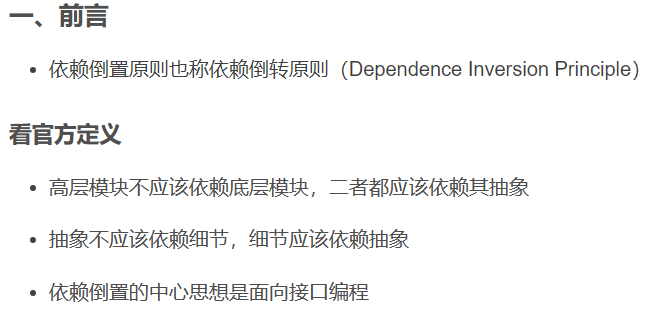
# 面向对象设计原则之五大原则

* 单一职责原则（The Single Responsibility Principle，简称SRP）
* 开放－封闭原则（The Open-Close Principle，简称OCP）
* Liskov替换原则（The Liskov Substitution，简称LSP）

里氏代换原则（LSP）：子类型必须能够替换掉它们的父类型。[ASD]

* 依赖倒置原则（The Dependency Inversion Principle，简称DIP）
* 接口隔离原则（The Interface Segregation Principle，简称ISP）

## 依赖倒置原则



### 基础知识

**1.什么是依赖？**

就是对不同模块、对象或个体之间的相互作用或相互关联的描述。

实现类与抽象之间所表达的关系。

**2.依赖关系有哪些(UML的六种依赖)？**

继承、接口实现、组合(初始化依赖),聚合(字段数组依赖), 关联(字段依赖), 依赖

(构造参数依赖)。

实现类与抽象之间所表达的关系。

**3.抽象和细节具体指代什么？**

抽象：接口、抽象类。

细节：具体实现类

**4.解决什么问题？**

通过抽象解决强依赖问题。

### 原则解释

**1.高层模块不应该依赖低层模块，二者都应该依赖其抽象**。

含义：模块与模块之间不能直接依赖，需要通过抽象进行依赖。

有点：降低模块之间的耦合度，提高系统的灵活性和可维护性

**2.抽象不应该依赖细节，细节应该依赖抽象。**

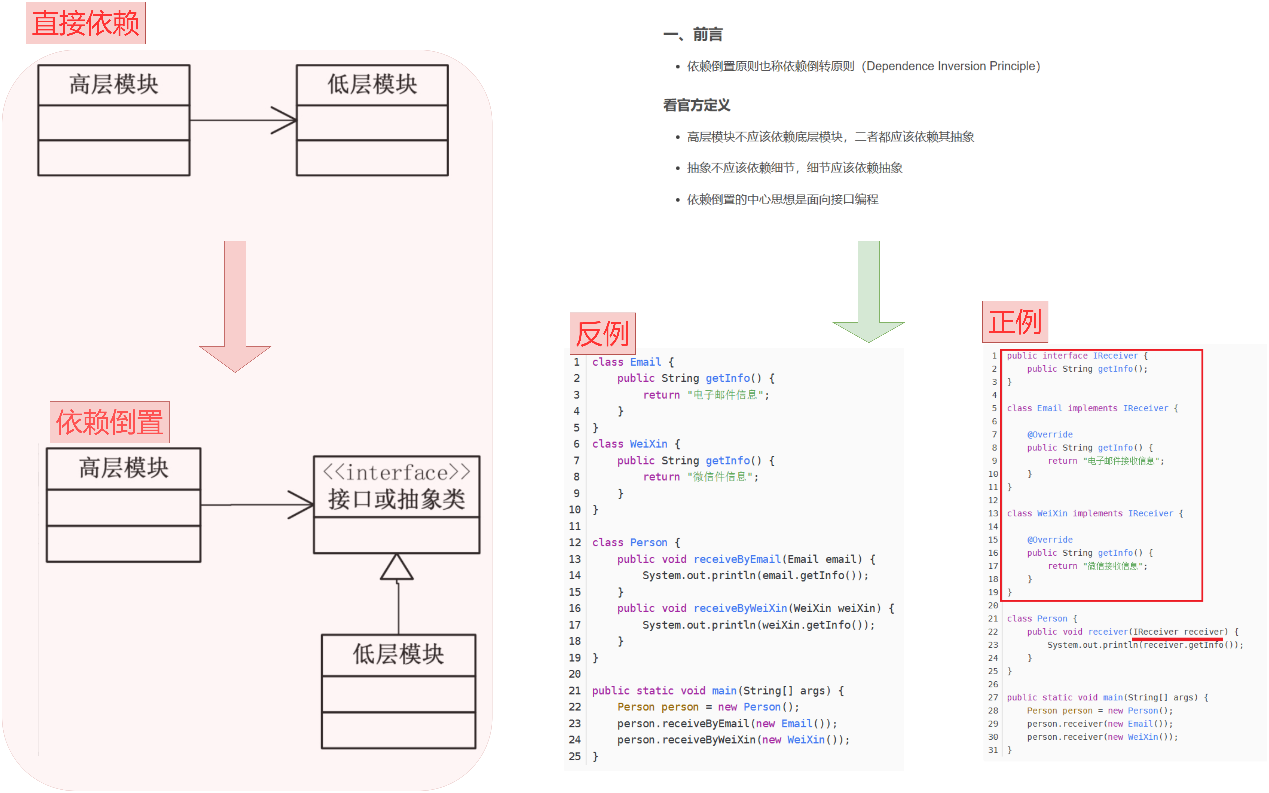
含义：***抽象类的定义与具体的实现细节无关(不能产生依赖关系)，是对事物规律***

***的一个抽象（就是定义一个抽象类或接口）。***

优点：提供一组稳定的接口，及模块间解耦.

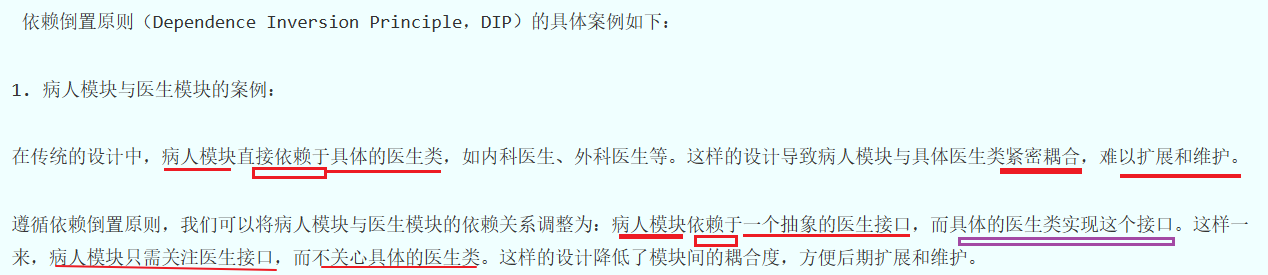
**3.依赖倒转的中心思想是面向接口编程。**

### 实现方法

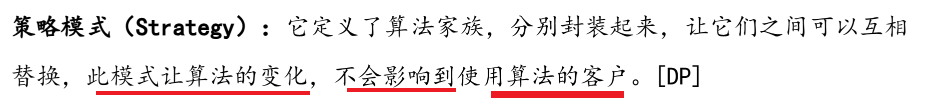


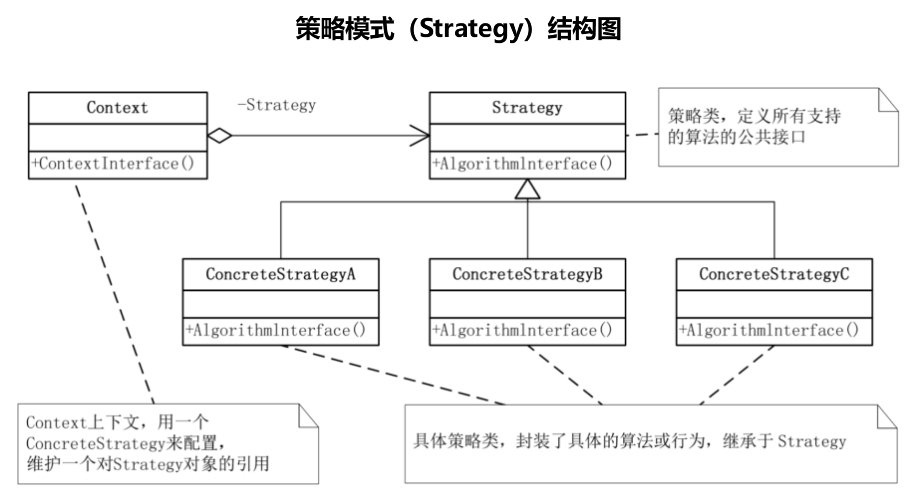
### 名字由来

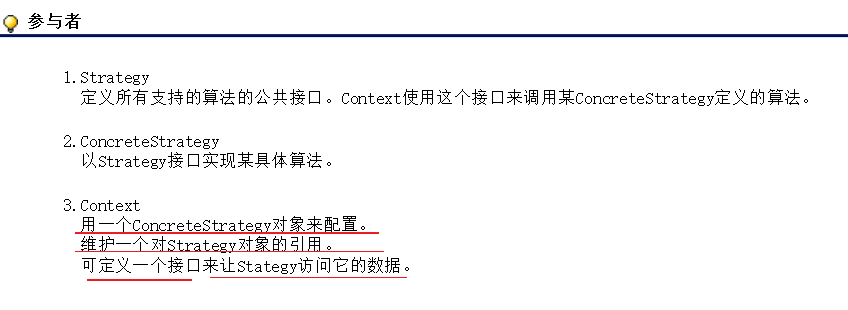
描述了依赖关系的调整。



# 策略模式



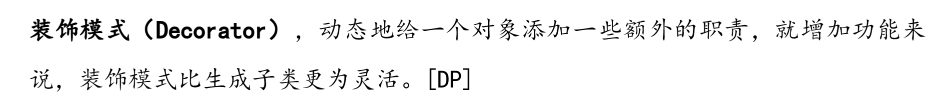




## 总结

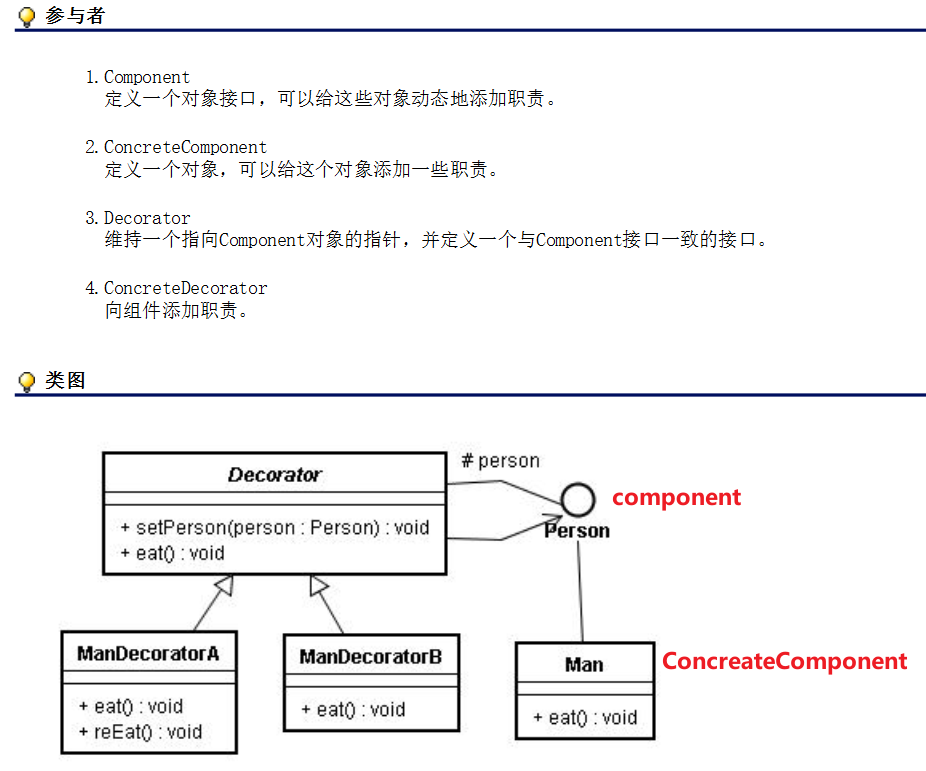
策略模式是一种定义一系列算法的方法，从概念上来看，所有**这些算法完成的都是相同的工作，只是实现不同**，它可以以相同的方式调用所有的算法，减少了各种算法类与使用算法类之间的耦

# 装饰模式

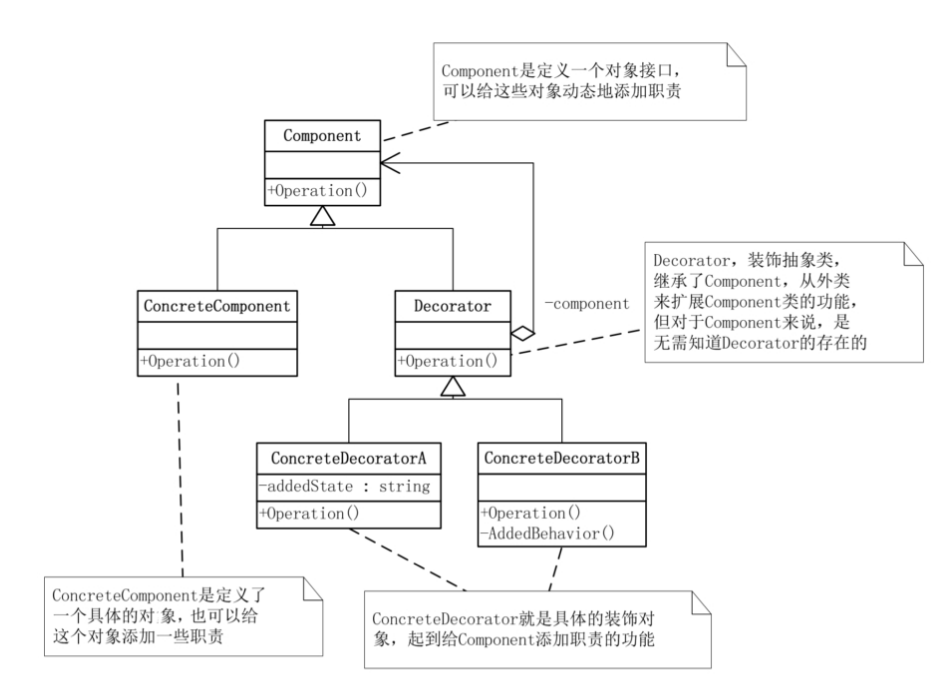


## 结构图

标准类图



参考图：下面类图不标准，标准装饰模式的结果。



## 要点

主要运用了依赖倒置原则与面向对象的继承特性，通过Decorator模块维护的指针，进行动态的功能扩展。

## 总结

**1.把类中的装饰功能从类中搬移去除，这样可以简化原有的类，降低复杂度。**

**解释：**

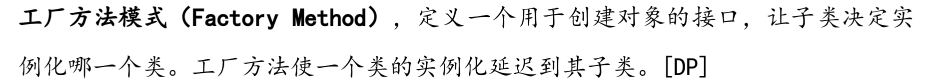
*当系统需要新功能的时候，是向旧的类中添加新的代码。这些新加的代码通常 装饰了原有类的核心职责或主要行为.这种做法的问题在于增加了主类的复杂 度.而这些新加入的东西仅仅是为了满足只在某种特定情况下才会执行的特殊 行为需要。*

2.有效地把类的核心职责和装饰功能区分开了。而且可以去除相关类中重复的装饰逻辑。

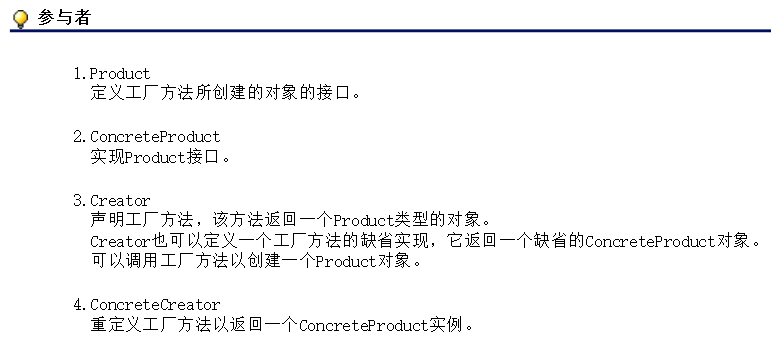
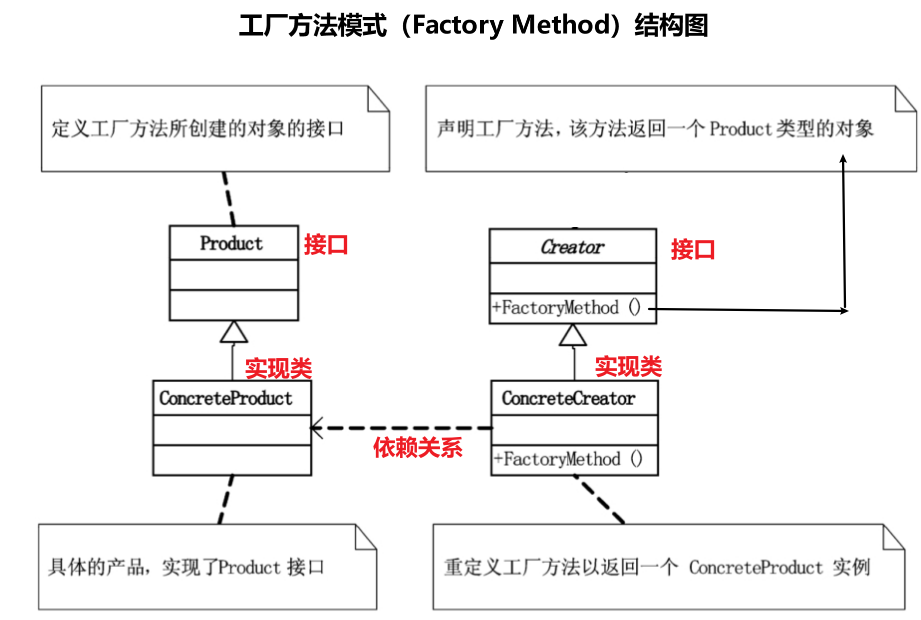
**解释：**

*装饰模式却提供了一个非常好的解决方案，它把每个要装饰的功能放在单独的 类中，并让这个类包装它所要装饰的对象，因此，当需要执行特殊行为时，客 户代码就可以在运行时根据需要有选择地、按顺序地使用装饰功能包装对象了*

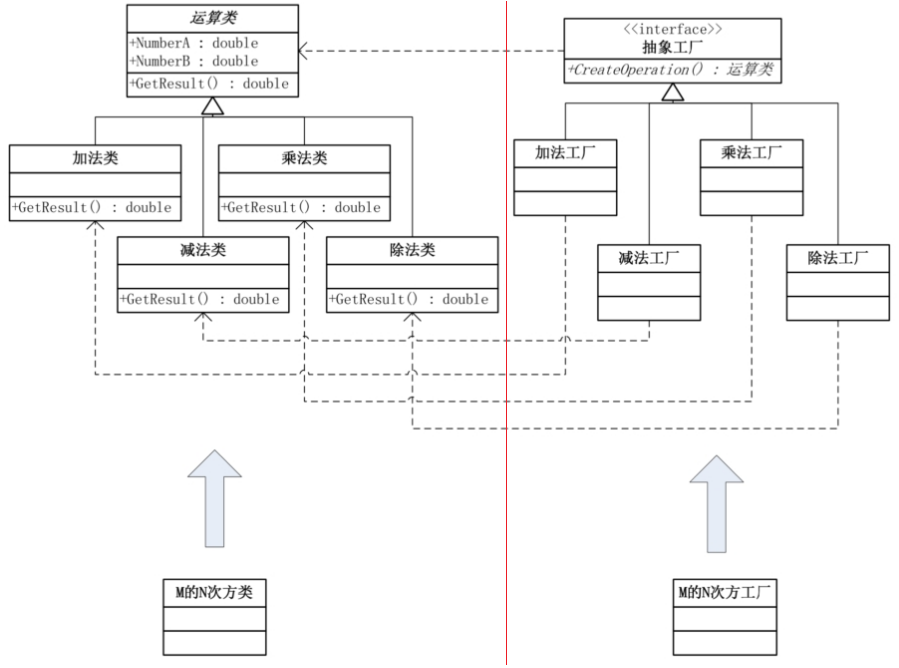
# 工厂方法模式



## 结构图



## 工厂方法模式总结

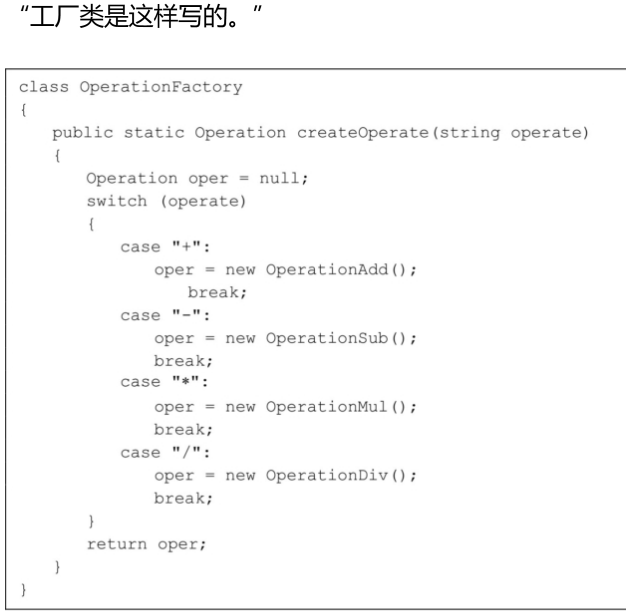
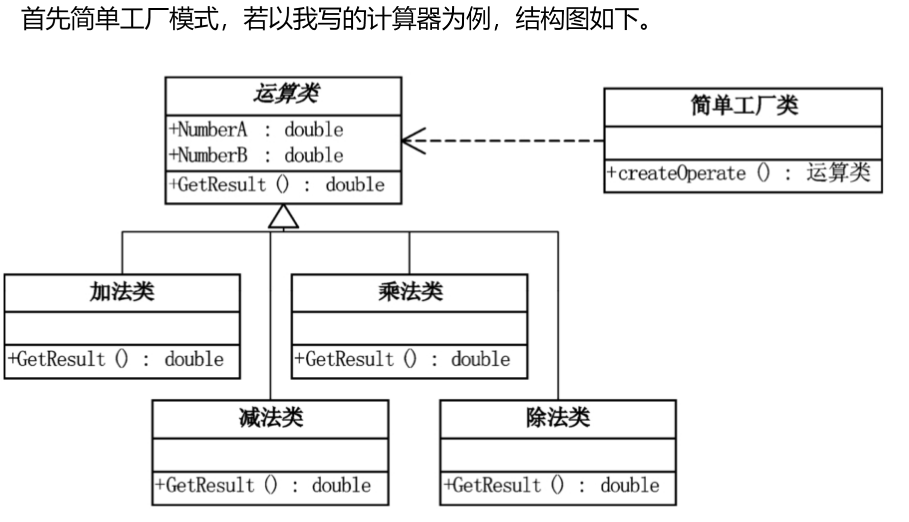


工厂方法模式实现时，**客户端 需要决定 实例化哪一个工厂 来实现运算类**，选择判断的问题还是存在的。

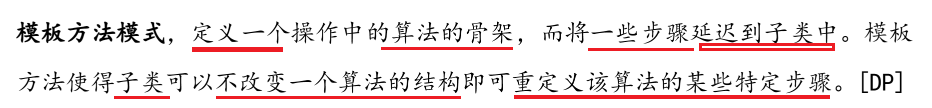
也就是说，工厂方法把简单工厂的内部逻辑判断移到了客户端代码来进行。你*想要加功能，本来是改工厂类的，而现在是修改客户端！*

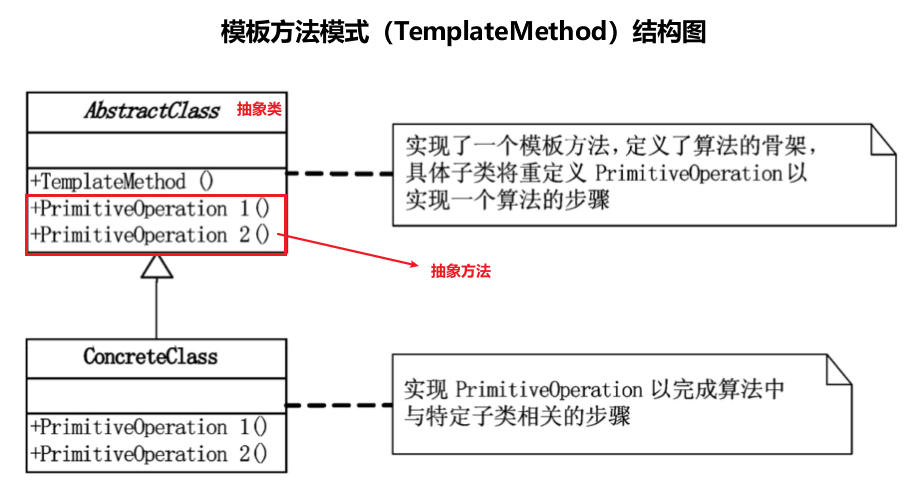
## 简单工厂总结

简单工厂模式的最大优点在于 **工厂类中包含了必要的逻辑判断**，根据客户端的选择条件动态实例化相关的类，对于客户端来说，去除了与具体产品的依赖。



# 模板方法模式





## 要点

抽象类中的模板方法是一个实例方法，它的主要作用是定义算法的骨架。具体实现细节定义为抽象方法，通过继承实现具体步骤。