 源自：http://blog.csdn.net/z69183787/article/details/8348498

[Spring3核心技术之AOP配置](http://blog.csdn.net/z69183787/article/details/8348498)

2012-12-20 09:53 2544人阅读 [评论](http://blog.csdn.net/z69183787/article/details/8348498#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/z69183787/article/details/8348498#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

SPRING-AOP（20） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg SPRING（92） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

在Spring配置文件中，所有AOP相关定义必须放在<aop:config>标签下，该标签下可以有<aop:pointcut>、<aop:advisor>、<aop:aspect>标签，配置顺序不可变。   
   
  
**● <aop:pointcut>**：用来定义切入点，该切入点可以重用；   
**● <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；和advisor的区别就在此，advisor只包含一个通知和一个切入点。

**Java代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **public** **class** Interceptor {
3. **public** **void** beforeDomain() {
4. System.out.println("This is beforeDomain....");
5. }
7. **public** **void** afterDomain() {
8. System.out.println("This is afterDomain....");
9. }
11. **public** **void** afterReturning() {
12. System.out.println("This is afterReturning....");
13. }
15. **public** **void** afterThrowing() {
16. System.out.println("This is afterThrowing....");
17. }
19. **public** Object around(ProceedingJoinPoint pjp) **throws** Throwable {
20. System.out.println("===========around before advice");
21. Object retVal = pjp.proceed(**new** Object[] {"【环绕通知】"});
22. System.out.println("===========around after advice");
23. **return** retVal;
24. }
25. }

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. .....
3. **<bean** id="aspectBean" class="com.chou.spring.domain.Interceptor"**/>**
5. **<aop:config** proxy-target-class="false"**>**
6. **<aop:aspect** ref="aspectBean"**>**
7. <!-- 定义切入点 -->
8. **<aop:pointcut** id="myAspect"
9. expression="execution(public \* com.chou.spring.bean..\*.domain(..))" **/>**
11. <!-- 前置通知 -->
12. **<aop:before** pointcut-ref="myAspect" method="prepareDomain"**/>**
14. <!-- 后置通知 -->
15. **<aop:after-returning** pointcut-ref="myAspect" method="afterReturning"**/>**
16. **<aop:after-throwing** pointcut-ref="myAspect" method="afterThrowing"**/>**
17. **<aop:after** pointcut-ref="myAspect" method="afterDomain"**/>**
19. <!-- 环绕通知 -->
20. **<aop:around** method="around"
21. pointcut="execution(\* com.chou.spring.bean..\*.sayAround(..))"**/>**
22. **</aop:aspect>**
23. **</aop:config>**

**Java代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **public** **interface** MyBean {
2. **public** **void** domain();
3. }

6. **public** **class** MyBeanA{
7. **public** **void** domain() {
8. System.out.println("MyBeanA is executing...");
9. }
11. **public** **void** sayAround(String param) {
12. System.out.println("around param:" + param);
13. }
14. }
16. **public** **class** MyBeanB **implements** MyBean{
17. **public** **void** domain() {
18. System.out.println("MyBeanB is executing...");
19. //throw new RuntimeException("This is a RuntimeException");
20. }
21. }
23. //main方法....
24. String[] configs = **new** String[] {"applicationContext-aop.xml"};
25. ApplicationContext cxt = **new** ClassPathXmlApplicationContext(configs);
26. //如果Bean有interface那么就用JDK的Proxy.newProxyInstance得到代理对象进行aop
27. MyBean b = (MyBean)cxt.getBean("beanB");
28. b.domain();
29. //如果Bean没有实现任何interface那么就用CGLIB得到代理对象进行aop
30. MyBeanA a = cxt.getBean("beanA",MyBeanA.**class**);
31. a.domain();
32. a.sayAround("jjjjjjjjjjjjjjjjjjj");

**声明切面**   
    切面就是包含切入点和通知的对象，在Spring容器中将被定义为一个Bean，xml形式的切面需要一个切面支持Bean，该支持Bean的字段和方法提供了切面的状态和行为信息，并通过配置方式来指定切入点和通知实现。   
    切面使用<aop:aspect>标签指定，ref属性用来引用切面支持Bean。   
    切面支持Bean“aspectSupportBean”跟普通Bean完全一样使用，切面使用“ref”属性引用它。   
  
**声明切入点**   
    切入点在Spring中也是一个Bean，Bean定义方式可以有很三种方式：   
● 在<aop:config>标签下使用<aop:pointcut>声明一个切入点Bean，该切入点可以被多个切面使用，对于需要共享使用的切入点最好使用该方式，该切入点使用id属性指定Bean名字，在通知定义时使用pointcut-ref属性通过该id引用切入点，expression属性指定切入点表达式。   
● 在<aop:aspect>标签下使用<aop:pointcut>声明一个切入点Bean，该切入点可以被多个切面使用，但一般该切入点只被该切面使用，当然也可以被其他切面使用，但最好不要那样使用，该切入点使用id属性指定Bean名字，在通知定义时使用pointcut-ref属性通过该id引用切入点，expression属性指定切入点表达式   
● 匿名切入点Bean，可以在声明通知时通过pointcut属性指定切入点表达式，该切入点是匿名切入点，只被该通知使用

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:config>**
2. **<aop:aspect** ref="aspectSupportBean"**>**
3. **<aop:after** pointcut="execution(\* cn.javass..\*.\*(..))" method="afterAdvice"**/>**
4. **</aop:aspect>**
5. **</aop:config>**

**关于切入点的expression表达式用法可以参考这个博客(xml和注解形式都通用)**：   
<http://jinnianshilongnian.iteye.com/blog/1415606>   
  
**声明通知：(前置通知，后置通知，环绕通知)**   
**一、前置通知**:在切入点选择的连接点处的方法之前执行的通知，该通知不影响正常程序执行流程（除非该通知抛出异常，该异常将中断当前方法链的执行而返回）。   
Spring中在切入点选择的方法之前执行，通过<aop:aspect>标签下的<aop:before>标签声明：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:before** pointcut="切入点表达式"  pointcut-ref="切入点Bean引用"
2. method="前置通知实现方法名" arg-names="前置通知实现方法参数列表参数名字"**/>**

● pointcut和pointcut-ref：二者选一，指定切入点；   
● method：指定前置通知实现方法名，如果是多态需要加上参数类型，多个用“，”隔开，如beforeAdvice(java.lang.String)；   
● arg-names：指定通知实现方法的参数名字，多个用“，”分隔，可选，切入点中使用“args(param)”匹配的目标方法参数将自动传递给通知实现方法同名参数。   
关于arg-names具体用法可以参考博客：<http://jinnianshilongnian.iteye.com/blog/1418598>   
  
**二、后置通知**:在切入点选择的连接点处的方法之后执行的通知，包括如下类型的后置通知：   
● 后置返回通知:在切入点选择的连接点处的方法正常执行完毕时执行的通知，必须是连接点处的方法没抛出任何异常正常返回时才调用后置通知。   
在切入点选择的方法正常返回时执行，通过<aop:aspect>标签下的<aop:after-returning>标签声明：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:after-returning** pointcut="切入点表达式"  pointcut-ref="切入点Bean引用"
2. method="后置返回通知实现方法名"
3. arg-names="后置返回通知实现方法参数列表参数名字"
4. returning="返回值对应的后置返回通知实现方法参数名"
5. **/>**

● 后置异常通知:在切入点选择的连接点处的方法抛出异常返回时执行的通知，必须是连接点处的方法抛出任何异常返回时才调用异常通知。   
在切入点选择的方法抛出异常时执行，通过<aop:aspect>标签下的<aop:after-throwing>标签声明：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:after-throwing** pointcut="切入点表达式"  pointcut-ref="切入点Bean引用"
2. method="后置异常通知实现方法名"
3. arg-names="后置异常通知实现方法参数列表参数名字"
4. throwing="将抛出的异常赋值给的通知实现方法参数名"**/>**

● 后置最终通知:在切入点选择的连接点处的方法返回时执行的通知，不管抛没抛出异常都执行，类似于Java中的finally块。   
在切入点选择的方法返回时执行，不管是正常返回还是抛出异常都执行，通过<aop:aspect>标签下的<aop:after >标签声明：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:after** pointcut="切入点表达式"  pointcut-ref="切入点Bean引用"
2. method="后置最终通知实现方法名"
3. arg-names="后置最终通知实现方法参数列表参数名字"**/>**

**三、环绕通知**：环绕着在切入点选择的连接点处的方法所执行的通知，环绕通知可以在方法调用之前和之后自定义任何行为，并且可以决定是否执行连接点处的方法、替换返回值、抛出异常等等。   
环绕着在切入点选择的连接点处的方法所执行的通知，环绕通知非常强大，可以决定目标方法是否执行，什么时候执行，执行时是否需要替换方法参数，执行完毕是否需要替换返回值，可通过<aop:aspect>标签下的<aop:around >标签声明：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:around** pointcut="切入点表达式"  pointcut-ref="切入点Bean引用"
2. method="后置最终通知实现方法名"
3. arg-names="后置最终通知实现方法参数列表参数名字"**/>**

环绕通知第一个参数必须是org.aspectj.lang.ProceedingJoinPoint类型，在通知实现方法内部使用ProceedingJoinPoint的proceed()方法使目标方法执行，proceed 方法可以传入可选的Object[]数组，该数组的值将被作为目标方法执行时的参数。   
  
**四、引入**   
    Spring允许为目标对象引入新的接口，通过在< aop:aspect>标签内使用< aop:declare-parents>标签进行引入，定义方式如下：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:declare-parents**
2. types-matching="AspectJ语法类型表达式"
3. implement-interface=引入的接口"
4. default-impl="引入接口的默认实现"
5. delegate-ref="引入接口的默认实现Bean引用"**/>**

具体用法请参考博客：<http://jinnianshilongnian.iteye.com/blog/1418598>   
  
**五、Advisor**   
Advisor表示只有一个通知和一个切入点的切面，由于Spring AOP都是基于AOP的拦截器模型的环绕通知的，所以引入Advisor来支持各种通知类型（如前置通知等5种），Advisor概念来自于Spring1.2对AOP的支持，在AspectJ中没有相应的概念对应。   
Advisor可以使用<aop:config>标签下的<aop:advisor>标签定义：

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. **<aop:advisor** pointcut="切入点表达式" pointcut-ref="切入点Bean引用"
2. advice-ref="通知API实现引用"**/>**
4. **<bean** id="beforeAdvice" class="cn.javass.spring.chapter6.aop.BeforeAdviceImpl"**/>**
5. **<aop:advisor** pointcut="execution(\* cn.javass..\*.sayAdvisorBefore(..))"
6. advice-ref="beforeAdvice"**/>**

**除了在进行事务控制的情况下，其他情况一般不推荐使用该方式，该方式属于侵入式设计，必须实现通知API**

**Xml代码  [收藏代码](http://blog.csdn.net/z69183787/article/details/8348498)**

1. <!-- 事务管理器配置,单数据源事务 -->
2. **<bean** id="transactionManager"
3. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
4. **<property** name="sessionFactory" ref="sessionFactory" **/>**
5. **</bean>**
7. **<aop:config>**
8. **<aop:advisor** pointcut="execution(\* com.spring.test.service..\*.\*(..))"
9. advice-ref="txAdvice" **/>**
10. **</aop:config>**
12. **<tx:advice** id="txAdvice" transaction-manager="transactionManager"**>**
13. **<tx:attributes>**
14. **<tx:method** name="get\*" read-only="true" **/>**
15. **<tx:method** name="find\*" read-only="true" **/>**
16. **<tx:method** name="list\*" read-only="true" **/>**
17. **<tx:method** name="save\*" **/>**
18. **<tx:method** name="update\*" **/>**
19. **<tx:method** name="delete\*" **/>**
20. **</tx:attributes>**
21. **</tx:advice>**