**[备忘录模式(Memento pattern)](http://www.cnblogs.com/lds85930/articles/643099.html)**

**一、引子**　　俗话说：世上难买后悔药。所以凡事讲究个“三思而后行”，但总常见有人做“痛心疾首”状：当初我要是……。如果真的有《大话西游》中能时光倒流的“月光宝盒”，那这世上也许会少一些伤感与后悔——当然这只能是痴人说梦了。  
  
　　但是在我们手指下的程序世界里，却有的后悔药买。今天我们要讲的备忘录模式便是程序世界里的“月光宝盒”。  
  
　　**二、定义与结构**  
  
　　备忘录（Memento）模式又称标记（Token）模式。GOF给备忘录模式的定义为：在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到原先保存的状态。  
  
　　在讲命令模式的时候，我们曾经提到利用中间的命令角色可以实现undo、redo的功能。从定义可以看出备忘录模式是专门来存放对象历史状态的，这对于很好的实现undo、redo功能有很大的帮助。所以在命令模式中undo、redo功能可以配合备忘录模式来实现。  
  
　　其实单就实现保存一个对象在某一时刻的状态的功能，还是很简单的——将对象中要保存的属性放到一个专门管理备份的对象中，需要的时候则调用约定好的方法将备份的属性放回到原来的对象中去。但是你要好好看看为了能让你的备份对象访问到原对象中的属性，是否意味着你就要全部公开或者包内公开对象原本私有的属性呢？如果你的做法已经破坏了封装，那么就要考虑重构一下了。  
  
　　备忘录模式只是GOF对“恢复对象某时的原有状态”这一问题提出的通用方案。因此在如何保持封装性上——由于受到语言特性等因素的影响，备忘录模式并没有详细描述，只是基于C++阐述了思路。那么基于Java的应用应该怎样来保持封装呢？我们将在实现一节里面讨论。  
  
　　来看下“月光宝盒”备忘录模式的组成部分：  
  
　　1) 备忘录（Memento）角色：备忘录角色存储“备忘发起角色”的内部状态。“备忘发起角色”根据需要决定备忘录角色存储“备忘发起角色”的哪些内部状态。为了防止“备忘发起角色”以外的其他对象访问备忘录。备忘录实际上有两个接口，“备忘录管理者角色”只能看到备忘录提供的窄接口——对于备忘录角色中存放的属性是不可见的。“备忘发起角色”则能够看到一个宽接口——能够得到自己放入备忘录角色中属性。   
  
　　2) 备忘发起（Originator）角色：“备忘发起角色”创建一个备忘录，用以记录当前时刻它的内部状态。在需要时使用备忘录恢复内部状态。  
  
　　3) 备忘录管理者（Caretaker）角色：负责保存好备忘录。不能对备忘录的内容进行操作或检查。  
  
　　备忘录模式的类图真是再简单不过了：

|  |
| --- |
| http://dev.yesky.com/imagelist/05/08/um649oea69lz.jpg |

　　**三、举例**  
  
　　按照定义中的要求，备忘录角色要保持完整的封装。最好的情况便是：备忘录角色只应该暴露操作内部存储属性的的接口给“备忘发起角色”。而对于其他角色则是不可见的。GOF在书中以C++为例进行了探讨。但是在Java中没有提供类似于C++中友元的概念。在Java中怎样才能保持备忘录角色的封装呢？  
  
　　下面对三种在Java中可保存封装的方法进行探讨。  
  
　　第一种就是采用两个不同的接口类来限制访问权限。这两个接口类中，一个提供比较完备的操作状态的方法，我们称它为宽接口；而另一个则可以只是一个标示，我们称它为窄接口。备忘录角色要实现这两个接口类。这样对于“备忘发起角色”采用宽接口进行访问，而对于其他的角色或者对象则采用窄接口进行访问。  
  
　　这种实现比较简单，但是需要人为的进行规范约束——而这往往是没有力度的。  
  
　　第二种方法便很好的解决了第一种的缺陷：采用内部类来控制访问权限。将备忘录角色作为“备忘发起角色”的一个私有内部类。好处我不详细解释了，看看代码吧就明白了。下面的代码是一个完整的备忘录模式的教学程序。它便采用了第二种方法来实现备忘录模式。  
  
　　还有一点值得指出的是，在下面的代码中，对于客户程序来说“备忘录管理者角色”是不可见的，这样简化了客户程序使用备忘录模式的难度。下面采用“备忘发起角色”来调用访问“备忘录管理者角色”，也可以参考门面模式在客户程序与备忘录角色之间添加一个门面角色。

http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gifclass Originator{  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//这个是要保存的状态   
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　private int state= 90;  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//保持一个“备忘录管理者角色”的对象  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　private Caretaker c = new Caretaker();  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//读取备忘录角色以恢复以前的状态  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　public void setMemento(){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　Memento memento = (Memento)c.getMemento();  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　state = memento.getState();   
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　System.out.println("the state is "+state+" now");  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//创建一个备忘录角色，并将当前状态属性存入，托给“备忘录管理者角色”存放。  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　public void createMemento(){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　c.saveMemento(new Memento(state));   
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//this is other business methodshttp://www.cnblogs.com/Images/dot.gif  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//they maybe modify the attribute state   
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　public void modifyState4Test(int m){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　state = m;  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　System.out.println("the state is "+state+" now");  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}   
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//作为私有内部类的备忘录角色，它实现了窄接口，可以看到在第二种方法中宽接口已经不再需要  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　//注意：里面的属性和方法都是私有的  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　private class Memento implements MementoIF{  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　private int state ;  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　　private Memento(int state){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　　this.state = state ;   
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　　}  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　　private int getState(){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　　return state;   
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　　}   
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif　}  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif　//测试代码——客户程序  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif　public class TestInnerClass{  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　public static void main(String[] args){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　Originator o = new Originator();   
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　o.createMemento();  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　o.modifyState4Test(80);  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　o.setMemento();  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif　}  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif　//窄接口  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif　interface MementoIF{}  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif　//“备忘录管理者角色”  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif　class Caretaker{  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　private MementoIF m ;  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　public void saveMemento(MementoIF m){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　this.m = m;   
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif　　public MementoIF getMemento(){  
http://www.cnblogs.com/Images/OutliningIndicators/InBlock.gif　　　return m;   
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif　　}  
http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif　}  
http://www.cnblogs.com/Images/OutliningIndicators/None.gif

第三种方式是不太推荐使用的：使用clone方法来简化备忘录模式。由于Java提供了clone机制，这使得复制一个对象变得轻松起来。使用了clone机制的备忘录模式，备忘录角色基本可以省略了，而且可以很好的保持对象的封装。但是在为你的类实现clone方法时要慎重啊。  
  
　　在上面的教学代码中，我们简单的模拟了备忘录模式的整个流程。在实际应用中，我们往往需要保存大量“备忘发起角色”的历史状态。这时就要对我们的“备忘录管理者角色”进行改造，最简单的方式就是采用容器来按照顺序存放备忘录角色。这样就可以很好的实现undo、redo功能了。  
  
　　**四、适用情况**  
  
　　从上面的讨论可以看出，使用了备忘录模式来实现保存对象的历史状态可以有效地保持封装边界。使用备忘录可以避免暴露一些只应由“备忘发起角色”管理却又必须存储在“备忘发起角色”之外的信息。把“备忘发起角色”内部信息对其他对象屏蔽起来, 从而保持了封装边界。  
  
　　但是如果备份的“备忘发起角色”存在大量的信息或者创建、恢复操作非常频繁，则可能造成很大的开销。  
  
　　GOF在《设计模式》中总结了使用备忘录模式的前提：   
  
　　1) 必须保存一个对象在某一个时刻的(部分)状态, 这样以后需要时它才能恢复到先前的状态。  
  
　　2) 如果一个用接口来让其它对象直接得到这些状态，将会暴露对象的实现细节并破坏对象的封装性。  
  
　　**五、总结**  
  
　　介绍了怎样来使用备忘录模式实现存储对象历史状态的功能，并对基于Java的实现进行了讨论。欢迎大家指正。