设计模式原则（重用性、可读性、可靠性、可维护性）

1 单一原则：一个方法、一个类尽量只做一个事情（尽量保证在修改一个功能的时候不会影响别的功能）

2 里氏替换原则：所有引用基类的地方必须能透明地使用其子类的对象。子类可以扩展父类的功能，但是不能改变父类原有的功能（不能破坏集成关系）。

3 依赖倒置原则：应该是依赖接口编程，而尽量减少依赖具体实现编程。

4 接口隔离：类似于单一原则，但是用来约束接口。一个N多个功能的接口可以拆成多个单一功能的接口，用来减少依赖。

5 迪米特原则：最少知道原则，一个对象应当对其他对象尽可能少的了解，使得系统功能模块相对独立。低耦合，高内聚。

6 开闭原则：对扩展开放，对修改关闭。

高内聚，低耦合：用户模块、账户模块、支付模块（比如用户模块中如果直接修改账户模块数据，同时账户模块也可以修改用户模块数据，这就行相当于低内聚，高耦合。但是如果是用户模块维护用户模块数据，账户模块维护账户模块数据，二者通过接口进行交互，则可以实现，高内聚，低耦合，一个大的系统进行微服务拆分。）

创建型模式

1 单例模式

```
* 首先我们要先了解下单例的四大原则：
*
* 1.构造私有
*
* 2.以静态方法或者枚举返回实例
*
* 3.确保实例只有一个，尤其是多线程环境
*
* 4.确保反序列换时不会重新构建对象
*
*
* 饿汉模式
*  关注点：
*       1 构造方法：必须是private
*       2 有一个private static final的属性，重点final
*       3 提供一个静态方法
* 缺点：即使用不到这个单例对象，也会生成占用内存空间
```

饿汉模式-浪费空间

```
* 懒汉模式
*       关注点：
*            1 构造方法必须是private
*            2 懒加载
*       缺点 ：
*            会有并发问题（对象的半初始化）
```

懒汉模式-线程安全问题

DCL单例-必须有volatile关键字+双重锁

```
* DCL模式
*       主要点:
*           1 构造方法是private
*           2 属性必须是volatile的
*           3 双检查
*       缺点 :
*           1 写法复杂
*           2 可通过反射获取
```

内部类实现单例-可以被反射生成和序列化问题

```
* 内部类单例
*       关注点:
*           1 构造方式是private
*           2 有一个静态内部类并且是private
*           3 内部类中有一个静态属性是单例对象，采用饿汉模式实现
*       优点:
*           1 外部类加载时并不需要立即加载内部类，内部类不被加载则不去初始化INSTANCE，故而不占
内存
*           2 JVM加载类会保证单例
*       缺点:
*           1 可以被反射获得
*           2 序列化问题(要么就不实现Serializable方法，要么就会有序列化问题)
*           3 传参很麻烦
```

枚举单例-可以解决反射和反序列化问题

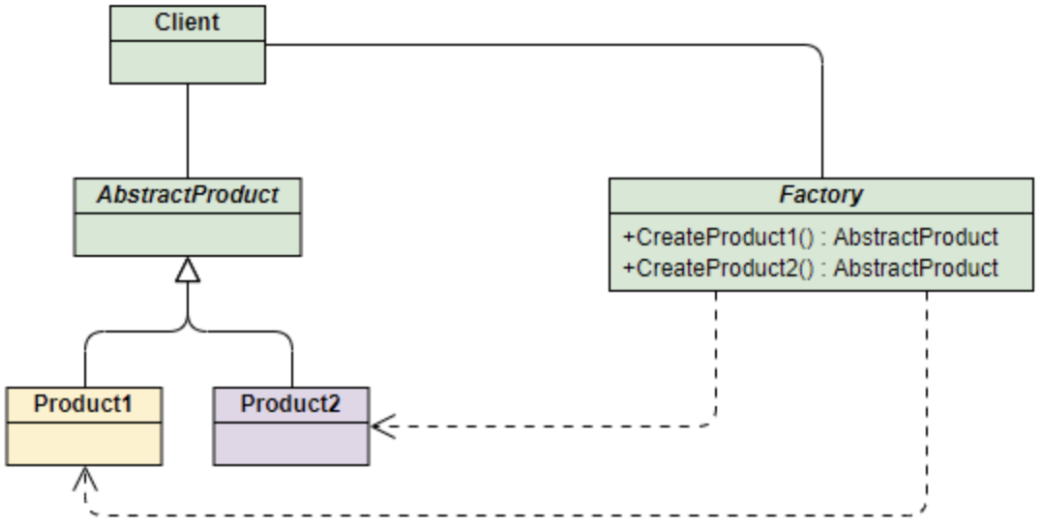
```
* 枚举单例
* 可以解决反射、序列化、唯一问题
```

2 工厂模式（BeanFactory\FactoryBean）

（对于调用者来说，隐藏了复杂的逻辑处理过程，调用者只关心执行结果，工厂要对结果负责，保证生产出符合规范的产品）

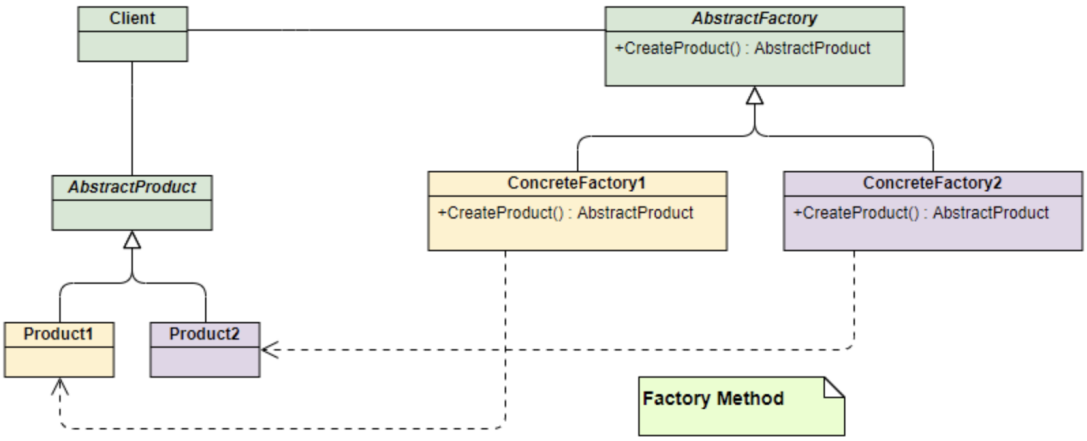
实际例子：spring容器中bean对象的获取方法就是工厂方法

简单工厂



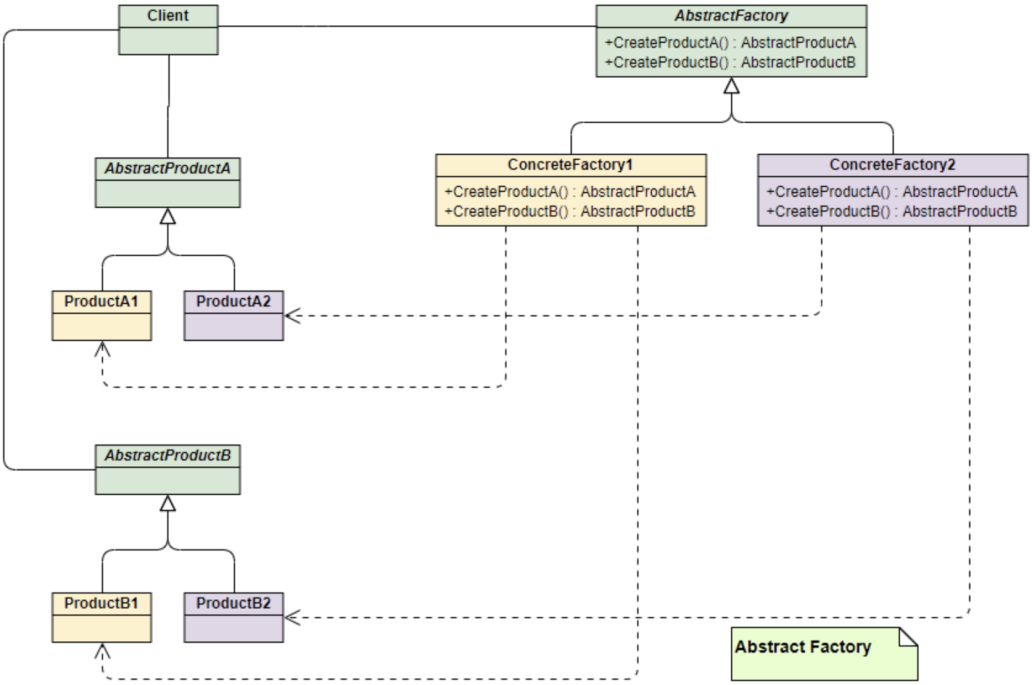

```
* 1 简单工厂
*
*       主要点：
*           1 有一个抽象产品接口
*           2 有多个抽象产品实现类
*           3 有一个抽象工厂-该工厂可以通过if-else或者switch语句生产对应的产品，或者就是多个方法，每个方法返回一种产品
*
*       参考spring中bean对象的获取
*       AbstractBeanFactory.getBean(String name)方法
```

工厂方法




```
*  工厂方法
*       要点：
*           1 有一个抽象产品接口
*           2 有多个抽象产品的实现类
*           3 有一个抽象工厂接口，抽象工厂接口中有一个生产抽象产品的方法
*           4 抽象工厂有多个实现类
```

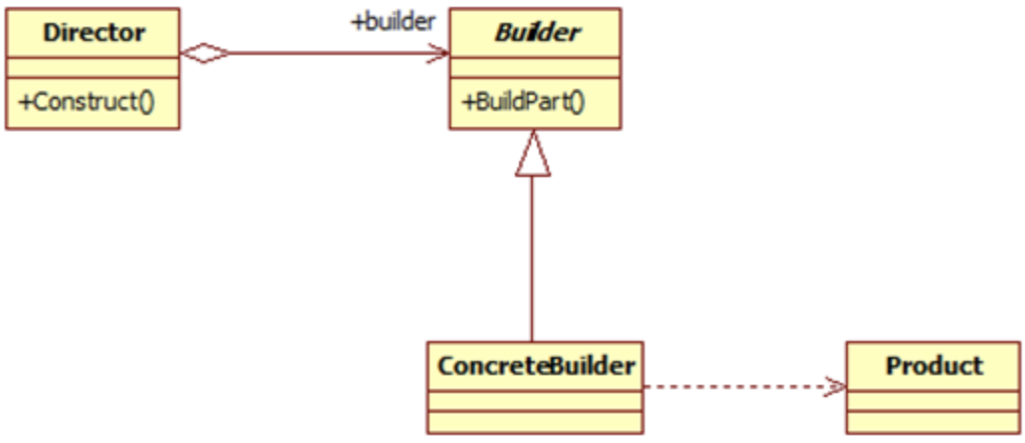
抽象工厂




```
*  抽象工厂方法：
*        主要点：
*              与工厂方法最大的不同是，工厂方法每个工厂只能生产一个产品，而工厂方法可以生成一个产品族
```

9 建造者模式（复杂对象的构建，比如一个对象有N多个属性，会有多种组合方式产生多种对象。可以采用建造者模式，根据需求定制产品。链式编程，实际例子，StringBuilder）

```
* 构建者模式
*        主要点：
*              作用于复杂对象的构建上，使用者可以根据自己的需求生产不同的对象(工厂模式是把复杂的构建过程封装，使用者不关心构建过程，
*              每次获取的对象都是具有相似属性的对象，而构建者模式，可以封装部分固定内容，其余的可由使用者指定)
*
*        参考java中的StringBuilder
```

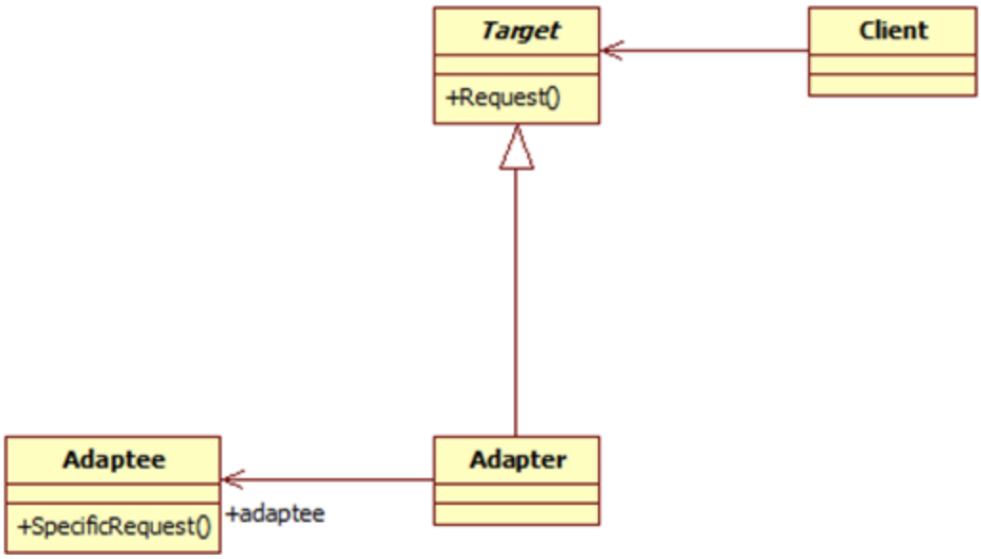


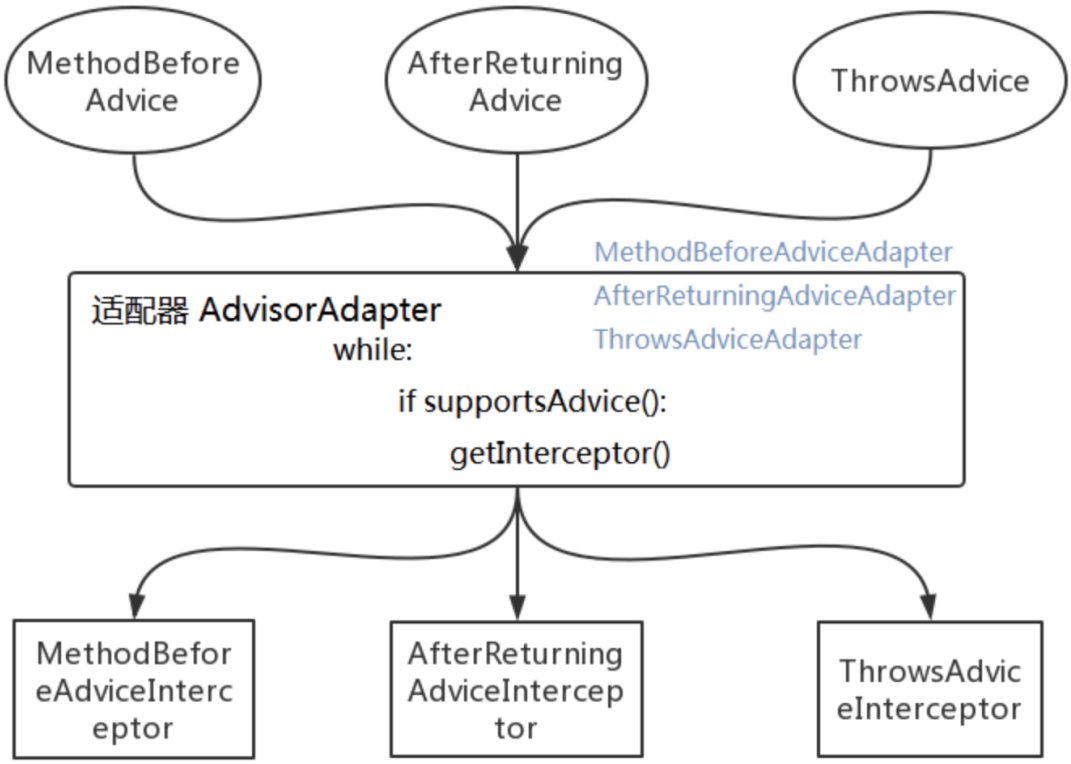
15 原型模式-复制对象（参考clone方法，会引入深copy和浅copy的问题）

结构型模式

5 适配器模式（spring中的AOP使用了适配器模式）

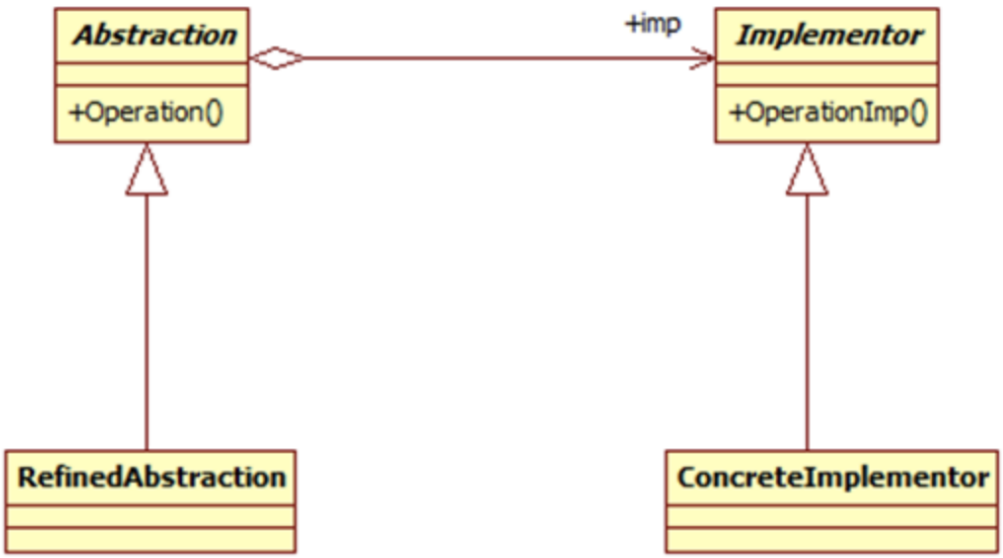
```
* 适配器模式：
*       1 把b方法的功能通过a方法包装之后给目标对象调用
*
*       spring中的适配器模式:aop中的增强器
```





8 桥建模式（两个分支，通过桥梁连接）

```
*  主要点：
*       1 两个纬度上进行扩展
*       2 第二个纬度上有第一个纬度的对象当成属性
```



14 组合模式（用来处理树状结构的模式）

10 装饰模式

```
* 装饰器模式：
*       主要点：
*            1 装饰器类也需要实现原始类，然后在装饰器中有一个属性是原始类
*            与适配器的不同是：适配器是替换原方法内容，而装饰器是增强功能
*
*            典型应用：java-io
```

7 代理模式

静态代理

动态代理

```
* cglib动态代（底层还是asm字节码技术）
*           1 可以为具体类做代理，原理是继承，代理类必须实现MethodInterceptor类，重写
intercept方法
*                intercept方法详解:1 代理类本身 2 拦截方法 3 方法参数 4 增强的方法
*           2 类不能为final的，否则直接报错，方法可以为final，但是final的方法不会代理，非
final的方法可以被代理
*           2 非private、非final、非static可以生成代理
*
*           cglib的原理是这样，它生成一个继承B的类型C（代理类），这个代理类持有一个
MethodInterceptor，
*           我们setCallback时传入的。 C重写所有B中的方法（方法名一致），然后在C中，
*           构建名叫“CGLIB”+“$父类方法名$”的方法（下面叫cglib方法，所有非private的方法都会
被构建），
*           方法体里只有一句话super.方法名()，可以简单的认为保持了对父类方法的一个引用，方便调
用。
*
* 这样的话，C中就有了重写方法、cglib方法、父类方法（不可见），还有一个统一的拦截方法（增强方法
intercept）。
* 其中重写方法和cglib方法肯定是有映射关系的。
*
* C的重写方法是外界调用的入口（LSP原则），它调用MethodInterceptor的intercept方法，
```

```
* 调用时会传递四个参数，第一个参数传递的是this，代表代理类本身，第二个参数标示拦截的方法，
* 第三个参数是入参，第四个参数是cglib方法，intercept方法完成增强后，
* 我们调用cglib方法间接调用父类方法完成整个方法链的调用。
*
*
* 代理对象调用this.setPerson方法->调用拦截器->methodProxy.invokeSuper-
>CGLIB$setPerson$0->被代理对象setPerson方法
```

```
* 代理模式
*       静态代理(类似装饰器)
*
*       动态代理：
*           jdk动态代理（底层是asm字节码技术）
*           1 必须是接（可以通过指定参数查看生成的代理类）
*           2 必须实现InvocationHandler类,重写invoke方法，不是必须继承抽象类
*               inovke方法参数详解,1 生成的代理类本身 2 当前执行的方法 3 当前执行的方法的参
数
*           3 通过Proxy.newProxyInstance方法创建代理对象
*               newProxyInstance方法的参数详解 1 加载目标类的classloader 2 加载目标类的
接口 3 使用那种代理策略
```

13 享元模式（对比对象池模式，享元模式可以只不同对象存在一个地方，而对象池技术是，池中的对象都是一样的，取那个都行）

11 门面模式-调停者模式（对外是门面模式，对内是调停者模式）

行为型模式

3 策略模式（一个接口，多个实现类，可以自由切换多个实现类，可以实现算法的定义和算法的使用分来）

4 责任链模式（spring中的AOP使用了责任链模式）

```
* 责任链模式
*       主要点：
*           实现方式一：每个实现类内部都持有下一个实现类的引用
*           实现方式二：aop实现方式(每个方法都传入一个队列，或者持有这个队列的对象)
```

6 观察者模式（监听模式、钩子函数、回调函数,spring中的各种listener）

```
* 观察者模式
*       1 当对象的某种状态发生了变化之后，一个或者多个观察者观察到相应的改变之后，作出相应的动作
*
*       spring中的多个listener
```

12 模板模式（类比策略模式：模板模式的调用者是调用的基类中的方法，由基类方法来控制整个流程的运行。典型的是JUC中AQS的acquire方法的实现）