# 一、引言

## 1. 组织编目

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

根据两条准则对模式进行分类。

目的准则：模式是用来完成什么工作的。

创建型模式与对象的创建有关，创建型类将对象的部分创建工作延迟到子类中，创建型对象将它延迟到另一个类中；结构型模式处理类或对象的组合，结构型类使用继承机制来组合类，结构型对象描述对象的组装方式；行为型模式对类或对象怎样交互和怎样分配职责进行描述，行为型类模式使用继承描述算法和控制流，而行为型对象描述一组对象怎样协作完成单个对象无法完成的工作。

范围准则：指定模式主要是用于类还是对象。

类模式处理类和子类之间的关系，这些关系通过集成建立，是静态的，在编译时刻就确定下来了；

对象模式处理对象间的关系，这些关系在运行时刻是可以变化的，更具动态性。

## 2. 怎么使用设计模式解决问题

你可以写出一个问题的描述，找出名词和动词，进而创建相应的类和操作；或者关注于系统的协作和职责关系；对现实世界建模，再将分析时发现的对象转化到设计中。

## 3. 定义

signature:对象声明的每一个操作指定操作名、作为参数的对象和返回值。（即函数）

接口：对象所定义的所有操作signature的集合。

类型(type)：用来标识特定接口的一个名字。

动态绑定(dynamic binding)：发送给对象的请求和它的相应操作在运行时刻的连接。动态绑定允许你在运行时刻彼此替换有相同接口的对象（多态）。

设计模式通过确定接口的主要成分及经过接口发送的数据类型，来帮助人们定义接口。

一个对象的类定义了对象是怎么实现的，同时也定义了对象内部状态和操作的实现。

对象的类型只与它的接口有关。

类继承是根据一个对象的实现定义了另一个对象的实现，它是代码和表示的共享机制

接口继承描述了一个对象什么时候能够被用来代替另一个对象

**对接口编程，而不是对实现编程。（接口的定义）**

不将变量声明为某个特定的具体类的实例对象，而是让它遵从抽象类所定义的接口。

**优先使用对象组合，而不是类继承。**

类继承可以较方便地改变被复用的实现，但是无法在运行时刻改变从父类继承的实现，而且父类实现中的任何改变都必然导致子类发生变化。可用的解决办法就是指继承接口。

对象组合式通过获得对其他对象的引用而在运行时刻动态定义的。只要类型一致，运行时刻还可以用一个对象来代替另一个对象；因为对象实现是基于接口的，所以是线上存在较少的依赖关系。

导致重新设计的一般原因：

通过显示地指定一个类来创建对象。（可以采用Spring Bean的模式）

对特殊操作的依赖

对硬件和软件平台的依赖。（将对外部依赖封装起来）。

对对象表示或实现的依赖

算法依赖

紧耦合

通过子类来扩充功能

不方便对类进行修改

# 二、创建型模式

创建型模式抽象了实例化过程，他们帮助一个系统独立于如何创建、组合和表示它的那些对象。一个类创建型模式使用继承改变被实例化的类，而一个对象创建型模式将实例化委托给另一个对象。

**根据所用的配置，方便地变换创建或装配对象的结构类型（自己体会）**

## 1. Abstract Factory (抽象工厂) – 对象创建型模式

意图：

提供一个创建一系列相关或相互依赖的对象的接口，而无需指定它们具体的类。

结构：



AbstractFactory: 声明一个创建抽象产品对象的操作接口

ConcreteFactory: 实现创建具体产品对象的操作

AbstractProduct: 为一类产品对象声明一个接口

ConcreteProduct: 定义一个将被相应的具体工厂创建的产品对象

Client: 仅使用AbstractFactory和AbstractProduct接口

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产品对象的设计以便进行联合使用时
* 当你提供一个产品类库，而指向显示它们的接口而不是实现时

优缺点：

* 它分离了具体的类：它将客户与类的实现分离，客户通过他们的抽象接口操纵实例
* 它使得易于交换产品系列
* 它有利于产品的一致性
* 难以支持新种类的产品： 若要支持新种类的产品，要将Factory从头改到尾

实现：

* 将工厂作为单件：一个应用中一般每个产品系列只需一个ConcreteFactory实例，因此工厂通常最好实现为一个Singleton（构造器protected）；
* 创建产品：AbstractFactory为创建产品的接口，ConcreteFactory创建
* 定义可扩展的工厂： 新增产品要求改变AbstractFactory的接口以及所有相关的类。一个更灵活但不太安全的设计是给创建对象的操作增加一个参数，该参数指定了被创建对象的种类。

相关模式：

AbstractFactory通常使用工厂方法来实现，也可以用Prototype实现，一个具体的工厂通常是一个单件。

## **2. BUILDER（生成器模式）**

意图：

将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

适用性：

当创建复杂对象的算法应该独立于该对象的组成部分以及它们的装配方式时

当构造过程必须允许被构造的对象有不同的表示时

结构：



参与者：

* Builder： 为创建一个Product对象的各个部件指定抽象接口
* ConcreteBuilder： 实现ConcreteBuilder的接口以构造和装配该产品的各个部件； 定义并明确它所创建的表示；提供一个检索产品的接口(getResult)
* Director: 构造一个使用Builder接口的对象
* Product：表示被构造的复杂对象，ConcreteBuilder创建该产品的内部表示并定义它的装配过程；包含定义组成部件的类，包括将这些部件装配成最终产品的接口

过程：

* 客户创建Director对象，并用它想要的Builder对象进行配置
* 一旦产品部件被生成，导向器就会通知生成器
* 生成器处理导向器的请求，并将部件添加到该产品中
* 客户从生成器中检索产品

效果：

* 它使你可以改变一个产品的内部表示
* 它将构造代码和表示代码分开：不同的Director可以复用ConcreteBuilder
* 它使你可对构造过程进行更精细的控制

相关模式：

AbstractFactory与Builder相似，因为它也可以创建复杂对象。主要的区别是Builder模式着重于一步步构造一个复杂的对象；而AbstractFactory着重于多个系列的产品对象。Builder在最后的一步返回产品，而对于AbstractFactory来说，产品是立即返回的

**Composite通常是Builder生成的。**

## 3. Factory Method (工厂方法)

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。Factory Method使一个类的实例化延迟到其子类

适用性：

* 当一个类不知道它所必须创建对象的类的时候
* 当一个类希望由它的子类来指定它所创建的对象的时候
* 当类创建对象的职责委托给多个帮助子类中的某一个，并且你希望将哪一个帮助子类是代理者这一信息局部化的时候

结构：



参与者：

* Product：定义工厂方法所创建的对象的接口
* ConcreteProduct：实现Product接口
* Creator：声明工厂方法，该方法返回一个Product类型的对象。Creator也可以定义一个缺省的实现，返回一个缺省的ConcreteProduct
* ConcreteCreator：重定义工厂方法以返回一个ConcreteProduct

协作：

Creator依赖于它的子类来定义工厂方法，所以它返回一个适当的ConcreteProduct。

效果：

* 为子类提供挂钩（hook）
* 连接平行的类层次

## 4. PROTOTYPE（对象创建型模式）

意图：

用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象

适用性：

* 一个系统应该独立于它的产品创建、构成和表示
* 实例化的类使运行时刻指定的：动态装载
* 避免创建一个与产品类层次平行的工厂类层次时
* 当一个类的实例只能有几个不同状态组合中的一种时。建立相应数目的原型并克隆他们可能比每次用合适的状态手工实例化该类更方便

结构：



参与者：

* Prototype:声明一个克隆自身的接口
* ConcretePrototype: 实现一个克隆自身的操作
* Client: 让一个原型克隆自身从而创建一个新的对象