# Spring中的设计者模式

**Spring 框架中用到了哪些设计模式：**

**工厂设计模式** : Spring使用工厂模式通过 BeanFactory、ApplicationContext 创建 bean 对象。

**代理设计模式** : Spring AOP 功能的实现。

**单例设计模式** : Spring 中的 Bean 默认都是单例的。

**模板方法模式** : Spring 中 jdbcTemplate、hibernateTemplate 等以 Template 结尾的对数据库操作的类，它们就使用到了模板模式。

**包装器设计模式** : 我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。

**观察者模式**: Spring 事件驱动模型就是观察者模式很经典的一个应用。

**适配器模式** :Spring AOP 的增强或通知(Advice)使用到了适配器模式、spring MVC 中也是用到了适配器模式适配Controller。

## 工厂模式

Spring使用工厂模式可以通过 BeanFactory 或 ApplicationContext 创建 bean 对象。

两者对比：

BeanFactory ：延迟注入(使用到某个 bean 的时候才会注入),相比于BeanFactory来说会占用更少的内存，程序启动速度更快。

ApplicationContext ：容器启动的时候，不管你用没用到，一次性创建所有 bean 。BeanFactory 仅提供了最基本的依赖注入支持，ApplicationContext 扩展了 BeanFactory ,除了有BeanFactory的功能还有额外更多功能，所以一般开发人员使用ApplicationContext会更多。

## 单例设计模式

Spring 中 bean 的默认作用域就是 singleton(单例)的。

## 代理设计模式(AOP)

分JDK Proxy代理与CGLIB代理

AspectJ 应该算的上是 Java 生态系统中最完整的 AOP 框架了。

Spring AOP 和 AspectJ AOP 有什么区别?

Spring AOP 属于运行时增强，而 AspectJ 是编译时增强。 Spring AOP 基于代理(Proxying)，而 AspectJ 基于字节码操作(Bytecode Manipulation)

模板方法

Spring 中 jdbcTemplate、hibernateTemplate 等以 Template 结尾的对数据库操作的类，它们就使用到了模板模式。一般情况下，我们都是使用继承的方式来实现模板模式，但是 Spring 并没有使用这种方式，而是使用Callback 模式与模板方法模式配合，既达到了代码复用的效果，同时增加了灵活性。

public abstract class Template {

//这是我们的模板方法

public final void TemplateMethod(){

PrimitiveOperation1();

PrimitiveOperation2();

}

protected void PrimitiveOperation1(){

//当前类实现

}

//被子类实现的方法

protected abstract void PrimitiveOperation2();

}

public class TemplateImpl extends Template {

@Override

public void PrimitiveOperation2() {

//当前类实现

}

}

## 观察者模式

Spring 的事件流程总结:

定义一个事件: 实现一个继承自 ApplicationEvent，并且写相应的构造函数；

定义一个事件监听者：实现 ApplicationListener 接口，重写 onApplicationEvent() 方法；

使用事件发布者发布消息: 可以通过 ApplicationEventPublisher 的 publishEvent() 方法发布消息。

## 适配器模式

适配器模式(Adapter Pattern) 将一个接口转换成客户希望的另一个接口，适配器模式使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。

spring AOP中的适配器模式:我们知道 Spring AOP 的实现是基于代理模式，但是 Spring AOP 的增强或通知(Advice)使用到了适配器模式，与之相关的接口是AdvisorAdapter 。Advice 常用的类型有：BeforeAdvice（目标方法调用前,前置通知）、AfterAdvice（目标方法调用后,后置通知）、AfterReturningAdvice(目标方法执行结束后，return之前)等等。每个类型Advice（通知）都有对应的拦截器:MethodBeforeAdviceInterceptor、AfterReturningAdviceAdapter、AfterReturningAdviceInterceptor。Spring预定义的通知要通过对应的适配器，适配成 MethodInterceptor接口(方法拦截器)类型的对象（如：MethodBeforeAdviceInterceptor 负责适配 MethodBeforeAdvice）。

spring MVC中的适配器模式:在Spring MVC中，DispatcherServlet 根据请求信息调用 HandlerMapping，解析请求对应的 Handler。解析到对应的 Handler（也就是我们平常说的 Controller 控制器）后，开始由HandlerAdapter 适配器处理。HandlerAdapter 作为期望接口，具体的适配器实现类用于对目标类进行适配，Controller 作为需要适配的类。

## 装饰者模式

Spring 中配置 DataSource 的时候，DataSource 可能是不同的数据库和数据源。我们能否根据客户的需求在少修改原有类的代码下动态切换不同的数据源？这个时候就要用到装饰者模式(这一点我自己还没太理解具体原理)。Spring 中用到的包装器模式在类名上含有 Wrapper或者 Decorator。这些类基本上都是动态地给一个对象添加一些额外的职责