

第二章：Spring AOP 基础

小马哥 (mercyblitz)

Spring AOP 基础

1. Spring 核心基础：《小马哥讲 Spring核心编程思想》还记得多少？
2. @AspectJ 注解创建代理
3. 编程方式创建 @AspectJ 代理
4. XML 配置驱动 - 创建 AOP 代理
5. 标准代理工厂 API - ProxyFactory
6. @AspectJ Pointcut 指令与表达式
7. XML 配置 Pointcut
8. API 实现 Pointcut
9. @AspectJ 拦截动作：@Around 与 @Pointcut 有区别吗？
10. XML 配置 Around Advice

Spring AOP 总览

11. API 实现 Around Advice
12. @AspectJ 前置动作: @Before 与 @Around 谁优先级执行?
13. XML 配置 Before Advice
14. API 实现 Before Advice
15. @AspectJ 后置动作 - 三种 After Advice 之间的关系?
16. XML 配置三种 After Advice
17. API 实现三种 After Advice
18. 自动动态代理
19. 替换 TargetSource
20. 面试题精选

Spring 核心基础

- 《小马哥讲 Spring核心编程思想》
 - 第三章：Spring IoC 容器概述
 - 第九章：Spring Bean 生命周期 (Bean Lifecycle)
 - 第十章：Spring 配置元信息 (Configuration Metadata)
 - 第十八章：Spring 注解 (Annotations)
 - 第二十章：Spring IoC 容器生命周期 (Container Lifecycle)

@AspectJ 注解驱动

- 激活 @AspectJ 模块
 - 注解激活 - @EnableAspectJAutoProxy
 - XML 配置 - <aop:aspectj-autoproxy/>
- 声明 Aspect
 - @Aspect

编程方式创建 @AspectJ 代理

- 实现类
 - `org.springframework.aop.aspectj.annotation.AspectJProxyFactory`

XML 配置驱动 - 创建 AOP 代理

- 实现方法
 - 配置 `org.springframework.aop.framework.ProxyFactoryBean`
 - Spring Schema-Based 配置
 - `<aop:config>`
 - `<aop:aspectj-autoproxy/>`

标准代理工厂 API

- 实现类 - `org.springframework.aop.framework.ProxyFactory`

@AspectJ Pointcut 指令与表达式

- 支持的指令
 - `execution`: For matching method execution join points. This is the primary pointcut designator to use when working with Spring AOP.
 - `within`: Limits matching to join points within certain types (the execution of a method declared within a matching type when using Spring AOP).
 - `this`: Limits matching to join points (the execution of methods when using Spring AOP) where the bean reference (Spring AOP proxy) is an instance of the given type.
 - `target`: Limits matching to join points (the execution of methods when using Spring AOP) where the target object (application object being proxied) is an instance of the given type.
 - `args`: Limits matching to join points (the execution of methods when using Spring AOP) where the arguments are instances of the given types.

@AspectJ Pointcut 指令与表达式

- 支持的指令
 - @target: Limits matching to join points (the execution of methods when using Spring AOP) where the class of the executing object has an annotation of the given type.
 - @args: Limits matching to join points (the execution of methods when using Spring AOP) where the runtime type of the actual arguments passed have annotations of the given types.
 - @within: Limits matching to join points within types that have the given annotation (the execution of methods declared in types with the given annotation when using Spring AOP).
 - @annotation: Limits matching to join points where the subject of the join point (the method being run in Spring AOP) has the given annotation.

@AspectJ Pointcut 指令与表达式

- 不支持的指令
 - `call`, `get`, `set`, `preinitialization`, `staticinitialization`, `initialization`, `handler`, `adviceexecution`, `withincode`, `cflow`, `cflowbelow`, `if`, `@this`, 和 `@withincode`

XML 配置 Pointcut

- XML 配置
 - `<aop:pointcut />`

API 实现 Pointcut

- 核心 API – `org.springframework.aop.Pointcut`
 - `org.springframework.aop.ClassFilter`
 - `org.springframework.aop.MethodMatcher`
- 适配实现 – `DefaultPointcutAdvisor`

@AspectJ 拦截动作

- 注解 - @Around
 - 与 @Pointcut 有什么区别?

XML 配置 Around Advice

- XML 配置
 - `<aop:around />`

API 实现 Around Advice

- 思考：为什么 Spring AOP 不需要设计 Around Advice?
 - 线索
 - AspectJ @Around 与 org.aspectj.lang.ProceedingJoinPoint 配合执行被代理方法
 - ProceedingJoinPoint#proceed() 方法类似于 Java Method#invoke(Object, Object...)
 - Spring AOP 底层 API ProxyFactory 可通过 addAdvice 方法与 Advice 实现关联
 - 接口 Advice 是 Interceptor 的父亲接口，而接口 MethodInterceptor 又扩展了 Interceptor
 - MethodInterceptor 的 invoke 方法参数 MethodInvocation 与 ProceedingJoinPoint 类似

@AspectJ 前置动作

- Before Advice 注解 - `@org.aspectj.lang.annotation.Before`
- 思考 1: @Before 与 @Around 谁优先级执行?
 - 线索: 运行一下不就知道了吗?
- 思考 2: 多个 @Before 声明后, 如何控制它们的执行顺序?
 - 线索: 看看官方文档怎么说?

XML 配置 Before Advice

- XML 元素 - `<aop:before>`
 - 声明规则
 - `<aop:config>`
 - `<aop:aspect>`
 - `<aop:before>`
- 属性设置（来源于 Spring AOP Schema 类型 `basicAdviceType`）
 - `pointcut`: Pointcut 表达式内容
 - `pointcut-ref`: Pointcut 表达式名称

API 实现 Before Advice

- 核心接口 - `org.springframework.aop.BeforeAdvice`
 - 类型: 标记接口, 与 `org.aopalliance.aop.Advice` 类似
- 方法 `JoinPoint` 扩展 - `org.springframework.aop.MethodBeforeAdvice`
- 接受对象 - `org.springframework.aop.framework.AdvisedSupport`
 - 基础实现类 - `org.springframework.aop.framework.ProxyCreatorSupport`
 - 常见实现类
 - `org.springframework.aop.framework.ProxyFactory`
 - `org.springframework.aop.framework.ProxyFactoryBean`
 - `org.springframework.aop.aspectj.annotation.AspectJProxyFactory`

@AspectJ 后置动作

- After Advice 注解
 - 方法返回后: `@org.aspectj.lang.annotation.AfterReturning`
 - 异常发生后: `@org.aspectj.lang.annotation.AfterThrowing`
 - finally 执行: `@org.aspectj.lang.annotation.After`

XML 配置 After Advice

- XML 元素 - `<aop:after>`
 - 声明规则
 - `<aop:config>`
 - `<aop:aspect>`
 - `<aop:after>`
 - 属性设置（来源于 Spring AOP Schema 类型 `basicAdviceType`）
 - `pointcut`: Pointcut 表达式内容
 - `pointcut-ref`: Pointcut 表达式名称

API 实现三种 After Advice

- 核心接口 - `org.springframework.aop.AfterAdvice`
 - 类型: 标记接口, 与 `org.aopalliance.aop.Advice` 类似
 - 扩展
 - `org.springframework.aop.AfterReturningAdvice`
 - `org.springframework.aop.ThrowsAdvice`
- 接受对象 - `org.springframework.aop.framework.AdvisedSupport`
 - 基础实现类 - `org.springframework.aop.framework.ProxyCreatorSupport`
 - 常见实现类
 - `org.springframework.aop.framework.ProxyFactory`
 - `org.springframework.aop.framework.ProxyFactoryBean`
 - `org.springframework.aop.aspectj.annotation.AspectJProxyFactory`

自动动态代理

- 代表实现
 - `org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator`
 - `org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator`
 - `org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator`

替换 TargetSource

- 代表实现
 - `org.springframework.aop.target.HotSwappableTargetSource`
 - `org.springframework.aop.target.AbstractPoolingTargetSource`
 - `org.springframework.aop.target.PrototypeTargetSource`
 - `org.springframework.aop.target.ThreadLocalTargetSource`
 - `org.springframework.aop.target.SingletonTargetSource`

面试题精选

沙雕面试题 – Spring AOP 支持哪些类型的 Advice?



我真的没笑

答:

- Around Advice
- Before Advice
- After Advice
 - After
 - AfterReturning
 - AfterThrowing

面试题精选

996 面试题 – Spring AOP 编程模型有哪些，代表组件有哪些？



答：

注解驱动：解释和整合 AspectJ 注解，如 `@EnableAspectJAutoProxy`

XML 配置：AOP 与 IoC Schema-Based 相结合

API 编程：如 Joinpoint、Pointcut、Advice 和 ProxyFactory 等

面试题精选

劝退面试题 - Spring AOP 三种实现方式是如何设计的?



答：这个问题的答案将贯穿整个系列

