**设计模式概括**

# 设计模式概括

# 设计模式分类

# 设计模式详细讲解

## 简单工厂

### 定义

### 结构和说明

## 外观模式

### 定义

为子系统中的一组接口提供一个一致的界面，Facade模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

### 结构和说明



### 优缺点

优点：

* 松散耦合

外观模式松散了客户端与子系统的耦合关系，让子系统内部的模块能更容易扩展和维护

* 简单易用

外观模式让子系统更加易用，客户端不再需要了解子系统内部的实现，也不需要跟众多子系统内部的模块进行交互，只需要跟外观交互就可以了，相当于外观类为外部客户使用子系统提供了一站式服务。

* 更好地划分访问的层次

通过合理使用Façade，可以帮助我们更好地划分访问的层次。有些方法是对系统外的，有些方法是系统内部使用的。把需要暴露给外部的功能集中到外观中，这样既方便客户端使用，也很好地隐藏了内部的细节。

缺点：

* 过多的或者是不太合理的Façade也容易让人迷惑。到底是调用Façade好呢，还是直接调用模块好。

### 思考外观模式

* 本质

外观模式的本质是：封装交互，简化调用

* 对设计原则的体现

最少知识原则

* 何时选用外观模式
  + 如果你希望为一个复杂的子系统提供一个简单接口的时候，可以考虑使用外观模式。使用外观对象来实现大部分客户需要的功能，从而简化客户的使用。
  + 如果想要让客户程序和抽象类的实现部分松散耦合，可以考虑使用外观模式，使用外观对象来将这个子系统与它的客户分离开来，从而提高子系统的独立性和可移植性。
  + 如果构建多层结构的系统，可以考虑使用外观模式，使用外观对象作为每层的入口，这样可以简化层间调用，也可以松散层次之间的依赖关系。

### 相关模式

* 中介者模式
* 单例模式
* 抽象工厂

## 适配器模式

### 定义

将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

### 结构和说明



Client:客户端，调用自己需要的领域接口Target

Target:定义客户端需要的跟特定领域相关的接口

Adaptee:已经存在的接口，通常能满足客户端的功能要求，但是接口与客户端要求的特定领域接口不一致，需要被适配。

Adapter:适配器，把Adaptee适配成为Client需要的Target.

### 优缺点

优点：

* 更好的复用性

如果功能是已经有了的，只是接口不兼容，那么通过适配器模式就可以让这些功能得到更好的复用。

* 更好的可扩展性

在实现适配器功能的时候，可以调用自己开发的功能，从而自然地扩展系统的功能。

缺点：

* 过多的使用适配器，会让系统 非常零乱，不容易整体进行把握

比如，明明看到调用的是A接口，其实内部被适配成了B接口来实现，一个系统如果 太多出现这种情况，无异于一块灾难。因此如果不是很有必要，可以不使用适配器，而是直接对系统进行重构。

### 思考适配器模式

### 本质

适配器模式的本质是：转换匹配，复用功能。

### 使用时机

* + - 如果你想要使用一个已经存在的类，但是它的接口不符合你的需求，这种情况可以使用适配器模式，来把已有的实现转换成你需要的接口。
    - 如果你想创建一个可以复用的类，这个类可能和一些不兼容的类一起工作，这种情况可以使用适配器模式，到时候需要什么就适配什么。
    - 如果你想使用一些已经存在的子类，但是不可能对每一个子类都进行适配，这种情况可以选用对象适配器，直接适配这些子类的父类就可以了。

### 相关模式

* + 适配器模式与桥接模式
  + 适配器模式与装饰模式
  + 适配器模式与代理模式
  + 适配器模式与抽象工厂模式

## 单例模式

### 定义

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

### 结构和说明



### 实现方式

* 饿汉式
* 懒汉式
* 内部类 Holder
* 枚举

### 思考单例模式

本质：

单例模式的本质：控制实例数目

场景：

当需要控制 一个类的实例只能有一个，而且客户只能从一个全局访问点访问它时，可以选用单例模式，这些功能恰好是单例模式要解决的问题

### 相关模式

抽象工厂方法中的具体工厂类就通常是一个单例。

## 工厂方法模式

### 定义

定义一个用于创建对象的接口，让子类决定实例化哪个类，Factory Method使一个类的实例化延迟到其子类

### 结构和说明



**Product**: 定义工厂方法所创建的对象的接口，也就是实际需要使用的对象的接口

**ConcreteProduct**: 具体的Product接口的实现对象

**Creator**: 创建器，声明工厂方法，工厂方法通常会返回一个Product类型的实例对象，而且多是抽象方法。也可以在Creator里面提供工厂方法的默认实现，让工厂方法返回一个缺省的Product类型的实例对象。

**ConcreteCreator**: 具体的创建对象，覆盖实现Creator定义的工厂方法，返回具体的Product实例。

### 工厂方法模式的调用顺序示意图



### 优缺点

* 优点
  + 可以在不知具体实现的情况下编程
  + 更容易扩展对象的新版本
  + 连接平行的类层次
* 缺点
  + 具体产品对象和工厂方法的耦合性

### 思考工厂方法模式

#### 本质

工厂方法模式的本质：延迟到子类来选择实现

#### 设计原则

依赖倒置原则

#### 何时选用工厂方法模式

* 如果一个类需要创建某个接口的对象，但是又不知道具体的实现，这种情况可以选用工厂方法模式，把创建对象的工作延迟到子类中去实现
* 如果一个类本身就希望由它的子类来创建所需的对象的时候，应该使用工厂方法模式

### 相关模式

* 工厂方法模式和抽象工厂模式
* 工厂方法模式和模板方法模式

## 单例模式

### 定义

### 结构和说明

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