## 第22章 甲方要求改回第一版——备忘录模式

## 22.1 来自设计师的无奈

熊小猫：兔小白，早就到下班时间了，你怎么还没回家？

兔小白：今天收到一个需求变更，我想开发完再走。哎！要是软件开发永远没有需求变更该多好！

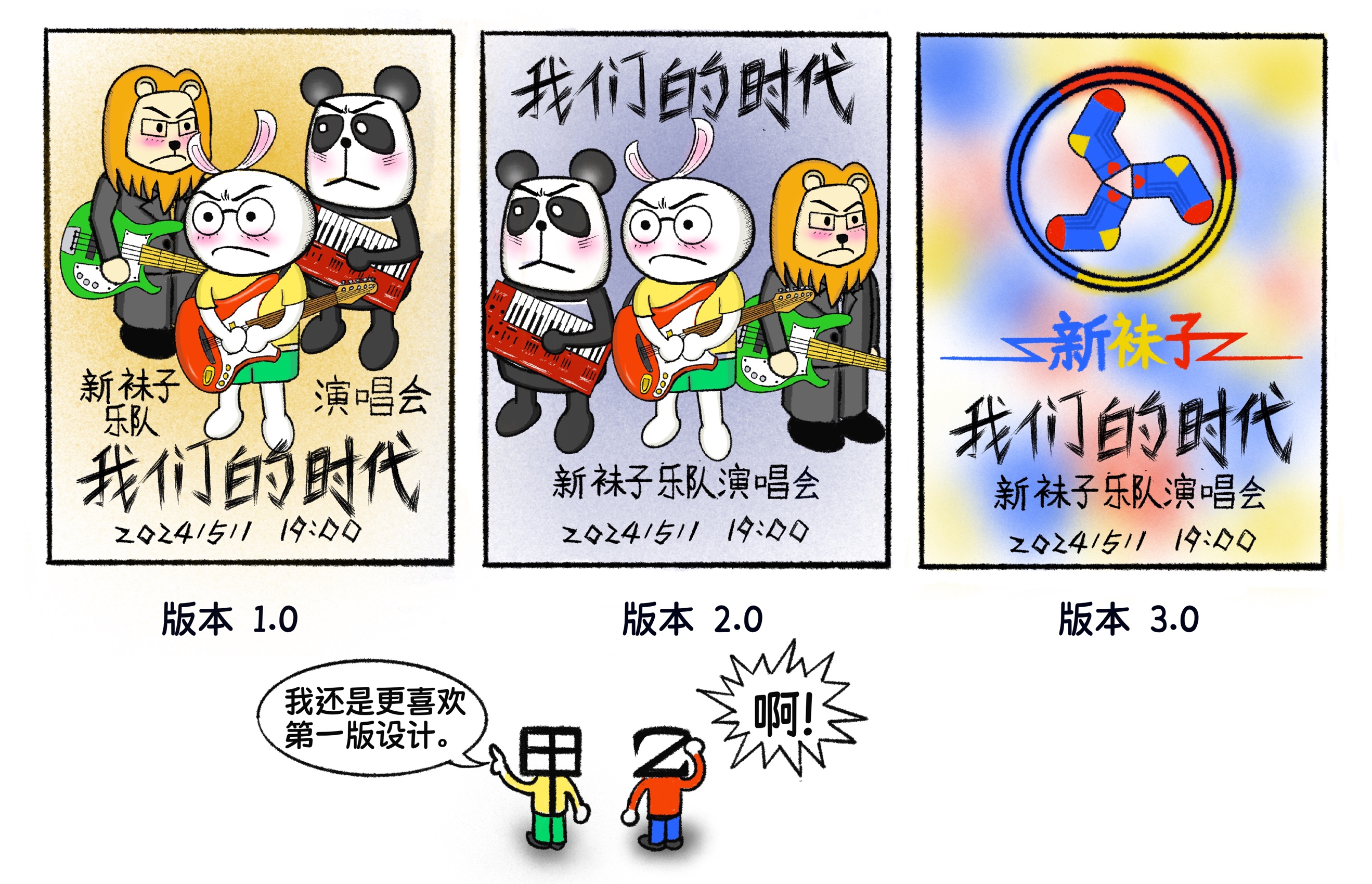
熊小猫：你的这个愿望太理想化了。如此复杂的软件系统，想要在需求分析阶段做到毫无纰漏，是不可能实现的。软件工程的一切方法论都只能降低需求变更的可能性。软件需求变更还算比较好处理，只要程序设计的灵活，一般不难应对。

兔小白：不知道其他行业有没有需求变更，又是怎样处理的。

熊小猫：当然有了，我有一位平面设计师朋友，工作是设计演出海报。他曾经和我吐槽，一张海报改了十多版。甲方最后说，看了这么多版设计，还是觉得第一版最好。

兔小白：晕倒......你朋友后面白忙活了。

熊小猫：这倒没什么，起码甲方有满意的设计。但让他吐血的是，由于改了太多版本，他已经找不到第一个版本的源文件。



兔小白：如果设计师使用Git[[1]](#footnote-1)1，按照软件开发“小步快跑[[2]](#footnote-2)2”的原则，绝对不会漏掉任何一个版本。

熊小猫：这是好多年前的事情了，再说设计师哪会用Git。他想备份的时候，会拷贝一份，改名留存。。

兔小白：什么工具都不用，纯手工操作，难怪会漏掉某个版本。

熊小猫：手工操作既麻烦，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快照保存下来就行啦！

熊小猫：是要保存快照，但怎么照，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，我再给你讲解。需求是这样的，一张海报包含标题、图片、演出时间这几种元素。海报在设计过程中可以随时存档，并且可以读取历史存档。

兔小白：放心，我设计的程序一定能解决你设计师朋友的痛点。

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，Poster实现了Cloneable接口。

public class Poster implements Cloneable {  
    private String picture;  
    private String title;  
    private String time;  
​  
    public Poster(String picture, String title, String time) {  
        this.picture = picture;  
        this.title = title;  
        this.time = time;  
   }  
​  
    public void setPicture(String picture) {  
        this.picture = picture;  
   }  
​  
    public void setTitle(String title) {  
        this.title = title;  
   }  
​  
    public void setTime(String time) {  
        this.time = time;  
   }  
​  
    public void print() {  
        System.out.println("---------------------");  
        System.out.println("标题：" + title);  
        System.out.println("背景图片：" + picture);  
        System.out.println("时间：" + time);  
        System.out.println("---------------------");  
   }  
​  
    public Poster clone() {  
        Poster poster = null;  
        try {  
            poster = (Poster) super.clone();  
       } catch (CloneNotSupportedException e) {  
            e.printStackTrace();  
       }  
        return poster;  
   }  
}

Caretaker类是存档管理器，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。

public class Caretaker {  
    private int size;  
    private Map<String, Poster> posters = new HashMap<>();  
​  
    public Caretaker(int size) {  
        this.size = size;  
   }  
​  
    public Poster getPost(String key) {  
        return posters.get(key);  
   }  
​  
    public void removePost(String key){  
        posters.remove(key);  
   }  
​  
    public void setPoster(String key, Poster poster) {  
        if (posters.size() < size) {  
            posters.put(key, poster);  
       }  
   }  
}

在客户端代码中，先设计了一版海报，进行存档。然后更改了海报的图片，再次存档。最后使用Caretaker读取第一版存档，进行输出。

Caretaker caretaker = new Caretaker(5);  
  
Poster poster = new Poster("乐队正面照片", "我们的时代", "2024年5月1日 19：00");  
caretaker.setPoster("v1", poster.clone());  
poster.print();  
​  
poster.setPicture("乐队Logo图片");  
caretaker.setPoster("v2", poster.clone());  
poster.print();  
​  
poster = caretaker.getPost("v1");  
poster.print();

从输出结果可以看到，第一版存档被正确保存和读取，背景图片被恢复为乐队正面照片。

---------------------  
标题：我们的时代  
背景图片：乐队正面照片  
时间：2024年5月1日 19：00  
---------------------  
---------------------  
标题：我们的时代  
背景图片：乐队Logo图片  
时间：2024年5月1日 19：00  
---------------------  
---------------------  
标题：我们的时代  
背景图片：乐队正面照片  
时间：2024年5月1日 19：00  
---------------------

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：组成海报的元素只有图片、标题和时间。

熊小猫：所以在存档的时候，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，将这三个属性的存档值，恢复到Poster对象中。Poster对象就可以还原到存档时的状态。

兔小白：现在存档保存的是整个Poster对象，确实没有必要。

## 22.3 存档瘦身，只留数据

熊小猫：这里就是要优化的地方，可以将存档需要的属性分离出来，存档时仅保存属性值。

兔小白：我明白了，将Poster存档相关的属性封装到另外一个类中。这个类用来保存存档数据。

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Poster的存档数据，也就是它的picture、title、time三个属性值，并提供读取方法。

public class Memento {  
    private final String picture;  
    private final String title;  
    private final String time;  
​  
    public Memento(String picture, String title, String time) {  
        this.picture = picture;  
        this.title = title;  
        this.time = time;  
   }  
​  
    public String getPicture() {  
        return picture;  
   }  
​  
    public String getTitle() {  
        return title;  
   }  
​  
    public String getTime() {  
        return time;  
   }  
}

Poster增加了两个方法，createMemento方法用于生成存档Memento对象，setMemento方法读取Memento对象的数据，将自己恢复到存档时的状态。Clone方法不再需要，直接移除掉。

public class Poster {  
    private String picture;  
    private String title;  
    private String time;  
​  
    public Poster(String picture, String title, String time) {  
        this.picture = picture;  
        this.title = title;  
        this.time = time;  
   }  
​  
    public void setPicture(String picture) {  
        this.picture = picture;  
   }  
​  
    public void setTitle(String title) {  
        this.title = title;  
   }  
​  
    public void setTime(String time) {  
        this.time = time;  
   }  
​  
    public void print() {  
        System.out.println("---------------------");  
        System.out.println("标题：" + title);  
        System.out.println("背景图片：" + picture);  
        System.out.println("时间：" + time);  
        System.out.println("---------------------");  
   }  
​  
    public Memento createMemento() {  
        return new Memento(picture, title, time);  
   }  
​  
    public void setMemento(Memento memento) {  
        this.picture = memento.getPicture();  
        this.title = memento.getTitle();  
        this.time = memento.getTime();  
   }  
}

由于存档对象从Poster变为了Memento，Caretaker修改为维护Memento对象。

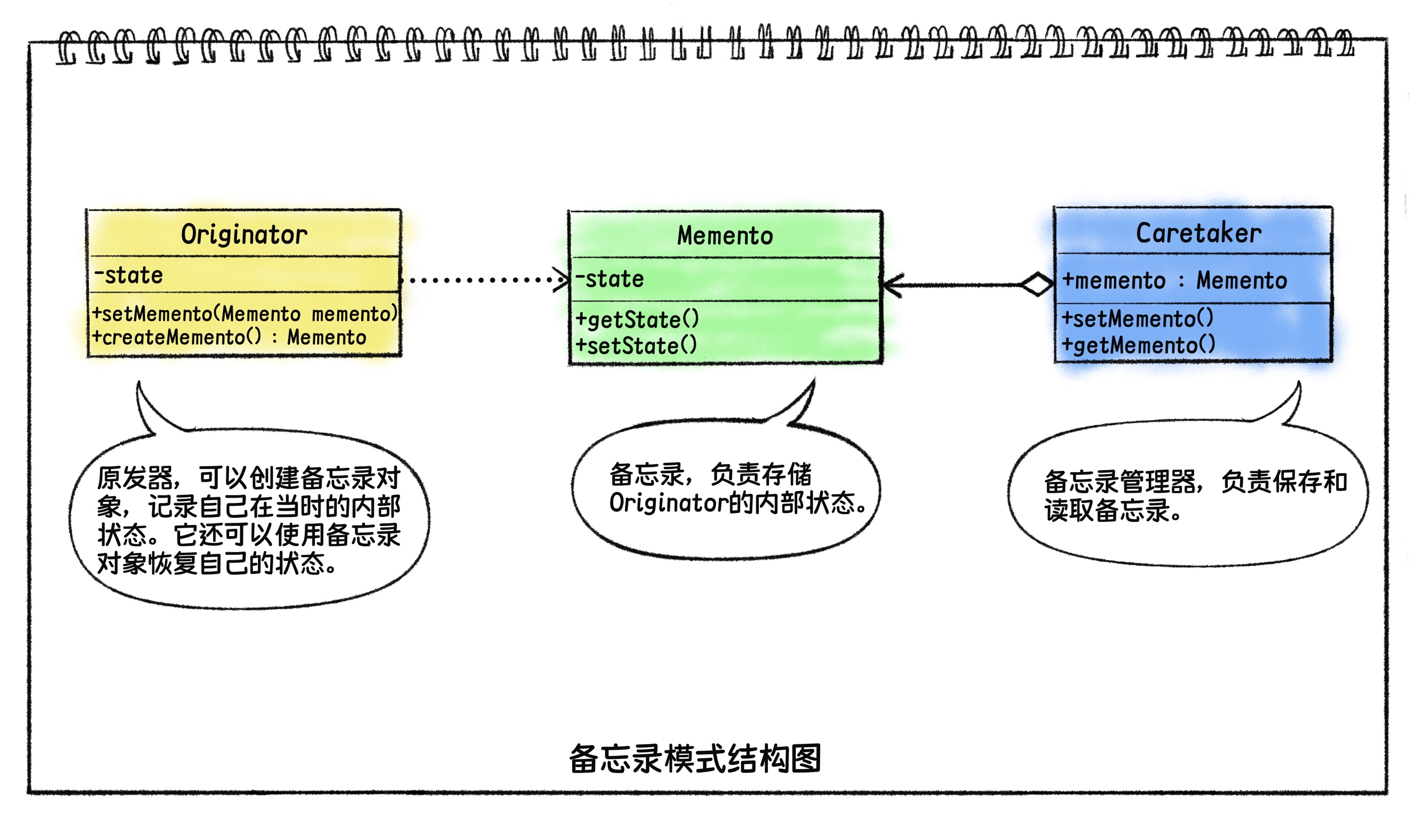
public class Caretaker {  
​  
    private final int size;  
    private final Map<String, Memento> mementos = new HashMap<>();  
​  
    public Caretaker(int size) {  
        this.size = size;  
   }  
​  
    public Memento getMemento(String key) {  
        return mementos.get(key);  
   }  
​  
    public void removeMemento(String key) {  
        mementos.remove(key);  
   }  
​  
    public void setMemento(String key, Memento memento) {  
        if (mementos.size() < size) {  
            mementos.put(key, memento);  
       }  
   }  
}

客户端代码改动不大，Poster使用从Caretaker中取得的v1版本Memento对象，恢复到v1版本状态。输出结果和之前一样，符合预期。

Caretaker caretaker = new Caretaker(5);  
​  
Poster poster = new Poster("乐队正面照片", "我们的时代", "2024年5月1日 19：00");  
caretaker.setMemento("v1", poster.createMemento());  
poster.print();  
​  
poster.setPicture("乐队Logo图片");  
caretaker.setMemento("v2", poster.createMemento());  
poster.print();  
​  
poster.setMemento(caretaker.getMemento("v1"));  
poster.print();

## 22.4 备忘录模式适用的场景

熊小猫：这版代码改的很好，其实已经符合备忘录模式的结构。下面我们来看看备忘录模式的结构图。



Originator为原发器。它是有存档需求的类，例如练习中的Poster。它可以创建备忘录对象，记录自己在某个时刻的内部状态，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，不破坏对象的封装性，将对象的内部状态保存在对象之外。适合使用备忘录模式的场景有如下几种。

1. 需要存档，并按存档恢复的场景。例如海报设计的例子、游戏存档等。
2. 可撤销的命令。命令模式可以结合备忘录模式使用，通过备忘录管理原发器对象历史状态。当命令撤销时，使用备忘录对象恢复原发器对象状态。
3. 支持回滚的操作。如果一个操作支持回滚，那么可以使用备忘录来记录操作前原发器对象的状态。一旦操作失败，发生回滚，原发器对象可以通过备忘录恢复到操作前的状态。
4. 需要记录对象状态变化的场景。如果程序想要跟踪、分析某个对象在一系列业务操作下的状态变化，可以使用备忘录模式提取关心的属性。此场景不需要恢复对象状态，可以对备忘录模式做相应简化。

兔小白：我最常用的场景是游戏存档。关键时刻先存档，否则一旦死了，就得从头再来！

熊小猫：哈哈，你这是使用了存档功能，不是使用了备忘录模式。等你以后工作中，有合适的机会再用起来吧！

1. 1 Git是一个免费开源的分布式版本控制系统。在软件开发中，Git被用于管理代码版本。 [↑](#footnote-ref-1)
2. 2 这里的小步快跑是一种敏捷开发实践，指开发人员完成一个单元测试对应的业务代码，就会向代码库提交一次代码。 [↑](#footnote-ref-2)