第四章：Spring AOP 设计模式

小马哥（mercyblitz）

Spring AOP 设计模式

1. 抽象工厂模式（Abstract factory）实现
2. 构建器模式（Builder）实现
3. 工厂方法模式（Factory method）实现
4. 原型模式（Prototype）实现
5. 单例模式（Singleton）实现
6. 适配器模式（Adapter）实现
7. 组合模式（Composite）实现
8. 装饰器模式（Decorator）实现
9. 享元模式（Flyweight）实现
10. 代理模式（Proxy）实现

Spring AOP 设计模式

11. 模板方法模式（Template Method）实现
12. 责任链模式（Chain of Responsibility）实现
13. 观察者模式（Observer）实现
14. 策略模式（Strategy）实现
15. 命令模式（Command）实现
16. 状态模式（State）实现
17. 面试题精选

抽象工厂模式（Abstract factory）实现

- 基本概念

抽象工厂模式提供了一种方式，可以将一组具有同一主题的单独的工厂封装起来。在正常使用中，客户端程序需要创建抽象工厂的具体实现，然后使用抽象工厂作为接口来创建这一主题的具体对象。客户端程序不需要知道（或关心）它从这些内部的工厂方法中获得对象的具体类型，因为客户端程序仅使用这些对象的通用接口。抽象工厂模式将一组对象的实现细节与他们的一般使用分离开来。

- Spring AOP 举例实现
  - 接口 - org.springframework.aop.framework.AopProxyFactory
  - 实现 - org.springframework.aop.framework.DefaultAopProxyFactory

构建器模式（Builder）实现

- 基本概念

建造模式，是一种对象构建模式。它可以將複雜對象的建造過程抽象出來（抽象類別），使这个抽象过程的不同实现方法可以构造出不同表现（属性）的对象。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.aspectj.annotation.BeanFactoryAspectJAdvisorsBuilder

工厂方法模式（Factory method）实现

- 基本概念

就像其他创建型模式一样，它也是处理在不指定对象具体类型的情况下创建对象的问题。工厂方法模式的实质是“定义一个创建对象的接口，但让实现这个接口的类来决定实例化哪个类。工厂方法让类的实例化推迟到子类中进行。”

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.framework.ProxyFactory

原型模式（Prototype）实现

- 基本概念

创建型模式的一种，其特点在于通过「复制」一个已经存在的实例来返回新的实例,而不是新建实例。被复制的实例就是我们所称的「原型」，这个原型是可定制的。

原型模式多用于创建复杂的或者耗时的实例，因为这种情况下，复制一个已经存在的实例使程序运行更高效；或者创建值相等，只是命名不一样的同类数据。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.target.PrototypeTargetSource

单例模式（Singleton）实现

- 基本概念

属于创建型模式的一种。在应用这个模式时，单例对象的类必须保证只有一个实例存在。许多时候整个系统只需要拥有一个的全局对象，这样有利于我们协调系统整体的行为。比如在某个服务器程序中，该服务器的配置信息存放在一个文件中，这些配置数据由一个单例对象统一读取，然后服务进程中的其他对象再通过这个单例对象获取这些配置信息。这种方式简化了在复杂环境下的配置管理。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.target.SingletonTargetSource

适配器模式（Adapter）实现

- 基本概念

有时候也称包装样式或者包装（英語：wrapper）。将一个类的接口轉接成用户所期待的。一个适配使得因接口不兼容而不能在一起工作的类能在一起工作，做法是将类自己的接口包裹在一个已存在的类中。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.framework.adapter.AdvisorAdapter
  - 适配对象 - org.aopalliance.aop.Advice
  - 目标对象 - org.aopalliance.intercept.MethodInterceptor

组合模式（Composite）实现

- 基本概念

The composite pattern describes a group of objects that are treated the same way as a single instance of the same type of object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.support.ComposablePointcut
  - 接口 - org.springframework.aop.Pointcut
  - 成员 - org.springframework.aop.Pointcut

装饰器模式（Decorator）实现

- 基本概念

一种动态地往一个类中添加新的行为的设计模式。就功能而言，修饰模式相比生成子类更为灵活，这样可以给某个对象而不是整个类添加一些功能。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.aspectj.annotation.LazySingletonAspectInstanceFactoryDecorator

享元模式（Flyweight）实现

- 基本概念

它使用物件用來儘可能減少記憶體使用量；於相似物件中分享儘可能多的資訊。當大量物件近乎重複方式存在，因而使用大量記憶體時，此法適用。通常物件中的部分狀態(state)能夠共享。常見做法是把它們放在資料結構外部，當需要使用時再將它們傳遞給享元。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.framework.adapter.AdvisorAdapterRegistry

代理模式（Proxy）实现

- 基本概念

所謂的代理者是指一個類別可以作為其它東西的介面。代理者可以作任何東西的介面：網路連接、記憶體中的大物件、檔案或其它昂貴或無法複製的資源。

- Spring AOP 举例实现
  - 实现 - org.springframework.aop.framework.AopProxy
    - JDK - org.springframework.aop.framework.JdkDynamicAopProxy
    - CGLIB - org.springframework.aop.framework.CglibAopProxy

模板方法模式（Template Method）实现

- 基本概念

模板方法是一個定義在父類別的方法，在模板方法中會呼叫多個定義在父類別的其他方法，而這些方法有可能只是抽象方法並沒有實作，模板方法僅決定這些抽象方法的執行順序，這些抽象方法的實作由子類別負責，並且子類別不允許覆寫模板方法。

- Spring AOP 举例实现
  - 模板类 - org.springframework.aop.framework.autoproxy.AbstractAutoProxyCreator
  - 模板方法 - getAdvicesAndAdvisorsForBean(Class,String,TargatSource)
  - 子类实现
    - org.springframework.aop.framework.autoproxy.InfrastructureAdvisorAutoProxyCreator(XML)
    - org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator(注解)

责任链模式（Chain of Responsibility）实现

- 基本概念

包含了一些命令对象和一系列的处理对象。每一个处理对象决定它能处理哪些命令对象，它也知道如何将它不能处理的命令对象传递给该链中的下一个处理对象。该模式还描述了往该处理链的末尾添加新的处理对象的方法。

- Spring AOP 举例实现
  - 接口 - org.springframework.aop.framework.AdvisorChainFactory
  - 实现 - org.springframework.aop.framework.DefaultAdvisorChainFactory

观察者模式（Observer）实现

- 基本概念

一個目標物件管理所有相依於它的觀察者物件，並且在它本身的狀態改變時主動發出通知。這通常透過呼叫各觀察者所提供的方法來實現。此種模式通常被用來實时事件處理系統。

- Spring AOP 举例实现
  - 观察者 - org.springframework.aop.framework.ProxyCreatorSupport
  - 被观察者 - org.springframework.aop.framework.AdvisedSupportListener
  - 通知对象 - org.springframework.aop.framework.AdvisedSupport

策略模式（Strategy）实现

- 基本概念

指對象有某個行爲，但是在不同的場景中，該行爲有不同的實現算法。比如每個人都要“交個人所得稅”，但是每國交個人所得稅就有不同的算稅方法。

- Spring AOP 举例实现
  - org.springframework.aop.framework.DefaultAopProxyFactory#createAopProxy
  - org.springframework.aop.config.ConfigBeanDefinitionParser#getAdviceClass

命令模式（Command）实现

- 基本概念

它嘗試以物件來代表實際行動。命令物件可以把行動(action) 及其參數封裝起來，於是這些行動可以被：

- 重複多次
- 取消（如果該物件有實作的話）
- 取消後又再重做

- Spring AOP 举例实现
  - org.aopalliance.intercept.MethodInvocation
  - org.aspectj.lang.ProceedingJoinPoint

状态模式（State）实现

- 基本概念

允许对象在内部状态发生变化时更改其行为。 这种模式接近于有限状态机的概念。 状态模式可以解释为策略模式，它能够通过调用模式接口中定义的方法来切换策略。

- Spring AOP 举例实现
  - 状态对象 - org.springframework.aop.framework.ProxyConfig
  - 影响对象 - org.springframework.aop.framework.AopProxy
    - org.springframework.aop.framework.JdkDynamicAopProxy
    - org.springframework.aop.framework.CglibAopProxy

面试题精选

沙雕面试题 - GoF 23 设计模式和它的归类？

答：

- Creational（创建模式）：Abstract factory、Builder、Factory method、Prototype 和 Singleton
- Structural（结构模式）：Adapter、Bridge、Composite、Decorator、Facade、Flyweight 和 Proxy
- Behavioral（行为模式）：Chain of responsibility、Command、Interpreter、Iterator、Mediator、Memento、Observer、State、Strategy、Template method 和 Visitor

面试题精选

996 面试题 - 举例介绍装饰器模式和代理模式的区别？

答：代理模式不要求和被代理对象存在层次关系
装饰器模式则需要和被装饰者存在层次关系
...

面试题精选

劝退面试题 - 请举例说明 Spring Framework 中使用设计模式的实现?

答：这个答案需要你自行探索。