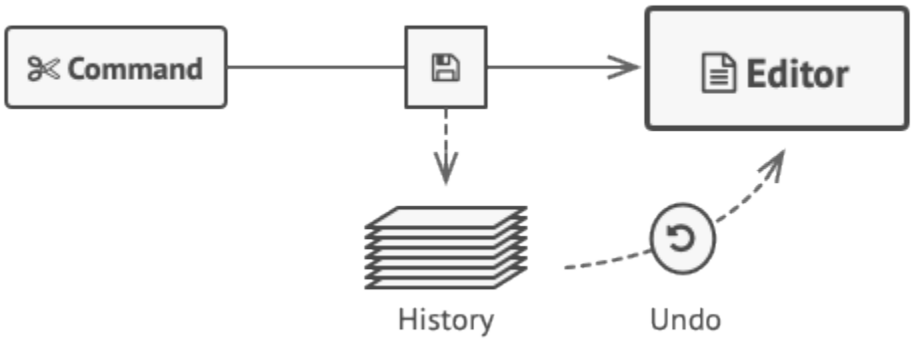
第4章 备忘录模式

提出问题

> 一款成熟的软件应当允许用户取消不确定的操作或从错误状态中恢复过来，例如文档编辑器。因此，程序必须提供必要的检查点和取消机制。在适当的时机程序需要检查对象的状态，如果有必要将状态信息保存在某处，以使对象有机会恢复到它们先前的状态。



模式名称

备忘录模式（Memento）

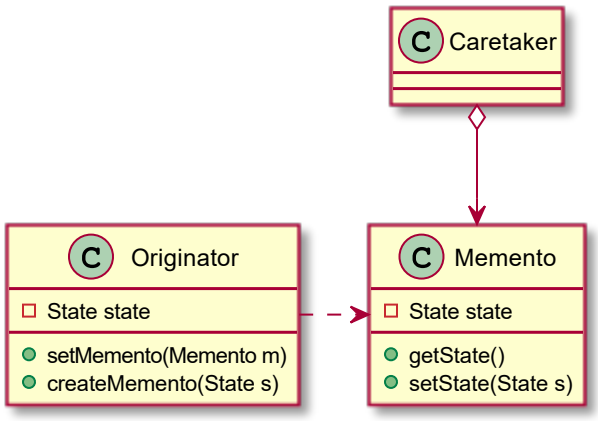
设计意图

备忘录模式在不破坏封装的前提下，捕获一个对象的内部状态，并在对象之外保存这个状态以便能将对象恢复到原先的状态。

> Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

设计结构

类图



参与者

- 发起者（Originator）：创建保存自己内部状态的备忘录对象；利用备忘录恢复自己之前的状态。
- 备忘录（Memento）：存储发起者的部分或全部状态；防止发起者之外的对象访问保存的状态。
- 管理员（Caretaker）：负责存储和管理备忘录，无法对备忘录内容进行访问。

实现

发起者实现：

```
public class Originator {
    private String state;

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public Memento createMemento() {
        return new Memento(this);
    }

    public void setMemento(Memento memento) {
        state = memento.getState();
    }

    public void display() {
        System.out.println(state);
    }
}
```

备忘录实现：

```
public class Memento {
    private String state;

    public Memento(Originator originator) {
        this.state = originator.getState();
    }

    public String getState() {
        return state;
    }

}
```

管理者实现：

```
public class Caretaker {
    private Memento memento;

    public Memento getMemento() {
        return memento;
    }

    public void setMemento(Memento memento) {
        this.memento = memento;
    }
}
```

客户端实现：

```
public class Client {

    public static void main(String[] args) {
        Originator o = new Originator();
        o.setState("S1");
        o.display();

        Caretaker c = new Caretaker();
        c.setMemento(o.createMemento());

        o.setState("S2");
        o.display();

        o.setMemento(c.getMemento());
        o.display();
    }

}
```

思考：上面实现中`Memento`的方法全部是`public`，违背了备忘录模式要求，如何避免这种情况?

效果与适用性

优点

- 简化发起人职责，实现信息封装，客户端无须关心状态保存细节；
- 提供状态恢复功能。

缺点

- 消耗额外的存储资源

适用性

- 需要保存历史快照。
- 在对象之外保存状态，并且不破坏封装性。

扩展案例

草稿箱

> 假设一个编辑器（Editor）支持草稿箱功能，编写文章的过程中能将文章保存到草稿箱中用于恢复。程序运行效果如下：

```
标题：设计模式1
内容：设计模式有三种类型
>>>>保存
标题：设计模式2
内容：设计模式有三种类型：创建型模式
>>>>保存
标题：设计模式3
内容：设计模式有三种类型：创建型模式、结构型模式
>>>>撤销
标题：设计模式2
内容：设计模式有三种类型：创建型模式
>>>>撤销
标题：设计模式1
内容：设计模式有三种类型
```

编辑器实现：

```
public class Editor {
    private String title;
    private String content;
    public TextMemento saveText() {
        return new TextMemento(this.title, this.content);
    }
    public void restore(TextMemento tm) {
        this.title = tm.getTitle();
        this.content = tm.getContent();
    }
    public void show() {
        System.out.println("标题: " + this.getTitle());
        System.out.println("内容: " + this.getContent());
    }
}
```

备忘录实现：

```
public class TextMemento {
    private String title;
    private String content;
    public TextMemento(String t, String c) {
        title = t;
        content = c;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
}
```

草稿箱实现：

```
public class DraftBox {
    private final Stack<TextMemento> box = new Stack<TextMemento>();
    public TextMemento get() {
        return box.pop();
    }
    public void add(TextMemento tm) {
        box.push(tm);
    }
}
```

客户端实现：

```
public class Client {
    public static void main(String[] args) {
        DraftBox box = new DraftBox();

        Editor editor = new Editor();

        // 第一次修改保存
        editor.setTitle("设计模式1");
        editor.setContent("设计模式有三种类型");
        editor.show();
        box.add(editor.saveText());

        // 第二次修改保存
        editor.setTitle("设计模式2");
        editor.setContent("设计模式有三种类型：创建型模式");
        editor.show();
        box.add(editor.saveText());

        // 第二次修改，未保存
        editor.setTitle("设计模式3");
        editor.setContent("设计模式有三种类型：创建型模式、结构型模式");
        editor.show();

        // 第一次撤销
        editor.restore(box.get());
        editor.show();
        // 第二次撤销
        editor.restore(box.get());
        editor.show();
    }
}
```