# **什么是设计模式**

设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。设计模式（Design pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。

使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。 毫无疑问，设计模式于己于他人于系统都是多赢的，设计模式使代码编制真正工程化，设计模式是软件工程的基石，如同大厦的一块块砖石一样。项目中合理的运用设计模式可以完美的解决很多问题，每种模式在现在中都有相应的原理来与之对应，每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的核心解决方案，这也是它能被广泛应用的原因。简单说：

**模式：在某些场景下，针对某类问题的某种通用的解决方案。**

场景：项目所在的环境

问题：约束条件，项目目标等

解决方案：通用、可复用的设计，解决约束达到目标。

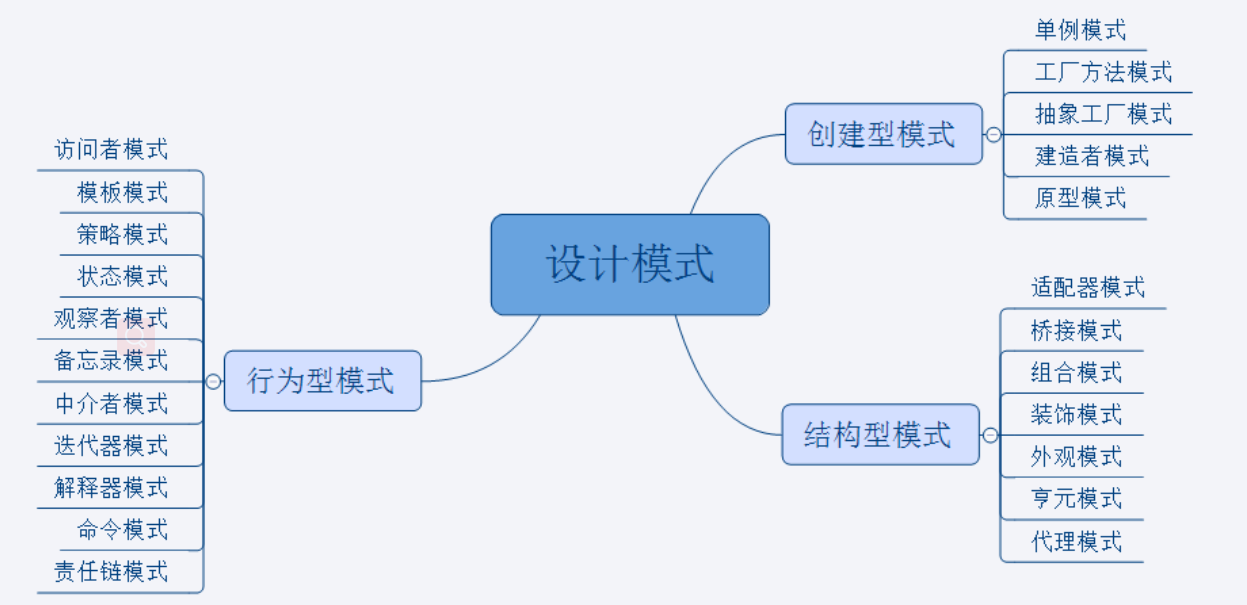
# **设计模式的三个分类**

**创建型模式：对象实例化的模式，创建型模式用于解耦对象的实例化过程。**

**结构型模式：把类或对象结合在一起形成一个更大的结构(功能组合)。**

**行为型模式：类和对象如何交互，及划分责任和算法(访问交互)。**

**如下图所示：**



# 设计模式34个

## 工厂(方法)模式

工厂方法(接口)：定义一个创建对象的接口，让子类决定实例化那个类。

简单工厂(接口实例)：一个工厂类根据传入的参量决定创建出那一种产品类的实例。

### 定义

作为抽象工厂模式的孪生兄弟，工厂方法模式定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个，也就是说工厂方法模式让实例化推迟到子类。

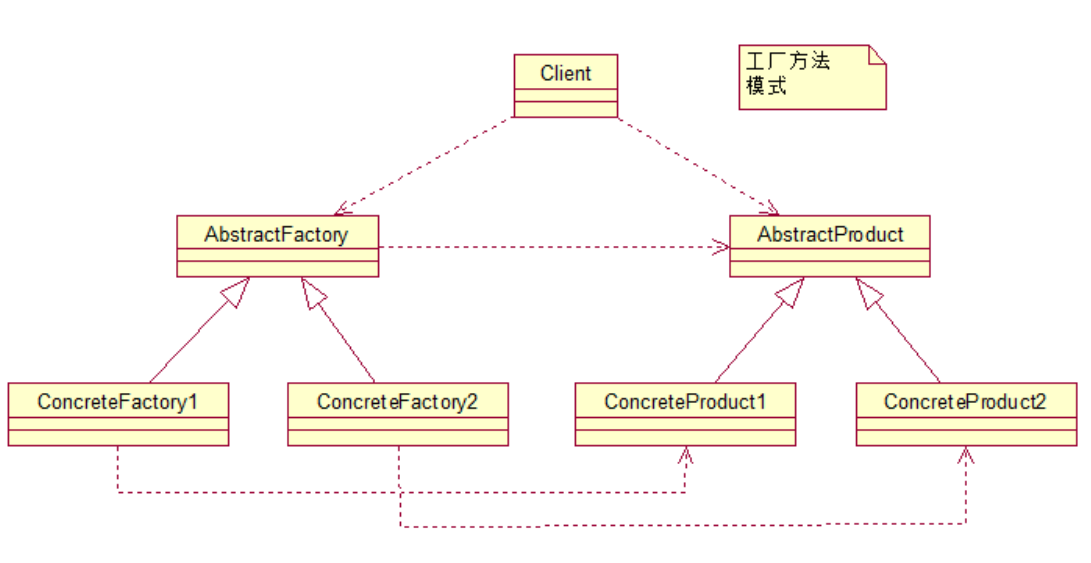
优点：

1. 工厂方法模式非常符合“开闭原则”，当需要增加一个新的产品时，我们只需要增加一个具体的产品类和与之对应的具体工厂即可，无须修改原有系统。
2. 同时在工厂方法模式中用户只需要知道生产产品的具体工厂即可，无须关心产品的创建过程，甚至连具体的产品类名称都不需要知道。

缺点：

1. 虽然他很好的符合了“开闭原则”，但是由于每新增一个新产品时就需要增加两个类，这样势必会导致系统的复杂度增加。
2. 客户端使用抽象的接口来创建一个相关的产品，需要关心实际产出的具体产品是什么, 可与和具体的产品耦合度比较高。

### 1.2 UML 类图



在工厂方法模式中用户只需要知道生产产品的具体工厂即可

客户端使用抽象的接口来创建一个相关的产品，需要关心实际产出的具体产品是什么

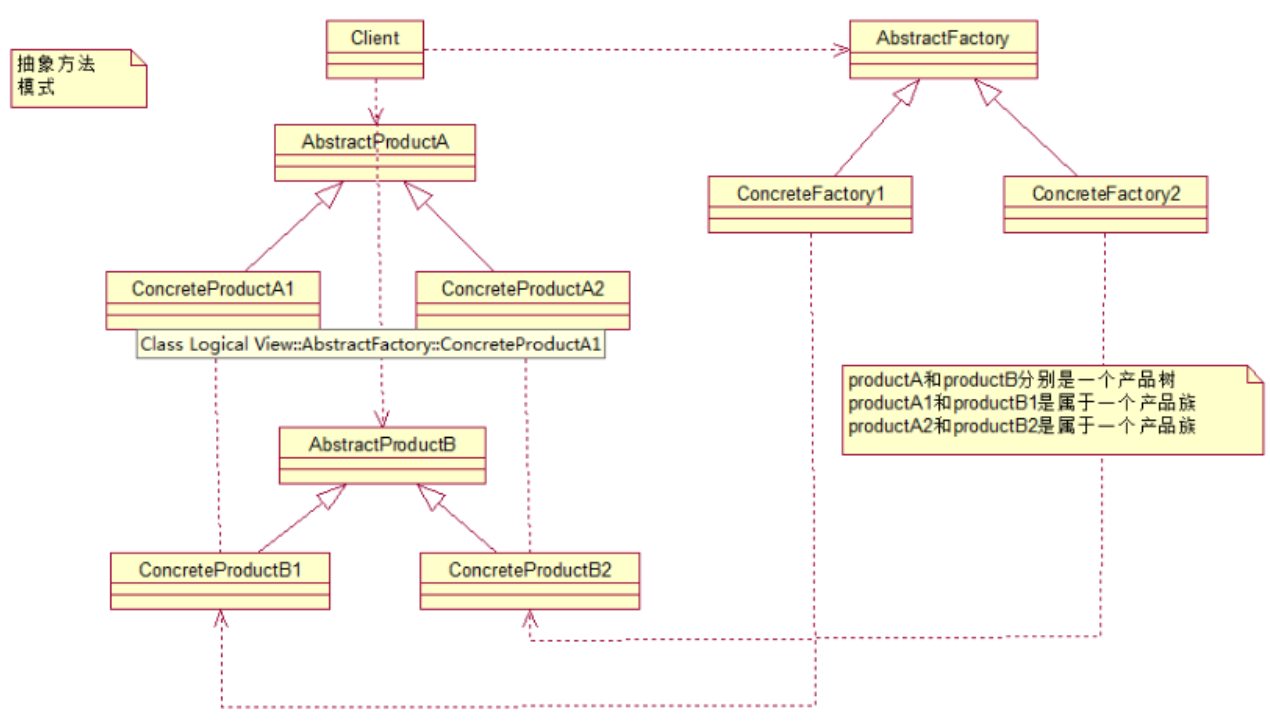
## 抽象工厂模式

抽象工厂：创建相关或依赖对象的家族，而无需明确指定具体类。

### 定义

所谓抽象工厂模式就是提供一个接口，用于创建相关或者依赖对象的家族，而不需要明确指定具体类。他允许客户端使用抽象的接口来创建一组相关的产品，而不需要关心实际产出的具体产品是什么。这样一来，客户就可以从具体的产品中被解耦。它的优点是隔离了具体类的生成，使得客户端不需要知道什么被创建了，而缺点就在于新增新的行为会比较麻烦，因为当添加一个新的产品对象时，需要更改接口及其下所有子类。

### UML类图



## 单例模式

单例模式：某个类只能有一个实例，提供一个全局的访问点。

### 3.1 定义

单例模式，它的定义就是确保某一个类只有一个实例，并且提供一个全局访问点。

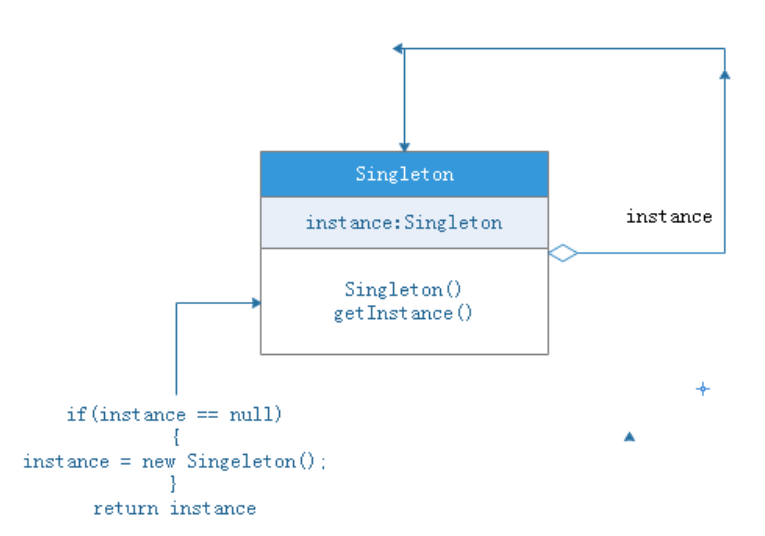
单例模式具备典型的3个特点：1、只有一个实例。 2、自我实例化。 3、提供全局访问点。

 因此当系统中只需要一个实例对象或者系统中只允许一个公共访问点，除了这个公共访问点外，不能通过其他访问点访问该实例时，可以使用单例模式。

单例模式的主要优点就是节约系统资源、提高了系统效率，同时也能够严格控制客户对它的访问。也许就是因为系统中只有一个实例，这样就导致了单例类的职责过重，违背了“单一职责原则”，同时也没有抽象类，所以扩展起来有一定的困难。

### 3.2 UML类图

其UML结构图非常简单，就只有一个类：



## 建造者模式

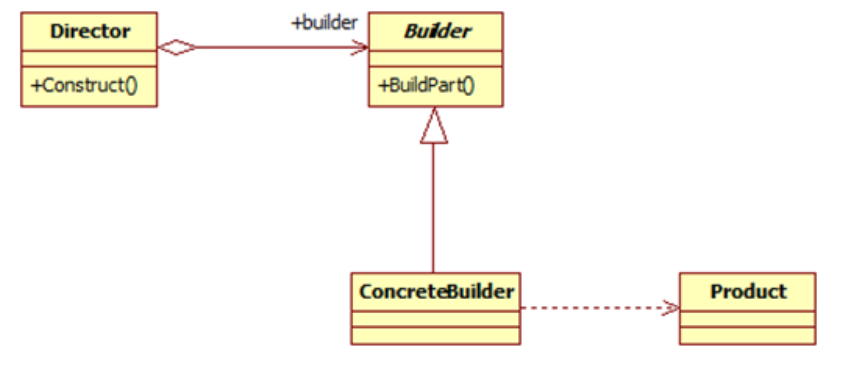
建造者模式：封装一个复杂对象的构建过程，并可以按步骤构造。

### 定义

对于建造者模式而言，它主要是将一个复杂对象的构建与表示分离，使得同样的构建过程可以创建不同的表示。适用于那些产品对象的内部结构比较复杂。

建造者模式将复杂产品的构建过程封装分解在不同的方法中，使得创建过程非常清晰，能够让我们更加精确的控制复杂产品对象的创建过程，同时它隔离了复杂产品对象的创建和使用，使得相同的创建过程能够创建不同的产品。但是如果某个产品的内部结构过于复杂，将会导致整个系统变得非常庞大，不利于控制，同时若几个产品之间存在较大的差异，则不适用建造者模式，毕竟这个世界上存在相同点大的两个产品并不是很多，所以它的使用范围有限。

### UML类图



### 4.3 建造者模式的四个角色

1. Product(产品角色) ：一个具体的产品对象
2. Builder(抽象建造者): 创建一个Product对象的各个部件指定的接口。
3. ConcreteBuilder(具体的建造者)：实现接口，构建和装配各个部件。
4. Director(指挥者)：构建一个使用Builder接口的对象。它主要是用于创建一个复杂的对象。它主要有两个作用：a. 隔离了客户与对象的生产过程。 b. 负责控制产品对象的生产过程。

## 原型模式

### 5.1模式描述

原型模式：通过复制现有的实例来创建新的实例。

## 适配器模式

适配器模式：将一个类的方法接口转换成客户希望的另外一个接口。

## 桥接模式

桥接模式：将抽象部分和它的实现部分分离，使它们都可以独立的变化。

## 过滤器模式

## 组合模式

组合模式：将对象组合成树形结构以表示“”部分-整体“”的层次结构。

## 装饰器模式

装饰模式：动态的给对象添加新的功能。

## 外观模式

外观模式：对外提供一个统一的方法，来访问子系统中的一群接口。

## 亨元模式

亨元（蝇量）模式：通过共享技术来有效的支持大量细粒度的对象。

## 代理模式

代理模式：为其他对象提供一个代理以便控制这个对象的访问。

## 责任链模式

责任链模式：将请求的发送者和接收者解耦，使的多个对象都有处理这个请求的机会。

## 命令模式

命令模式：将命令请求封装为一个对象，使得可以用不同的请求来进行参数化。

## 解释器模式

解释器模式：给定一个语言，定义它的文法的一种表示，并定义一个解释器。

## 迭代器模式

迭代器模式：一种遍历访问聚合对象中各个元素的方法，不暴露该对象的内部结构。

## 中介者模式

中介者模式：用一个中介对象来封装一系列的对象交互。

## 备忘录模式

备忘录模式：在不破坏封装的前提下，保持对象的内部状态。

## 观察者模式

观察者模式：对象间的一对多的依赖关系。

## 状态模式

状态模式：允许一个对象在其对象内部状态改变时改变它的行为。

## 空对象模式

## 策略模式

策略模式：定义一系列算法，把他们封装起来，并且使它们可以相互替换。

## 模板模式

模板模式：定义一个算法结构，而将一些步骤延迟到子类实现。

## 访问者模式

访问者模式：在不改变数据结构的前提下，增加作用于一组对象元素的新功能。

## MVC模式

## 业务代表模式

## 组合实体模式

## 数据访问对象模式

## 前端控制器模式

## 拦截过滤器模式

## 服务定位器模式

## 传输对象模式

## 设计模式资源