
回到首页

芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2020-01-01
Spring

精尽 Spring 源码分析 —— AOP 源码简单导读

1. 前置内容

Spring AOP 是基于 Spring IoC 机制来实现的，所以建议对 IoC 的源码有一定的了解。

当然，一般情况下，也是先看完 IoC 相关的源码，在来看 AOP 的源码。

2. 如何调试

① 调试基于 `@Aspect` 注解的 AOP 的流程

可调试 `org.springframework.aop.aspectj.autoproxy.AspectJAutoProxyCreatorTests` 这个单元测试里的方法。

3. 推荐资料

【必读】首先，推荐的是《Spring 源码深度解析》的 「第7章 AOP」 章节。
因为 Spring 静态 AOP 实际场景下，使用较少，胖友可以选择性看看。感兴趣的话，可以对 Java Instrumentation 机制做一定的了解。目前主流的链路追踪系统的 Java Agent 都是基于它来实现的。

然后，也推荐看看五月的仓颉关于 AOP 的两篇文章：

> 《【Spring源码分析】AOP源码解析（上篇）》 ，对 Spring AOP XML 配置的方式进行源码解析。
> 《【Spring源码分析】AOP源码解析（下篇）》 ，内容上，和《Spring 源码深度解析》的 「第7章 AOP」 章节，作为互为补充噶。

再然后，也非常推荐田小波关于 AOP 的四篇文章，非常细致，特别是对 AOP 的概念解释以及流程的拆分上：

> 《Spring AOP 源码分析系列文章导读》
> 《Spring AOP 源码分析 - 筛选合适的通知器》
> 《Spring AOP 源码分析 - 创建代理对象》
> 《Spring AOP 源码分析 - 拦截器链的执行过程》

再再然后，还会灰常推荐 JavaDoop 关于 AOP 的一篇文章，看着大气：

> 《Spring AOP 源码解析》

最后，推荐一些和 AOP 相关的有趣的文章：

> 《Spring AOP,AspectJ, CGLIB 有点晕》
> 《Cglib 与 JDK 动态代理的运行性能比较》
> 《Spring AOP 源码解析》

4. 后置内容

看完 Spring AOP 之后，当然是看 Spring Transaction 啦。

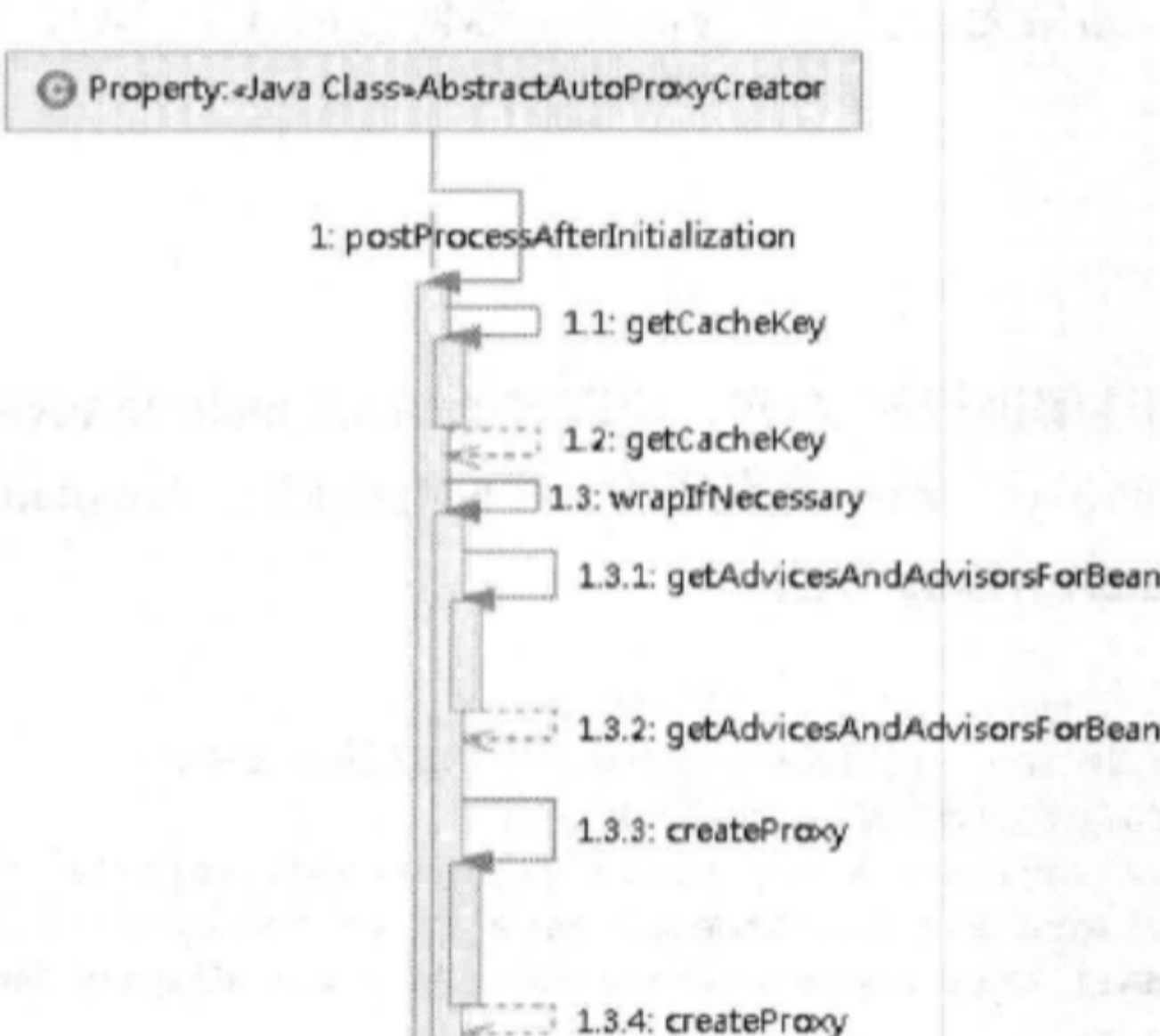
哈哈哈哈，和我们日常开发，息息相关。

5. 重要的类

> 老艿艿：本小节，就是艿艿简单的笔记，可以忽略。嘿嘿。后面，在找时间完善下。

核心流程如下图：

> FROM 《Spring 源码深度解析》



5.1 aopalliance

对应 `org.aopalliance` 包。

Spring 对 AOP 联盟接口的变种。

Advice
- Interceptor
  - MethodInterceptor

Joinpoint ，连接点接口。每个方法，都对应一个 Joinpoint 对象。
- Invocation ，调用接口。
  - MethodInvocation ，方法调用接口。
    - ProxyMethodInvocation ，代理方法调用接口。在根目录的包里。
      - ReflectiveMethodInvocation ，反射方法调用实现类。在 `framework` 包里。
        - CglibMethodInvocation ，基于 CGLIB 方法调用实现类。在 `framework` 包里。

Advice ，定义的横切逻辑。在如下几个时机，可以进行执行：
- Before ：在目标方便调用前执行通知。
- After ：在目标方法完成后执行通知。
- After Returning ： 在目标方法执行成功后，调用通知。
- After Throwing ：在目标方法抛出异常后，执行通知。
- Around ：在目标方法调用前后均可执行自定义逻。

5.2 config

对应 `org.springframework.aop.config` 包。

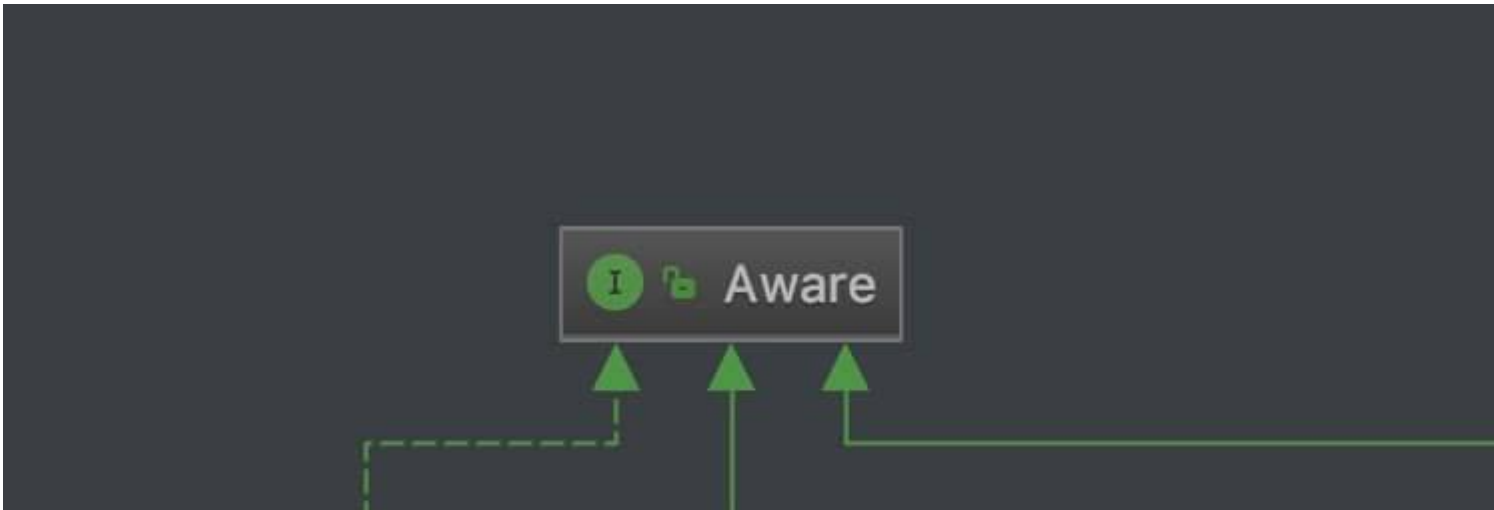
AopNamespaceHandler ，对 `<aop:/>` 命名空间的处理器。

AspectJAutoProxyBeanDefinitionParser ，将 `<aop:aspectj-autoproxy/>` 标签，解析成 BeanDefinition 的解析器。
ScopedProxyBeanDefinitionDecorator
ConfigBeanDefinitionParser
SpringConfiguredBeanDefinitionParser

5.3 framework

对应 `org.springframework.aop.framework` 包。

AutoProxyCreator 整体类图如下：



AbstractAutoProxyCreator ，自动代理构造器抽象类。

- #wrapIfNecessary(Object bean, String beanName, Object cacheKey) 方法，如果需要，将制定 Bean 对象包装成 Proxy 代理对象。
- #createProxy(Class<?> beanClass, String beanName, Object[] specificInterceptors, TargetSource targetSource) 方法，创建 Proxy 代理对象。

AbstractAdvisorAutoProxyCreator ，基于 Advisor 的 自动代理构造器抽象类。

- #findAdvisorsThatCanApply(List<Advisor> candidateAdvisors, Class<?> beanClass, String beanName) 方法，获得 Bean 对象可使用的 Advisor 集合，是从 candidateAdvisors 参数中筛选出来的。
- #getAdvicesAndAdvisorsForBean(Class<?> beanClass, String beanName, TargetSource targetSource) 方法，获得 Bean 对象可使用的 Advisor 集合。

AnnotationAwareAspectJAutoProxyCreator ，支持 AOP 注解的，基于 AspectJ 的代理自动构造器。

ProxyFactory ，创建 Proxy 的工厂类。

- DefaultAopProxyFactory ，默认 ProxyFactory 实现类。

AopProxy ，AOP 代理接口，创建对应的代理对象。

- 它有两个子类：
  - JdkDynamicAopProxy ，基于 JDK 的 AOP 代理实现类。
    - #invoke(Object proxy, Method method, Object[] args) 方法，实现 InvocationHandler 的接口方法，执行代理的调用。【重要】
  - ObjenesisCglibAopProxy ，基于 CGLIB 的 AOP 的代理实现类。
- #getProxy(ClassLoader classLoader) 方法，创建对应的代理对象。

AopContext ，AOP 上下文，基于 ThreadLocal 实现。

AdvisorChainFactory ，Advisor 调用链工厂接口，用于获取每个方法调用时的调用链。

- #getInterceptorsAndDynamicInterceptionAdvice(Advised config, Method method, Class<?> targetClass) 方法，获得指定方法拦截的拦截器链。
- DefaultAdvisorChainFactory ，默认实现类。

ProxyMethodInvocation ，代理方法调用接口。

- ReflectiveMethodInvocation ，基于反射的方式，代理方法调用实现类。
  - #proceed() 方法，执行方法。基于递归的方式，调用每个拦截器链中的拦截器，最后调用真正的方法。
  - #invokeJoinpoint() 方法，执行真正的方法，即切点的方法。
  - CglibMethodInvocation ，基于 CGLIB 的方式，进一步优化调用的实现类。

各种情况下，调用：

- MethodBeforeAdviceInterceptor
- AfterReturningAdviceInterceptor
- AspectJAroundAdvice
- AspectJAfterAdvice
- AspectJAfterThrowingAdvice
- ThrowsAdviceInterceptor

5.4 aspectj

对应 org.springframework.aop.aspectj 包。

ReflectiveAspectJAdvisorFactory

BeanFactoryAspectJAdvisorsBuilder

- #buildAspectJAdvisors() 方法，构建所有 @Aspect 注解类的增强方法对应的 Advisor 对象。

AnnotationAwareAspectJAutoProxyCreator

- #findCandidateAdvisors() 方法，获得所有增强方法对应的 Advisor 对象。

InstantiationModelAwarePointcutAdvisorImpl

AbstractAspectJAdvice ，基于 AspectJ ，实现 Advice 接口的 Advice 抽象类。

- 有如下几个子类：
  - AspectJAroundAdvice
  - AspectJMethodBeforeAdvice
  - AspectJAfterAdvice
  - AspectJAfterReturningAdvice
  - AspectJAfterThrowingAdvice
- 通过 `ReflectiveAspectJAdvisorFactory#getAdvice(...)` 方法，构建 AbstractAspectJAdvice 对象。
- 通过 `#invokeAdviceMethodWithGivenArgs(Object[] args)` 方法，执行增强 Advice 的方法。【重要】

5.5 interceptor

对应 `org.springframework.aop.interceptor` 包。

ExposeInvocationInterceptor

5.6 scope

对应 `org.springframework.aop.scope` 包。

5.7 support

对应 `org.springframework.aop.support` 包。

5.8 target

对应 `org.springframework.aop.target` 包。

提供多种 TargetSource 的实现类，而 TargetSource 可以理解成 Target 的数据源，可以根据每次方法调用，返回相同或不同的 Target 对象。可以选择性看看，感兴趣的胖友，可以看看 《spring-aop组件详解——TargetSource目标源》 。

5.9 根目录

对应 `org.springframework.aop` 包的根目录。

Pointcut ，切点接口，用于选择连接点。
- 其定义了两个接口方法，分别返回：
  - ClassFilter ，匹配哪些类。
  - MethodMatcher ，匹配哪些方法。
- AspectJExpressionPointcut ，AspectJ 的切点表达式。

文章目录

回到首页