

- 设计模式
- 设计模式简介
- 工厂模式
- 抽象工厂模式
- 单例模式
- 建造者模式
- 原型模式
- 适配器模式
- 桥接模式
- 过滤器模式
- 组合模式
- 装饰器模式
- 外观模式
- 享元模式
- 代理模式
- 责任链模式
- 命令模式
- 解释器模式
- 迭代器模式
- 中介者模式
- 备忘录模式
- 观察者模式
- 状态模式
- 空对象模式
- 策略模式
- 模板模式
- 访问者模式
- MVC 模式

## 备忘录模式

备忘录模式（Memento Pattern）保存一个对象的某个状态，以便在适当的时候恢复对象。备忘录模式属于行为型模式。

### 介绍

**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。

**主要解决：**所谓备忘录模式就是在不破坏封装的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，这样可以在以后将对象恢复到原先保存的状态。

**何时使用：**很多时候我们总是需要记录一个对象的内部状态，这样做的目的就是为了让用户取消不确定或者错误的操作，能够恢复到他原先的状态，使得他有"后悔药"可吃。

**如何解决：**通过一个备忘录类专门存储对象状态。

**关键代码：**客户不与备忘录类耦合，与备忘录管理类耦合。

**应用实例：** 1、后悔药。 2、打游戏时的存档。 3、Windows 里的 ctrl + z。 4、IE 中的后退。 5、数据库的事务管理。

**优点：** 1、给用户提供了一种可以恢复状态的机制，可以使用户能够比较方便地回到某个历史的状态。 2、实现了信息的封装，使得用户不需要关心状态的保存细节。

**缺点：**消耗资源。如果类的成员变量过多，势必会占用比较大的资源，而且每一次保存都会消耗一定的内存。

**使用场景：** 1、需要保存/恢复数据的相关状态场景。 2、提供一个可回滚的操作。

**注意事项：** 1、为了符合迪米特原则，还要增加一个管理备忘录的类。 2、为了节约内存，可使用原型模式+备忘录模式。

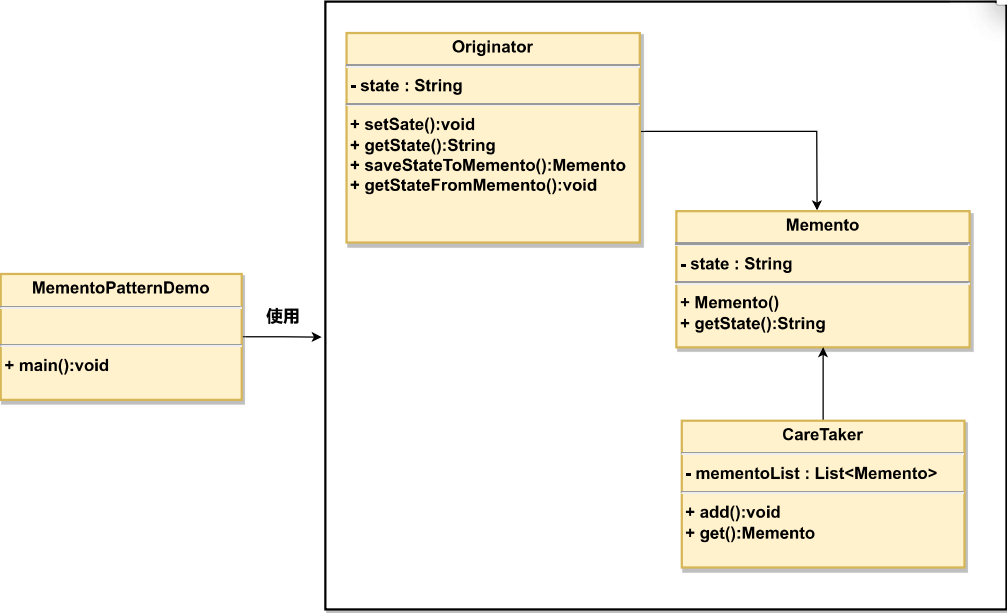
### 实现

备忘录模式使用三个类 *Memento*、*Originator* 和 *CareTaker*。*Memento* 包含了要被恢复的对象的内部状态。*Originator* 创建并在 *Memento* 对象中存储状态。*Caretaker* 对象负责从 *Memento* 中恢复对象的状态。

*MementoPatternDemo*，我们的演示类使用 *CareTaker* 和 *Originator* 对象来显示对象的状态恢复。

- HTML / CSS
- JavaScript
- 服务端
- 数据库
- 数据分析
- 移动端
- XML 教程
- ASP.NET
- Web Service
- 开发工具
- 网站建设





步骤 1

创建 Memento 类。

**Memento.java**

```
public class Memento {
    private String state;

    public Memento(String state){
        this.state = state;
    }

    public String getState(){
        return state;
    }
}
```

步骤 2

创建 Originator 类。

**Originator.java**

```
public class Originator {
    private String state;

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

    public String getState(){
        return state;
    }

    public Memento saveStateToMemento(){
        return new Memento(state);
    }

    public void getStateFromMemento(Memento Memento){
        state = Memento.getState();
    }
}
```

### 步骤 3

创建 CareTaker 类。

#### CareTaker.java

```
import java.util.ArrayList;
import java.util.List;

public class CareTaker {
    private List<Memento> mementoList = new ArrayList<Memento>();

    public void add(Memento state){
        mementoList.add(state);
    }

    public Memento get(int index){
        return mementoList.get(index);
    }
}
```

### 步骤 4

使用 CareTaker 和 Originator 对象。

#### MementoPatternDemo.java

```
public class MementoPatternDemo {
    public static void main(String[] args) {
        Originator originator = new Originator();
        CareTaker careTaker = new CareTaker();
        originator.setState("State #1");
        originator.setState("State #2");
        careTaker.add(originator.saveStateToMemento());
        originator.setState("State #3");
        careTaker.add(originator.saveStateToMemento());
        originator.setState("State #4");

        System.out.println("Current State: " + originator.getState());
        originator.getStateFromMemento(careTaker.get(0));
        System.out.println("First saved State: " + originator.getState());
        originator.getStateFromMemento(careTaker.get(1));
        System.out.println("Second saved State: " + originator.getState());
    }
}
```

### 步骤 5

验证输出。

```
Current State: State #4
First saved State: State #2
Second saved State: State #3
```





Python 方式:

54

```
# Memento Pattern with Python Code
from abc import abstractmethod, ABCMeta

# 创建Memento类
class Memento():
    _state = ""
    def __init__(self, strState):
        self._state = strState
    def getState(self):
        return self._state

# 创建Originator类
class Originator():
    _state = ""
    def setState(self, strState):
        self._state = strState
    def getState(self):
        return self._state
    def saveStateToMemento(self):
        return Memento(self._state)
    def getStateFromMemento(self, inMemento):
        self._state = inMemento.getState()

# 创建CareTaker类
class CareTaker():
    _mementoList = []
    def add(self, inMemento):
        self._mementoList.append(inMemento)
    def get(self, inIndex):
        return self._mementoList[inIndex]

# 调用输出
if __name__ == '__main__':
    originator = Originator()
    careTaker = CareTaker()
    originator.setState("State #1")
    originator.setState("State #2")
    careTaker.add(originator.saveStateToMemento())
    originator.setState("State #3")
    careTaker.add(originator.saveStateToMemento())
    originator.setState("State #4")

    print("Current State: " + originator.getState())
    originator.getStateFromMemento(careTaker.get(0))
    print("First saved State: " + originator.getState())
    originator.getStateFromMemento(careTaker.get(1))
    print("First saved State: " + originator.getState())
```



在线实例

- [HTML 实例](#)
- [CSS 实例](#)
- [JavaScript 实例](#)
- [Ajax 实例](#)
- [jQuery 实例](#)
- [XML 实例](#)
- [Java 实例](#)

字符集&工具

- [HTML 字符集设置](#)
- [HTML ASCII 字符集](#)
- [JS 混淆/加密](#)
- [PNG/JPEG 图片压缩](#)
- [HTML 拾色器](#)
- [JSON 格式化工具](#)
- [随机数生成器](#)

最新更新

- [Vue3 创建单文件...](#)
- [Vue3 指令](#)
- [Matplotlib imre...](#)
- [Matplotlib imsa...](#)
- [Matplotlib imsh...](#)
- [Matplotlib 直方图](#)
- [Python object\(\)...](#)

站点信息

- [意见反馈](#)
- [免责声明](#)
- [关于我们](#)
- [文章归档](#)

关注微信

