利用O erver模式实现组件间通信 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022__E5_88_A9_E7_94_A8O_er_c104_252448.htm 1. 问题的提出 以前做一个界面的时候常常会遇到这样的尴尬情况：希望保留各个独立的组件（类），但又希望它们之间能够相互通信。譬如Windows中的Explorer，我们希望鼠标点击左边是树型目录的一个节点，右边的文件浏览能及时列出该节点目录下的文件和子目录，类似这样一个简单的应用，如果只有一个类继承JFrame，而树型组件和浏览文件的面板作为成员，就像：这样当然容易在两者之间传递消息，但是可扩展性较差。通常容易想到的是两种办法：在一个组件里保留另一个组件类型的成员，初始化时作为参数传入引用，比如：class TreePanel extends JPanel{ JTree tree. ...}class FilePanel extends JPanel{ public FilePanel(JTree tree){...} ...} 或者将一个组件线程化，不停地监听另一个组件的变化，然后作出相应的反映，比如：class TreePanel extends JPanel{ JTree tree. ...}class FilePanel extends JPanel implements Runnable{ public void run() { while (true) { //监听tree的变化 } ... } ...} 这样确实可以达到我们的目的，但是第一种方案显然不利于松散耦合，第二种方案比较占用系统资源。通过学习设计模式，我们发现可以用Observer模式来解决这个问题。 2. Observer模式 设计模式分为创建型、结构型和行为型，其中行为型模式专门处理对象间通信，指定交互方式等，Observer模式就是属于行为型的一种设计模式。按照“ 四人帮 ”（Gang of Four）在“ Design Patterns ”里的定义，Observer模式“ 定义对象间的一种一对多的依赖关

系,当一个对象的状态发生改变时,所有依赖于它的对象都得到通知并被自动更新”，这个描述正好符合我们对“组件通信”问题的需求。让我们先看看Observer模式的结构： 其中各元素的含义如下： Subject：被观察的目标的抽象接口，它提供对观察者（Observer）的注册、注销服务，Notify方法通知Observer目标发生改变； Object：观察者的抽象接口，Update方法是当得到Subject状态变化的通知后所要采取的动作； ConcreteSubject：Subject的具体实现； ConcreteObserver：Observer的具体实现 Observer模式在实现MVC结构时非常有用，为数据和数据表示解耦合。 3. Java中的Observer模式：Observer和Observable 在大致了解了Observer模式的描述之后，现在我们更为关心的是它在Java中是如何应用的。幸运的是，自从JDK 1.0起，就有了专门处理这种应用的API，这就是Observer接口和Observable类，它们是属于java.util包的一部分。看来Java的开发者们真是深谙设计模式的精髓，而Java的确是为了真正的面向对象而生的，呵呵！