## 第22章——备忘录模式

## 22.1 来自设计师的无奈

熊小猫：兔小白，到下班时间了，还不回家？

兔小白：今天收到一个需求变更，之前的需求文档遗漏了一个场景，我开发完再走。哎！要是软件开发永远都没有需求变更就好了！

熊小猫：你的这个愿望太理想化了。如此复杂的软件系统，妄想在需求分析阶段，一点问题都不出是不可能的。软件工程的一切方法论都只能降低需求变更的可能性。软件开发中的需求变更还算比较好处理，一般不会涉及到核心代码改动。当然，这也要看你的程序设计是不是足够灵活。

兔小白：还有比软件开发变更更多的工种吗？

熊小猫：当然有了，我有一位朋友是平面设计师，他的工作是设计演唱会海报。他和我吐槽，有一次一张海报改了十多版，甲方最后说，比较了这么多版，反而觉得第一版最好。但是让他吐血的是，由于改了太多版本，他已经找不到第一个版本了。

//好多海报，1，2，3.。。。甲方，还是第一个好点，设计师，啥，源文件好像找不到了。

兔小白：看来设计师没有使用Git啊，否则每一个版本都有存档。

熊小猫：让设计师使用Git要求高了点吧？他都是在文件夹里自己拷贝一份，改名备份。

兔小白：这个方法也太土了吧！

熊小猫：这个方法确实是既麻烦，又容易出错。说到这，我想考考你，假如程序中需要保存对象的历史状态你会怎么做？

兔小白：这还不好办，我将对象在某个时刻的状态快照保存下来就行啦！

熊小猫：快照没问题，但是怎么照，怎么保存并没有那么简单。否则也不会有一种设计模式专门用来解决此类问题。这种设计模式叫做备忘录模式。

兔小白：我觉得没有多难呀，你给我讲讲备忘录模式？

## 22.2 克隆实现海报设计存档

熊小猫：你还是先自己尝试去实现，理解效果才会更好。需求是这样的，一张海报包含图片、文字、演出时间这几种元素，海报在设计过程中可以随时存档，并且可以加载历史存档。

兔小白：我设计的程序一定不会重蹈你设计师朋友的覆辙。

15分钟后，兔小白完成了第一版程序。

兔小白：Poster为海报类，三个属性分别对应图片、标题、演出时间。print方法负责输出海报内容。由于存档需要保存当前海报对象的快照，所以我在这里使用了克隆。

//代码

Caretaker类是一个存档管理器，职责是保存和读取存档。初始化时，需要设置存档的数量限制。

//代码

客户端代码我设计了两版海报，并且分别进行了存档。然后从Caretaker中读取这两个存档进行输出。

//代码

输出结果符合预期，两版存档被正确保存和读取。

//代码

你的设计师朋友，如果有像Caretaker这样的存档工具， 他就可以方便地进行存档管理，不至于需要的时候发现没存或者找不到了。

熊小猫：将管理存档的职责抽象成Caretaker类的想法很好，但是距离备忘录模式还差一步！你再想一想，组成海报的元素有哪些？

兔小白：现在组成海报的元素只有图片、标题、时间。

熊小猫：所以存档的时候，是不是只存储这三个属性的值就可以了呢？读取存档的时候，用这三个属性的存档值，设置到Poster对象中。Poster对象就恢复到了存档的状态。

兔小白：现在保存的是整个Poster对象，确实没有必要。

熊小猫：这里完全可以进行优化。存档只需要属性的值，可以数据部分剥离出来，存档仅保存属性值。多个属性可以单独封装到一个类中。

兔小白：我明白了，存档数据单独封装在一个对象中，用来保存Poster属性的数据快照。我这就去修改。

15分钟后，兔小白改好了代码。

兔小白：我增加了Memento类，用来保存存档的数据，也就是Poster对象的picture、title、time三个属性的值。并提供三个读取属性值的方法。

//代码

Poster增加了两个方法，一个用于保存存档为Memento对象，还一个是读取Memento的数据，将自己恢复到存档的状态。Clone方法不需要了，移除掉。

//代码

由于存档对象从Poster变为了Memento，所以Caretaker只是改为了维护Memento。

//代码

客户端代码改动不大，Poster对象通过setMemento方法恢复到存档的状态。

//代码

输出结果和之前一样，符合预期。

## 22.3 备忘录模式适合的场景

熊小猫：这版代码其实已经运用了备忘录模式。下面我们来看看备忘录模式的结构图。

//图

Originator为原发器 ，如例子中的Poster。它可以创建备忘录对象，记录自己在当时的内部状态。它还可以使用备忘录对象恢复自己的状态。Memento为备忘录，负责存储Originator的内部状态。Caretaker为备忘录管理器，负责保存和读取备忘录。

备忘录模式的优点是将对象内部状态抽象为备忘录。这样可以达到更好的封装效果，当Originator有