链滴

CRUD 很无聊？一起学设计模式吧！ -- 模板模式

作者：jianzh5
原文链接：https://ld246.com/article/1574085151804
来源网站：链滴

如果在项目开发中你经常看到一个类的某些方法和其他类的方法功能相同，只有部分不同或者只有具实现不同，亦或者是你看到某些方法在多个地方都存在，有很多重复代码，这个时候你就可以拿出模设计模式了。

定义与特点

模板方法（Template Method）模式的定义如下： 定义一个操作中的算法骨架，而将算法的一些步延迟到子类中，使得子类可以不改变该算法结构的情况下重定义该算法的某些特定步骤。它是一种类为型模式。

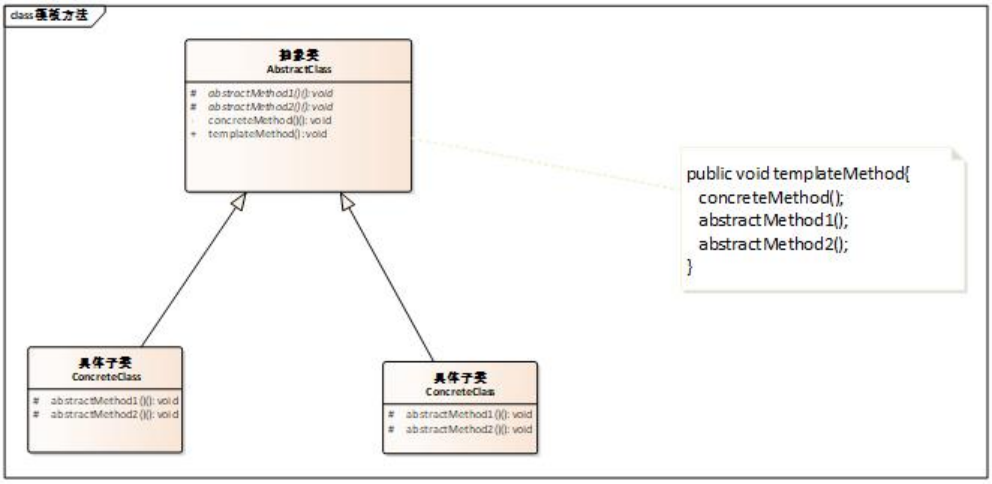
模板模式的主要优点如下：

- 它封装了不变部分，扩展可变部分。它把认为是不变部分的算法封装到父类中实现，而把可变部分法由子类继承实现，便于子类继续扩展。
- 它在父类中提取了公共的部分代码，便于代码复用。
- 部分方法是由子类实现的，因此子类可以通过扩展方式增加相应的功能，符合开闭原则。

主要缺点如下：

- 对每个不同的实现都需要定义一个子类，这会导致类的个数增加，系统更加庞大，设计也更加抽象。
- 父类中的抽象方法由子类实现，子类执行的结果会影响父类的结果，这导致一种反向的控制结构，提高了代码阅读的难度。

UML



角色定义

模板模式涉及三个角色：

● 抽象类（AbstractClass）角色：定义一个操作的算法轮廓和框架。它由一个模板方法和若干个基本法组成。

模板方法（templateMethod）：

定义了算法的骨架，按某种顺序调用其包含的基本方法，使用public修饰；

基本方法：是整个算法中的一个步骤，使用protected修饰，包含以下几个类型：

- 抽象方法：在抽象类中申明，由具体子类实现
- 具体方法：在抽象类中实现，但是子类可以继承或重写它。
- 钩子方法：在抽象类中已经实现，包括用于判断的逻辑方法和需要子类重写的空方法两种。

场景实战

我们的报销系统分为日常费用报销和差旅费用报销，报销的流程是先根据报销单上带的费用计算出报金额，然后计算出报销单中的补贴金额（若是差旅类型报销才需要计算补贴，日常报销不需要计算补），最后调用第三方接口创建流程。这个场景就适合用模板设计模式实现。

抽象类定义

定义报销流程的算法框架，算法框架使用final修饰，对于必须要子类实现的方法用abstract关键字修。

```
public abstract class AbstractReimburse {
    /**
     * 用作算法的模板
     * 定义成final，以免子模板改变算法的顺序
     */
    final void calAndCreateFlow(){
        BigDecimal totalMoney;
        BigDecimal changeMoney = calChangeMoney();
        BigDecimal subsidyMoney = BigDecimal.ZERO;
        if(hasTravel()){
```

```
            subsidyMoney = calSubsidyMoney();
        }
        totalMoney = changeMoney.add(subsidyMoney);
        createWorkeFlow(totalMoney);
    }

    /**
     * 具体方法，子类判断是否需要实现
     * @param totalMoney 报销总金额
     */
    protected void createWorkeFlow(BigDecimal totalMoney) {
        System.out.println("开始创建流程...,总报销金额:"+totalMoney);
        //todo
    }

    /**
     * 钩子方法，由子类决定是否实现,钩子可以作为条件控制，影响抽象类中的算法流程
     * 判断是否需要计算补贴
     */
    boolean hasTravel() {
        return false;
    }

    /**
     * 抽象方法,需要子类去实现
     * 返回需要报销的费用金额
     * @return 报销的费用总金额
     */
    abstract BigDecimal calChangeMoney();

    /**
     * 返回需要报销的补贴金额
     */
    abstract BigDecimal calSubsidyMoney();
}
```

具体实现类

- 差旅类报销实现逻辑

```
/**
 * 差旅类报销实现逻辑
 */
public class TravelReimburse extends AbstractReimburse{

    @Override
    BigDecimal calChangeMoney() {
        System.out.println("差旅类报销计算费用金额");
        return new BigDecimal(1000);
    }

    @Override
    BigDecimal calSubsidyMoney() {
```

```
        System.out.println("差旅类报销需要计算补贴");
        return new BigDecimal(500);
    }

    @Override
    boolean hasTravel(){
        return true;
    }
}
```

- 日常类报销实现逻辑

```
/**
 * 日常类报销实现逻辑
 */
public class DailyReimburse extends AbstractReimburse{

    @Override
    BigDecimal calChangeMoney() {
        System.out.println("日常类报销计算费用金额");
        return new BigDecimal(100);
    }

    @Override
    BigDecimal calSubsidyMoney() {
        return BigDecimal.ZERO;
    }

}
```

客户端模拟业务流程

```
public class ReimburseClient {
    public static void main(String[] args) {
        //差旅类报销处理逻辑
        TravelReimburse travelReimburse = new TravelReimburse();
        travelReimburse.calAndCreateFlow();
        System.out.println("==========================");
        //日常类报销处理逻辑
        DailyReimburse dailyReimburse = new DailyReimburse();
        dailyReimburse.calAndCreateFlow();

    }
}
```

运行结果

```
D:\development\Java\jdk1.8.0_112\bin\java.exe ...
差旅类报销计算费用金额
差旅类报销需要计算补贴
开始创建流程...,总报销金额:1500
==========================
日常类报销计算费用金额
开始创建流程...,总报销金额:100
```

应用场景

模板模式应该是众多设计模式中相对简单的一种，但是它使用的频率可一点也不低，在各种开源框架码中都可以看到它的身影，模板设计模式的应用场景主要有以下几类：

- 在多个子类中拥有相同的方法，而且逻辑相同，可以将这些方法抽出来放到一个模板抽象类中
- 程序主框架相同，仅实现细节不同时，也可以使用模板方法

tips

记得几年前电话面试的时候，面试官问我有没有用过模板设计模式，我回答说“啊，模板？你说的是fremarker吗？巴拉巴拉一大堆”，然后电话嘟嘟嘟，留我一人在风中凌乱。