https://blog.csdn.net/J080624/article/details/53996875

**Spring - 通知（Advice）和Advisor(增强器/顾问)**

2017年01月03日 18:29:10 [流烟默](https://me.csdn.net/J080624) 阅读数：2669

 版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/J080624/article/details/53996875

**【1】AOP (Aspect Oriented Programming 面向切面编程)**

**首先回顾基本术语：**

**（1）切面(Aspect)**

切面泛指[交叉业务逻辑]，类级别，包含了同一类型事务的不同增强方法(通知)。

比如：事务处理和日志处理可以理解为两个切面。

通知(Advice)实际就是对主业务逻辑的一种增强，是一种在目标方法执行之外额外的方法。

**如，日志处理–在目标方法执行前后添加日志。**

这些被添加日志的方法称为连接点；   
·   
通知方法执行的位置称为切入点；   
·   
对目标方法增强的方法称为通知；   
·   
对日志增强的所有通知方法所属的类称为切面。

**(2)织入（Weaving）**

把切面（aspect）连接到其它的应用程序类型或者对象上，并创建一个被通知（advised）的对象。 这些可以在编译时（例如使用AspectJ编译器），类加载时和运行时完成。

织入是指将切面代码插入到目标对象的过程。代理的invoke方法完成的工作，可以称为织入。

Spring和其他纯Java AOP框架一样，在运行时完成织入。

**（3） 连接点(JoinPoint)**

程序执行过程中明确的点，如方法的调用或特定的异常被抛出。

连接点由两个信息确定：

方法(表示程序执行点,即-在哪个目标方法)；

相对点(表示方位，即-目标方法的什么位置，如调用前、后。。。)

* 1

连接点是程序类中客观存在的事物。

**（4）切入点(PointCut)**

匹配连接点（Joinpoint）的断言。

通知和一个切入点表达式关联，并在满足这个切入点的连接点上运行（例如，当执行某个特定名称的方法时）。

切入点表达式如何和连接点匹配是AOP的核心：Spring缺省使用AspectJ切入点语法。

也就是说通过某种定义来决定哪些方法将会被通知（即需要事物处理），我们将这个定义秤为切入点（Pointcut），而每一个被处理的方法我们称之为连接点（Join Point）。

注意：被标记为final的方法是不能作为连接点与切入点的。因为最终的是不能被修改的，不能被增强的。

**类比：连接点相当于数据库中的记录，切点相当于查询条件。**

或者说，切点为AOP Aspect在系统中的入口，而JoinPoint是在系统中执行的地方。

切点和连接点不是一对一的关系，一个切点一般匹配多个连接点.

切点通过 org.springframework.aop.Pointcut 接口进行描述，它使用类和方法作为连接点的查询条件。

**(5)目标对象（Target）**

目标对象指将要被增强的对象，即包含主业务逻辑的类的对象。

或者说是被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 被通知（advised） 对象。

既然Spring AOP是通过运行时代理实现的，这个对象永远是一个 被代理（proxied） 对象。

**（6）通知（Advice）**

在切面的某个特定的连接点（Joinpoint）上执行的动作。通知是切面的一种功能实现，可以完成简单的织入功能。

通知定义了**增强代码切入到目标代码的时间点**(是目标方法执行之前执行，还是执行之后执行等)和**增强功能**。完整通知包括通知类型+切入点表达式+增强方法。

如下所示：

*//抽取公共的切入点表达式*

*//1、本类引用*

*//2、其他的切面引用*

@Pointcut("execution(public int com.web.aop.MathCalculator.\*(..))")

public void pointCut(){};

*//@Before在目标方法之前切入；切入点表达式（指定在哪个方法切入）*

@Before("pointCut()")

public void logStart(JoinPoint joinPoint){

Object[] args = joinPoint.getArgs();

System.out.println(""+joinPoint.getSignature().getName()+"运行。。。@Before:参数列表是：{"+Arrays.asList(args)+"}");

}

* 1

综上，切入点定义切入的位置，通知定义切入的时间和增强功能。通知需要与切入点配合使用。

**（7）Advisor(增强器/顾问)**

Advisor是切面的另一种实现，能够将通知以更为复杂的方式织入到目标对象中，是将通知包装为更复杂切面的装配器。Advisor由切入点和Advice组成。

*<!-- 拦截器方式配置事物 -->*

<tx:advice id="transactionAdvice" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="add\*" propagation="REQUIRED"/>

<tx:method name="append\*" propagation="REQUIRED"/>

<tx:method name="insert\*" propagation="REQUIRED"/>

<tx:method name="save\*" propagation="REQUIRED"/>

<tx:method name="update\*" propagation="REQUIRED"/>

<tx:method name="modify\*" propagation="REQUIRED"/>

<tx:method name="edit\*" propagation="REQUIRED"/>

<tx:method name="delete\*" propagation="REQUIRED"/>

<tx:method name="remove\*" propagation="REQUIRED"/>

<tx:method name="repair" propagation="REQUIRED"/>

<tx:method name="release\*" propagation="REQUIRED"/>

<tx:method name="delAndRepair" propagation="REQUIRED"/>

<tx:method name="get\*" propagation="SUPPORTS"/>

<tx:method name="find\*" propagation="SUPPORTS"/>

<tx:method name="load\*" propagation="SUPPORTS"/>

<tx:method name="search\*" propagation="SUPPORTS"/>

<tx:method name="datagrid\*" propagation="SUPPORTS"/>

<tx:method name="\*" propagation="SUPPORTS"/>

</tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut id="transactionPointcut" expression="execution(\* com.sychine.tower.service..\*Impl.\*(..))"/>

<aop:advisor pointcut-ref="transactionPointcut" advice-ref="transactionAdvice"/>

</aop:config>

* 1

“advisors”这个概念来自Spring1.2对AOP的支持，在AspectJ中是没有等价的概念。 advisor就像一个小的自包含的切面，这个切面只有一个通知。 切面自身通过一个bean表示，并且必须实现一个通知接口，大多数情况下会和transactional advice一起使用。

**【2】Aspect**

使用<aop:aspect>配置时,如果切面代码是自动注入的bean,那么<aop:aspect>的ref属性直接写bean的注入名字就可以了!

<bean id="validateAspect" class="com.web.aop.impl.xml.ValidateAspect">

</bean>

<aop:config>

<aop:pointcut id="pointcut" expression="execution(\* com.web.aop.impl.xml.ArithmeticCalculatorImpl.\*(int, int))" />

*<!-- 配置验证切面 -->*

<aop:aspect ref="validateAspect" order="2" >

<aop:before method="beforMethod" pointcut-ref="pointcut"/>

</aop:aspect>

</aop:config>

* 1

**【3】Advisor**

如，Spring中的事务应用：

*<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*2.配置事务属性 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->*

<tx:advice id="txadvice" transaction-manager="transactionManager" >

<tx:attributes>

*<!-- 根据方法名指定事务的属性 -->*

<tx:method name="purchase" propagation="REQUIRES\_NEW" isolation="READ\_COMMITTED" />

<tx:method name="delete\*" propagation="REQUIRED"/>

<tx:method name="remove\*" propagation="REQUIRED"/>

<tx:method name="search\*" propagation="SUPPORTS"/>

<tx:method name="datagrid\*" propagation="SUPPORTS"/>

<tx:method name="\*" propagation="SUPPORTS"/>

</tx:attributes>

</tx:advice>

*<!-- \*\*\*3.配置事务切入点，以及把事务切入点和事务属性关联起来 -->*

<aop:config>

<aop:pointcut id="txPointCut" expression="execution(\* com.web.xml.service.\*.\*(..))" />

<aop:advisor advice-ref="txadvice" pointcut-ref="txPointCut" />

</aop:config>

* 1

若<aop:advisor advice-ref="beanName" pointcut-ref="txPointCut" />，   
则该bean的class应为org.aopalliance.aop.Advice类型

**【Tips】**

Adivisor是一种特殊的Aspect，Advisor代表spring中的Aspect 。

● <aop:advisor>：用来定义只有一个通知和一个切入点的切面；

● <aop:aspect>：用来定义切面，该切面可以包含多个切入点和通知，而且标签内部的通知和切入点定义是无序的。其和advisor的区别就在此，advisor只包含一个通知和一个切入点。

**aop:aspect配置如下：**

<bean id="arithmeticCalculator" class="com.web.aop.impl.xml.ArithmeticCalculatorImpl"></bean>

*<!-- 配置切面bean -->*

<bean id="loggingAspect" class="com.web.aop.impl.xml.LoggingAspect"></bean>

<bean id="validateAspect" class="com.web.aop.impl.xml.ValidateAspect"></bean>

*<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AOP 配置\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->*

<aop:config>

*<!-- 配置切点表达式 -->*

<aop:pointcut id="pointcut" expression="execution(\* com.web.aop.impl.xml.ArithmeticCalculatorImpl.\*(int, int))" />

*<!-- 配置日志切面 -->*

<aop:aspect ref="loggingAspect" order="1" >

<aop:before method="beforMethod" pointcut-ref="pointcut"/>

<aop:after method="afterMethod" pointcut-ref="pointcut"/>

<aop:after-returning method="afterReturning" pointcut-ref="pointcut" returning="result"/>

<aop:after-throwing method="afterThrowing" pointcut-ref="pointcut" throwing="exception"/>

*<!-- <aop:around method="aroundMethod" pointcut-ref="pointcut" />-->*

</aop:aspect>

*<!-- 配置验证切面 -->*

<aop:aspect ref="validateAspect" order="2" >

<aop:before method="beforMethod" pointcut-ref="pointcut"/>

</aop:aspect>

</aop:config>

* 1

 收藏

 分享