模板方法模式

2020年2月15日 14:32

1. 概念

定义了一个算法的骨架，并允许子类为一个或多个步骤提供实现。例如把大象放冰箱中需要三步，打开冰箱，放入大象，关闭冰箱，可为一个算法骨架。
模板方法使得子类可以在不改变算法结构的情况下，重新定义算法的某些步骤。
模板方法是基于"继承"的。

2. UML结构图

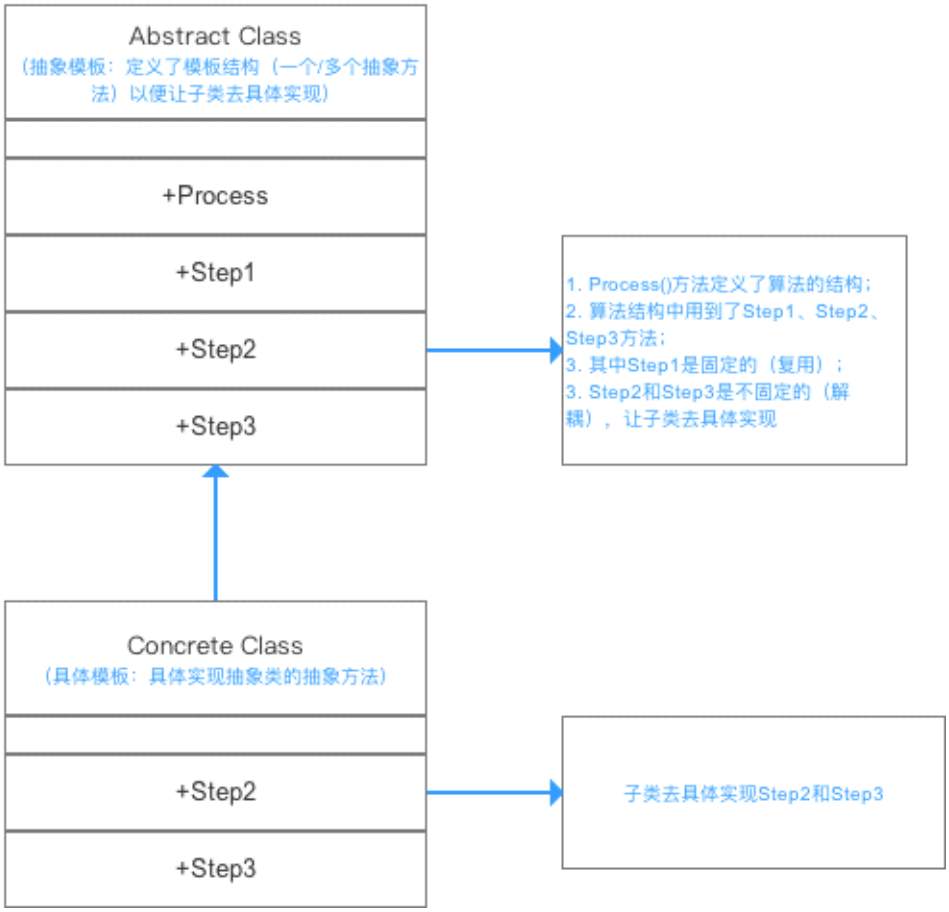

图2.1 模板方法UML结构图

1. AblstractClass（抽象类）：在抽象类中定义了一系列的操作，每个操作可以使具体的，也可以是抽象的，每个操作对应一个算法的步骤，在子类中可以对抽象类重新定义或实现这些步骤。
2. ConcreteClass（具体子类）：用于实现在父类中声明的抽象基本操作，也可以覆盖在父类中已经实现的具体基本操作。

3. 模板方法实例

在线教育网站上上传课程可看作为一个过程（算法）骨架，包括了制作PPT，制作视频，制作手记和打包课程。其中制作PPT，制作视频为算法的不变部分，打包课程为算法的可变部分，不同的课程有着不同的打包过程，而制作手记为可缺省部分，使用钩子方法。

1. 抽象类

```
public abstract class  ACourse  {
     //定义算法结构
     protected final void makeCourse(){
         this.makePPT();
         this.makeVideo();
         if (needWriteArticle()){
             this.writeArticle();
         }
         this.packageCourse();
     }

     //固定不变的行为，不希望父类重写，final修饰
     final void makePPT(){
         System.out.println("制作PPT");
     }

     final void makeVideo(){
         System.out.println("制作Video");
     }

     final void writeArticle(){
         System.out.println("编写手记");
     }

     // 钩子方法
     protected boolean needWriteArticle(){
         return false;
     }

     // 不同课程有着不同的要求，完全由子类实现
     abstract void packageCourse();
}
```

2. 具体实现类

```
public class DesignPatternCourse extends ACourse{

    @Override
    void packageCourse() {
        System.out.println("提供课程java源代码");
    }

    // 需要写手记，重写钩子方法即可
    @Override
    public boolean needWriteArticle() {
        return true;
    }
}
```

```
public class FECourse extends ACourse{
    private boolean needWriteArticle = false;
    public FECourse(boolean needWriteArticle) {
        this.needWriteArticle = needWriteArticle;
    }

    @Override
    void packageCourse() {
        System.out.println("提供课程源代码");
        System.out.println("提供课程所需要图片等资源");
    }

    // 可能FE种类更多，Vue需要手记，React不需要手记，将
    // needWriteArticle开放给应用层
    @Override
    public boolean needWriteArticle() {
        return needWriteArticle;
    }
}
```

3. 客户端

```
 @Test
    public void templateMethodTest(){
        System.out.println("后端设计模式课程开始制作");
        ACourse designPatternCourse = new DesignPatternCourse();
        designPatternCourse.makeCourse();
        System.out.println("后端设计模式课程制作完成");


        System.out.println("前端课程开始制作");
        ACourse FECourse = new FECourse(true);
        FECourse.makeCourse();
        System.out.println("前端课程制作完成");
    }
```

4. 结果

```
后端设计模式课程开始制作
制作PPT
制作Video
编写手记
提供课程java源代码
后端设计模式课程制作完成

前端课程开始制作
制作PPT
制作Video
编写手记
提供课程源代码
提供课程所需要图片等资源
前端课程制作完成
```

4. 优/缺点

1. 优点

- 提供复用性，将相同部分的代码放在抽象的父类中。
- 提供扩展性，将不同的代码放入不同的子类中，通过对子类的扩展增加新的行为。
- 符合开闭原则。

2. 缺点

- 类数目的增加。
- 增加了系统实现复杂度。
- 继承关系自身缺点，如果父类添加了新的抽象方法，所有子类都需要更改。

5. 使用场景

4. 一次性实现一个算法的不变部分，并将可变的行为留给子类来实现。例如打开冰箱门和关闭冰箱门可算为不变部分，具体放什么东西属于可变部分。
5. 各子类中公共的行为被提取出来并集中到一个公共父类中，从而避免了代码重复。

6. 扩展

1. 钩子方法：提供缺省的行为，子类可以在必要时进行扩展。

7. 参考资料

[1] https://www.jianshu.com/p/a3474f4fee57