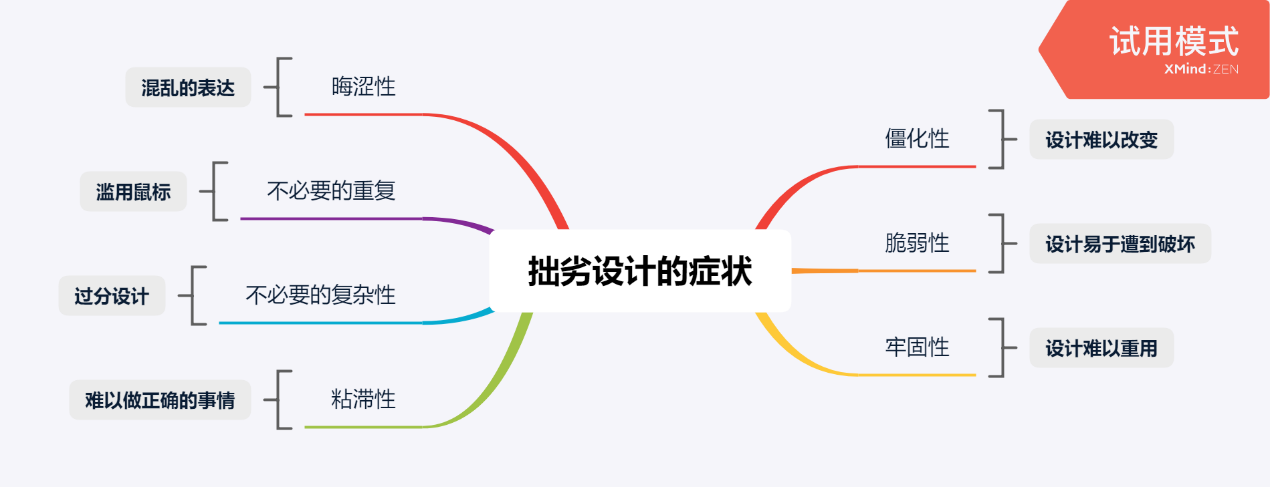
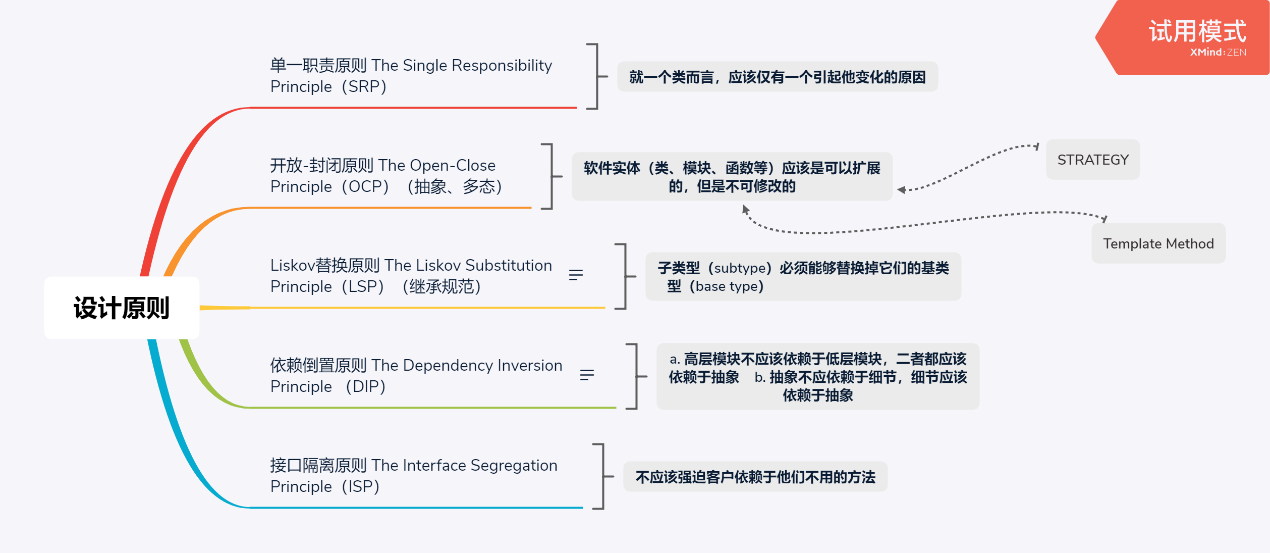
## 软件开发过程需要遵循的规则





## 什么是设计模式

**每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题解决方案的核心。**

1. 模式名称
2. 问题
3. 解决方案
4. 效果

## STRATEGY（策略）and 静态工厂（简单工厂）

**STRATEGY**

1. 意图：定义一系列的算法，把它们一个个封装起来，并且使他们可以相互替换。（该模式使得算法可以独立于使用它的客户而变化）
2. 适用性

许多类仅仅是行为有异

需要使用一个算法的不同变体

一个类定义了多种行为

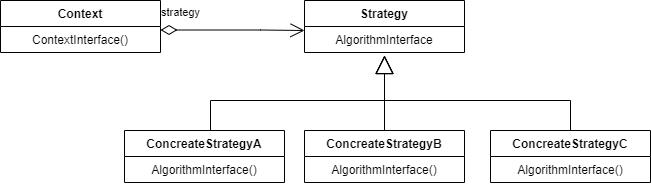
1. 效果：

相关算法系列；一个替代继承的方法；消除了一些条件语句

客户必须了解不同的Strategy；Stragety和Context之间的通信开销；增加了对象的数目

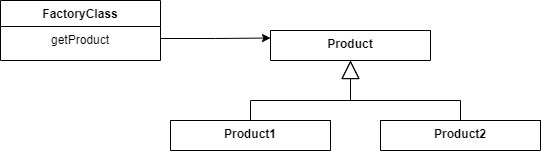
1. 实现：

定义Context和Strategy的接口：参数（解耦但是多余参数）；自身（需要定义数据接口；耦合度更高）



***~~类模式vs对象模式（类行为型模式vs对象行为型模式 and 对象创建型模式vs类创建型模式）~~***

**静态工厂**



***创建型模式 vs 行为模式***

## Template Method（模板方法）and Factory Method（工厂方法）

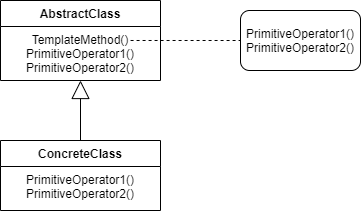
**Template Method**

1. 意图：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。
2. 适用性

一次性实现一个算法不变的部分，并将可变的行为留给子类来实现

各自类中公共的行为应当被提取出来并集中到一个父类以避免代码重复

1. 效果：代码复用的基本技术；



**Factory Method**

1. 意图：定义一个用于创建对象的接口，让子类决定实例化哪一个类。
2. 适用性

当一个类不知道它必须创建的对象的类的时候

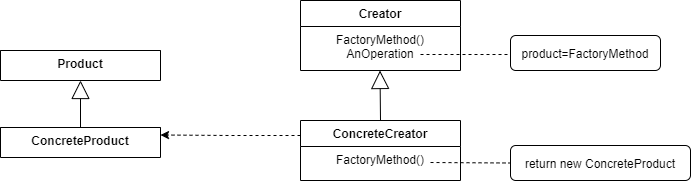
当一个类希望由它的子类来指定它所创建的对象的时候

1. 效果

不再将与特定应用有关的类绑定到代码中

代码仅处理Product接口

可扩展的ConcreteProduct



***简单工厂vs 工厂方法***

1. ***开闭原则***
2. ***抽象程度***

***策略模式 vs 模板方法***

1. ***依赖倒置***
2. ***抽象（继承） vs 灵活性（组合）***

***不是将通用的应用算法放进一个抽象的基类中，而是将它放进一个具体类中。把通用算法必须要调用的抽象方法定义在接口中，并由此进行派生。***

## Abstract Factory（抽象工厂）

**Abstract Factory**

1. 意图

提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

1. 适用性

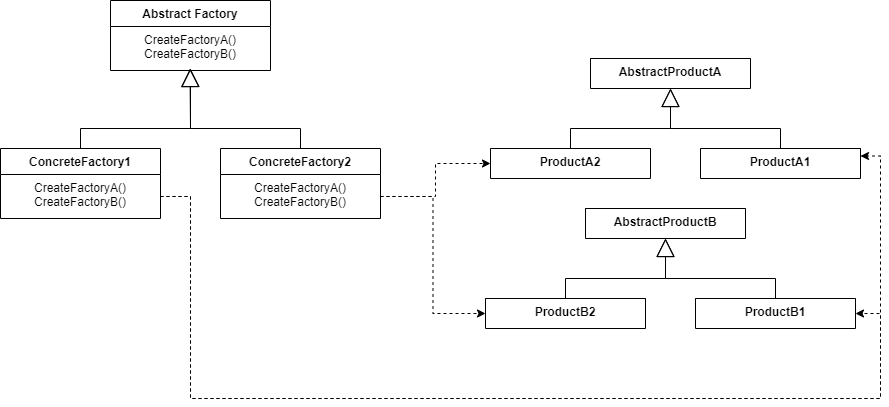
一个系统要独立于它的产品创建、组合和表示时

一个系统要由多个产品系列中的一个来配置时

当你要强调一系列相关的产品对象的设计以便进行联合使用时

当你提供了一个产品类库，而只想显示他们的接口而不是实现时

1. 结构



1. 效果

分离了具体的类

使得易于交换产品系列

有利于产品的一致性

难以支持新种类的增加

图示

描述已自动生成

|  |  |  |  |
| --- | --- | --- | --- |
|  | **目的** | | |
|  | **创建型** | **结构型** | **行为型** |
| **类** | Factory Method | Adapter | Interpreter  Template Method |
| **对象** | Abstract Factory  Builder  Prototype  Singleton | Adapter  Bridge  Composite  Decorator  Façade  Flyweight  Proxy | Chain of Responsibility  Command  Iterator  Mediator  Memento  Observer  State  Strategy  Visitor |

设计模式空间