止它处理它的其它任务——像绘制，响应鼠标移动，处理按钮事件，和应用的缩放。用户以为应用死掉了，但实际上并不是这样。在适当的线程中执行代码对确保系统正常地执行非常重要。 既然我们已经看到了在适当的线程中执行Swing应用的代码是多么重要，现在让我们如何实现这些线程。我们看看将代码放入和移出Swing线程的标准机制。在讲述过程中，我将突出几个和标准机制有关的问题和难点。正如我们看到的，大部分的问题都来自于企图在异步的Swing线程模型上实现同步的代码模型。从那儿，我们将看到如何修改我们的例子到事件驱动——移植整个方式到异步模型。 通用Swing线程解决方案 让我们以一个最常用的Swing线程错误开始。我们将企图使用标准的技术来修正这个问题。在这个过程中，我们将看到实现正确的Swing线程的复杂性和常见困难。并且，注意在修正这个Swing线程问题中，许多中间的例子也是不能工作的。在例子中，我在代码失败的地方以//broken开头标出。好了，现在，让我们进入我们的例子吧。 假设我们在执行图书查找。我们有一个简单的用户界面，包括一个查找文本域，一个查找按钮，和一个输出的文本区域。 用户输入书的标题，作者或者其它条件，然后显示一个结果的列表。下面的代码例子演示了按钮的ActionListener在同一个线程中调用 lookup()方法。在这些例子中，我使用了thread.sleep()休眠5秒来作为一个占位的外部查找。线程休眠的结果等同于一个耗时5秒的同步的服务器调用。 private void searchButton_actionPerformed() { outputTA.setText("Searching for: " searchTF.getText()). //Broken!!

Too much work in the Swing thread String[] results = lookup(searchTF.getText()). outputTA.setText(""). for (int i = 0. i