Memento（备忘录）

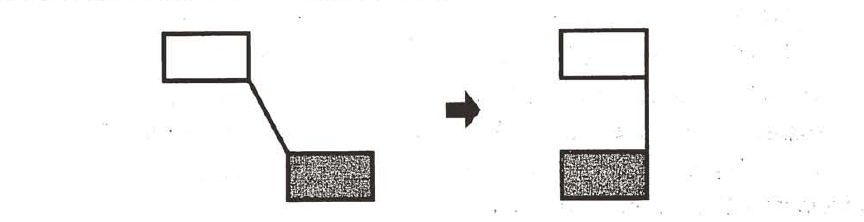
意图

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。

动机

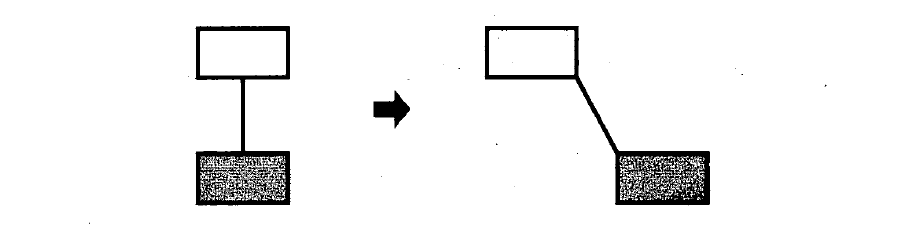
有时有必要记录一个对象的内部状态。为了允许用户取消不确定的操作或从错误中恢复过来，需要实现检查点或取消机制，而要实现这些机制，你必须实现将状态信息保存在某处，这样才能将对像恢复到它们先前的状态。但是对象通常封装了其部分或所有的状态信息，使得其状态不能被其他对象访问，也就不可能在该对象之外保存其状态。而暴露其内部状态又将违反封装的原则，可能有损应用的可靠性和可扩展性。

例如，考虑一个图形编辑器，它支持图形对象间的连线。用户可用一条直线连接两个矩形，而当用户移动任意一个矩形时，这两个矩形仍能保持连接。在移动过程中，编辑器自动伸展两条直线以保持该连接。



一个众所周知的保持对象间连接关系的方法是使用一个约束解释系统。我们可将这一功能封装在一个ConstraintSolver对象中。ConstraintSolver在连接生成时，记录这些连接并产生描述它们的数学方程。当用户生成一个连接或修改图形时，ConstraintSolver就求解这些方程。并根据他的计算结果重新调整图形，使各个对象保持正确的连接。

在这一应用中，支持取消操作并不像看起来那么容易。一个显而易见的方法是，每次移动时保存移动的距离，而在取消这次移动时该对象移回相等的距离。然而，这并不能保证所有的对象都会出现在它们原先出现的地方。设想在移动过程中某连接中有一些松弛。在这种情况下，简单地将矩形移回它原来的位置并不一定能得到预想的结果。

一般来说，ConstraintSolver的公共接口可能不注意精确地逆转它对其他对象的应用。为重建先前的状态，取消操作机制必须与ConstraintSolver更紧密的结合，但我们同时也应避免将ConstraintSolver的内部暴露给取消操作机制。

我们可用备忘录模式解决这一问题。一个备忘录是一个对象，它存储另一个对象在某个瞬间的内部状态，而后者称为备忘录的原发器。当需要设置原发器的检查点时，取消操作机制会向原发器请求一个备忘录。原发器用描述当前状态的信息初始化该备忘录。只有原发器可以向备忘录中存取信息，备忘录对其他对象不可见。

在刚才讨论的图形编辑器的例子中，ConstraintSolver可作为一个原发器。下面的时间序列描述了取消操作的过程：

1. 作为移动操作的一个副作用，编辑器向ConstraintSolver请求一个备忘录
2. ConstraintSolver创建并返回一个备忘录，在这个例子中该备忘录是SolverState类的一个实例。SolverState备忘录包含一些描述ConstraintSolver的内部等式和变量当前状态的数据结构
3. 此后当用户取消移动操作时，编辑器将SolverState备忘录送回给ConstraintSolver
4. 根据SolverState备忘录中的信息，ConstraintSolver改变它的内部结构以精确地将它的等式和变量返回到它们各自先前的状态。

这一方案允许ConstraintSolver把恢复先前状态所需的信息交给其他的对象，而又不暴露它的内部结构和表示。

结构图

