通过JavaSwing看透MVC设计模式 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/145/2021_2022__E9_80_9A_E8_BF_87Java_c104_145002.htm 一个好的用户界面(GUI)的设计通常可以在现实世界找到相应的表现。例如，如果在您的面前摆放着一个类似于电脑键盘按键的一个简单的按钮，然而就是这么简单的一个按钮，我们就可以看出一个GUI设计的规则，它由两个主要的部分构成，一部分使得它具有了按钮应该具有的动作特性，例如可以被按下。另外一部分则负责它的表现，例如这个按钮是代表了A还是B。 看清楚这两点你就发现了一个很强大的设计方法，这种方法鼓励重用reuse，而不是重新设计redesign。你发现按钮都有相同的机理，你只要在按钮的顶上喷上不同的字母便能制造出“ 不同 ”的按钮，而不用为了每个按钮而重新设计一份图纸。这大大减轻了设计工作的时间和难度。 如果您把上述设计思想应用到软件开发领域，那么取得相似的效果一点都不让人惊奇。一个在软件开发领域应用的非常广泛的技术Model/View/Controller(MVC)便是这种思想的一个实现。 这当然很不错，但是或许您又开始疑惑这和java基础类JFC(Java Foundation Class)中的用户界面设计部分(Swing)又有什么关系呢？好的，我来告诉你。 尽管MVC设计模式通常是用来设计整个用户界面(GUI)的，JFC的设计者们却独创性的把这种设计模式用来设计Swing中的单个的组件(Component)，例如表格Jtable,树Jtree,组合下拉列表框JcomboBox等等等等。这些组件都有一个Model,一个View，一个Controller，而且，这些model,view,controller可以独立的改变，就是当组件正在被使

用的时候也是如此。这种特性使得开发GUI界面的工具包显得非常的灵活。 MVC设计模式把一个软件组件区分为三个不同的部分，model,view,controller。 Model是代表组件状态和低级行为的部分，它管理着自己的状态并且处理所有对状态的操作，model自己本身并不知道使用自己的view和controller是谁，系统维护着它和view之间的关系，当model发生了改变系统还负责通知相应的view。 View代表了管理model所含有的数据的一个视觉上的呈现。一个Model可以有一个以上的View,但是Swing中却很少有这样的情况。 Controller管理着model和用户之间的交互的控制。它提供了一些方法去处理当model的状态发生了变化时的情况。 使用键盘上的按钮的例子来说明一下：Model就是按钮的整个机械装置，View/Controller就是按钮的表面部分。 下面的图解释了如何把一个JFC开发的用户界面分为model,view,controller，注意，view/Controller被合并到了一起，这是MVC设计模式通常的用法，它们提供了组件的用户界面(UI)。 用Button的例子详细说明 为了更好的理解MVC设计模式和Swing用户界面组件之间的关系，让我们更加深入的进行分析。我将采用最常见的组件button来说明。我们从model来开始。 Model 一个按钮的model所应该具备的行为由一个接口ButtonModel来完成。一个按钮model实例封装了其内部的状态，并且定义了按钮的行为。它的所有方法可以分为四类： 1、查询内部状态 2、操作内部状态 3、添加和删除事件监听器 4、发生事件法。 程序员通常并不会直接和model以及view/controller打交道，他们通常隐藏于那些继承自java.awt.Component的组件里面了，这些组件就像胶水一样把MVC三者合三为一。也正是由于这些继承的组件对象，一

个程序员可以很方便的混合使用Swing组件和AWT组件，然后，我们知道，Swing组件有很多都是直接继承自相应的AWT组件，它能提供比AWT组件更加方便易用的功能，所以通常情况下，我们没有必要混合使用两者。一个实例 现在我们已经明白了Java类与MVC各个部分的对应关系，我们可以更加深入一点去分析问题了。下面我们将要讲述一个小型的使用MVC模式开发的例子。因为JFC十分的复杂，我只能把我的例子局限于一个用户界面组件里面（如果你猜是一个按钮的例子，那么你对了！） 让我们来看看这个例子的所有部分吧。 Button类 最显而易见的开始的地方就是代表了按钮组件本省的代码，因为这个类是大部分程序员会接触的。 就像我前面提到的，按钮用户界面组件类实际上就是model和view/controller的之间的黏合剂。每个按钮组件都和一个model以及一个controller关联，model定义了按钮的行为，而view/controller定义了按钮的表现。而应用程序可以在任何事件改变这些关联。让我们看看得以实现此功能的代码。public void setModel(ButtonModel buttonmodel){ if (this.buttonmodel != null) { this.buttonmodel.removeChangeListener(buttonchangelistener). this.buttonmodel.removeActionListener(buttonactionlistener). buttonchangelistener = null. buttonactionlistener = null. } this.buttonmodel = buttonmodel. if (this.buttonmodel != null) { buttonchangelistener = new ButtonChangeListener(). buttonactionlistener = new ButtonActionListener(). this.buttonmodel.addChangeListener(buttonchangelistener). this.buttonmodel.addActionListener(buttonactionlistener). }

0updateButton().}public void setUI(ButtonUI buttonui){ if (this.buttonui != null) { this.buttonui.uninstallUI(this). } this.buttonui = buttonui. if (this.buttonui != null) { this.buttonui.installUI(this). } 0updateButton().}public void 0updateButton(){ invalidate().} 在进入下一节之前，你应该多花一些时间来仔细阅读一下Button类的源代码。 