备忘录模式

2020年2月20日 9:30

1. 概念

保存一个对象的某个状态，以便在适当的时候回恢复对象，也叫做快照模式。例如word文档的撤回操作，游戏中存档等等。

2. UML结构图

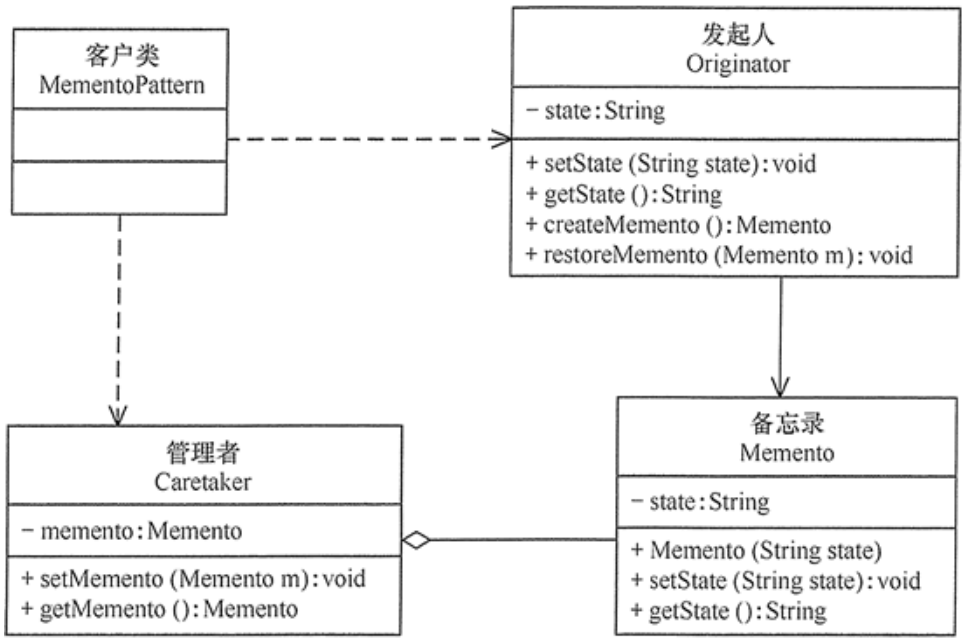

图2.1 备忘录模式UML结构图

1. 发起人（Originator）角色：记录当前时刻的内部状态信息，提供创建备忘录和恢复备忘录数据的功能，实现其他业务功能，它可以访问备忘录里的所有信息。
2. 备忘录（Memento）角色：负责存储发起人的内部状态，在需要的时候提供这些内部状态给发起人。
3. 管理者（Caretaker）角色：对备忘录进行管理，提供保存与获取备忘录的功能，但其不能对备忘录的内容进行访问与修改。

4. 备忘录模式示例

某在线课程网站提供了在线编写手记功能，进行编写时提供了撤回操作。采用备忘录模式进行设计。

1. 发起人类

```
public class Article {
    private String title;
    private String content;
    private String imgs;

    public Article(String title, String content, String imgs) {
        this.title = title;
        this.content = content;
        this.imgs = imgs;
    }

    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;
    }

    public String getImgs() {
        return imgs;
    }

    public void setImgs(String imgs) {
        this.imgs = imgs;
    }

    // 把Article转换为ArticleMemento类型，然后保存到ArticleMementoManager中
    public ArticleMemento saveToMemento(){
        ArticleMemento articleMemento = new
ArticleMemento(this.title,this.content,this.imgs);
        return articleMemento;
    }

    // 从ArticleMementoManager得到,然后撤回
    public void undoFromMemento(ArticleMemento articleMemento){
        this.title = articleMemento.getTitle();
        this.content = articleMemento.getContent();
        this.imgs = articleMemento.getImgs();
    }
}
```

2. 备忘录类

```
public class ArticleMemento {
    private String title;
    private String content;
    private String imgs;

    public ArticleMemento(String title, String content, String imgs) {
        this.title = title;
        this.content = content;
        this.imgs = imgs;
    }

    public String getTitle() {
        return title;
    }

    public String getContent() {
        return content;
    }

    public String getImgs() {
        return imgs;
    }
}
```

3. 管理者类

```
public class ArticleMementoManager {
    private final Stack<ArticleMemento> ARTICLE_MEMENTO_STACK = new Stack<ArticleMemento>();

    public ArticleMemento getMemento(){
        ArticleMemento articleMemento = ARTICLE_MEMENTO_STACK.pop();
        return articleMemento;
    }

    public void addMemento(ArticleMemento articleMemento){
        ARTICLE_MEMENTO_STACK.push(articleMemento);
    }
}
```

4. 客户端

```
    @Test
    public void mementoTest(){
        ArticleMementoManager articleMementoManager = new ArticleMementoManager();
        Article article = new Article("设计模式","学习设计模式需要注意的十件事","DesignPattern.jpg");
        System.out.println("最新的手记标题为: " + article.getTitle() + ",内容为: " +
article.getContent() + ",图片为: " + article.getImgs());

        // 保存为手记快照
        ArticleMemento articleMemento = article.saveToMemento();
        articleMementoManager.addMemento(articleMemento);
        System.out.println("===== 手记暂存成功！=====");

        // 修改内容
```

```
        System.out.println("===== 修改手记内容 =====");
        article.setTitle("设计模式注意事项");
        article.setContent("学习设计模式需要注意的五件事");
        System.out.println("修改后手记标题为: " + article.getTitle() + ",内容为: " +
article.getContent() + ",图片为: " +
article.getImgs());
        System.out.println("===== 修改手记完成 =====");

        // 进行恢复
        articleMemento = articleMementoManager.getMemento();
        article.undoFromMemento(articleMemento);
        System.out.println("===== 恢复手记成功 =====");
        System.out.println("恢复后手记标题为: " + article.getTitle() + ",内容为: "
+ article.getContent() + ",图片为: " + article.getImgs());
    }
```

5. 结果

```
最新的手记标题为: 设计模式,内容为: 学习设计模式需要注意的十件事,图片为: DesignPattern.jpg
===== 手记暂存成功! =====

===== 修改手记内容 =====
修改后手记标题为: 设计模式注意事项,内容为: 学习设计模式需要注意的五件事,图片为: DesignPattern.jpg
===== 修改手记完成 =====

===== 恢复手记成功 =====
恢复后手记标题为: 设计模式,内容为: 学习设计模式需要注意的十件事,图片为: DesignPattern.jpg
```

5. 优/缺点

1. 优点

- 为用户提供一种可恢复机制。
- 存档信息的封装。除了创建它的发起人之外，其他对象都不能访问这些状态信息。
- 简化了发起人类。发起人不需要管理和保存其内部状态的各个备份，所有状态信息都保存在备忘录中，并由管理者进行管理，这符合单一职责原则。

2. 缺点

- 如果要保存的内部状态信息过多或更新特别平凡，资源占用过多。

6. 适用场景

1. 保存及恢复数据相关业务场景。
2. 后悔时，想恢复到之前的状态。

7. 模式扩展

在备忘录模式中，通过定义“备忘录”来备份“发起人”的信息，而原型模式的 clone() 方法具有自备份功能，所以，如果让发起人实现 Cloneable 接口就有备份自己的功能，这时可以删除备忘录类，其结构图如下所示：

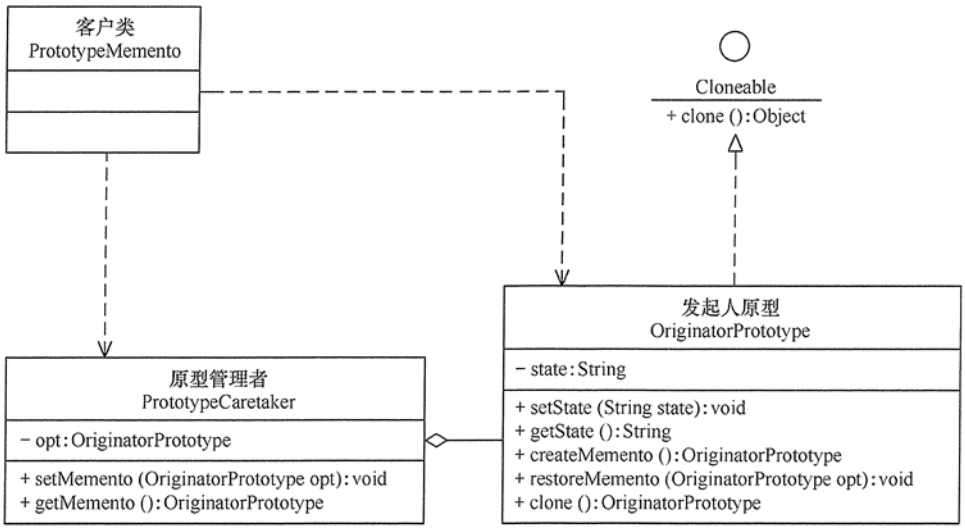

图7.1 备忘录模式扩展

1. 参考资料

[1] http://c.biancheng.net/view/1400.html