# 【第44话：Spring中通知到底有四种还是五种】

Hello 小伙伴们这节课给大家讲解一下：“Spring中通知的几种类型”。

在Spring框架中提供了两种方式实现AOP：

- Schema-based:所有的通知都需要实现特定类型的接口。
- AspectJ：可以使用普通Java类结合特定的配置标签实现通知。

其中Schema-based包含四种通知类型：

- 前置通知：在切入点之前执行的增强功能。通知需要实现MethodBeforeAdvice接口
- 后置通知：在切入点之后执行的增强功能。通知需要实现AfterReturningAdvice接口
- 环绕通知：一个方法包含了前置通知和后置通知的功能。通知需要实现MethodInterceptor接口
- 异常通知：如果切入点中出现了异常（绝对不能try...catch解决了异常）就会触发异常通知。通知需要实现ThrowsAdvice接口。

而AspectJ方式提供了五种通知：

- 前置通知before
- 后置通知分为两种：after表示无论是否出现异常都执行的后置通知。after-returning切入点不出现异常时才执行的后置通知
- 环绕通知around
- 异常通知after-throwing

所以在回答面试官这个问题时，不要上来就回答包含四种或包含五种。而是要先说分类，再说每个分类中包含的通知有哪些。

下面详细介绍一下这些通知。

先说一下Schema-based方式下实现的方式

**前置通知**

前置通知是在切入点之前执行的增强。

代码实现需要新建com.bjsxt.advice.MyBefore。

MethodBeforeAdvice接口中必须重写before方法，方法中三个参数：

- method：切入点方法对象
- args：切入点方法参数
- target：切入点方法所在的对象

```
package com.bjsxt.advice;

import org.springframework.aop.MethodBeforeAdvice;

import java.lang.reflect.Method;

public class MyBefore implements MethodBeforeAdvice {
    @Override
    public void before(Method method, Object[] args, Object target) throws
Throwable {
        System.out.println("前置通知");
    }
}
```

配置切面

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:component-scan base-
package="com.bjsxt.mapper,com.bjsxt.service.impl"></context:component-scan>

    <bean id="mybefore" class="com.bjsxt.advice.MyBefore"></bean>
    <aop:config>
        <!-- 切入点是固定语法：execution(* 方法的全限定路径带有参数) -->
        <!-- 无参方法：test()-->
        <!-- 有参方法：test(int,java.lang.String)-->
        <!-- 如果希望把所有叫做test的方法都匹配上，不考虑参数test(..)-->
        <!-- 全限定路径的层数绝对不能少，但是如果希望对类或包做统配使用*-->
        <!-- com.bjsxt.service.impl.*.*(..)-->
        <aop:pointcut id="mypoint" expression="execution(*
com.bjsxt.service.impl.PeopleServiceImpl.test())"/>
        <aop:advisor advice-ref="mybefore" pointcut-ref="mypoint"></aop:advisor>
    </aop:config>
</beans>
```

**后置通知**

后置通知是在切入点之后执行的增强。

新建通知类

新建com.bjsxt.advice.MyAfter。

AfterReturningAdvice接口中必须重写afterReturning方法，方法中四个参数：

- returnValue：返回值

- method：切入点方法对象
- args：切入点方法参数
- target：切入点方法所在的对象

```
package com.bjsxt.advice;

import org.springframework.aop.AfterReturningAdvice;
import java.lang.reflect.Method;

public class MyAfter implements AfterReturningAdvice {
    @Override
    public void afterReturning(Object returnValue, Method method, Object[] args,
Object target) throws Throwable {
        System.out.println("后置通知");
    }
}
```

配置切面

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:component-scan base-
package="com.bjsxt.mapper,com.bjsxt.service.impl"></context:component-scan>

    <bean id="mybefore" class="com.bjsxt.advice.MyBefore"></bean>
    <!-- 配置后置通知bean -->
    <bean id="myafter" class="com.bjsxt.advice.MyAfter"></bean>
    <aop:config>
        <aop:pointcut id="mypoint" expression="execution(*
com.bjsxt.service.impl.PeopleServiceImpl.test())"/>
        <aop:advisor advice-ref="mybefore" pointcut-ref="mypoint"></aop:advisor>
        <!-- 织入后置通知 -->
        <aop:advisor advice-ref="myafter" pointcut-ref="mypoint"></aop:advisor>
    </aop:config>
</beans>
```

**环绕通知**

环绕通知可以实现前置通知和后置通知两种功能。

新建通知类

新建com.bjsxt.advice.MyAround。

MethodInterceptor接口中必须重写invoke方法，方法中参数：

- invocation：方法调用器。通过invocation可以proceed()方法调用执行点。

```
package com.bjsxt.advice;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class MyAround implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation invocation) throws Throwable {
        System.out.println("环绕通知-前置增强");
        Object result = invocation.proceed();// 执行切入点
        System.out.println("环绕通知-后置增强");
        return result;
    }
}
```

配置切面

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:component-scan base-
package="com.bjsxt.mapper,com.bjsxt.service.impl"></context:component-scan>

    <bean id="mybefore" class="com.bjsxt.advice.MyBefore"></bean>
    <bean id="myafter" class="com.bjsxt.advice.MyAfter"></bean>
    <!-- 配置环绕通知bean -->
    <bean id="myaround" class="com.bjsxt.advice.MyAround"></bean>
    <aop:config>
        <aop:pointcut id="mypoint" expression="execution(*
com.bjsxt.service.impl.PeopleServiceImpl.test())"/>
        <!-- 只织入环绕，没有织入前置通知和后置通知。 -->
        <aop:advisor advice-ref="myaround" pointcut-ref="mypoint"></aop:advisor>
    </aop:config>
</beans>
```

**异常通知**

异常通知只有在切入点出现异常时才会被触发。如果方法没有异常，异常通知是不会执行的。

新建通知类

新建com.bjsxt.advice.MyThrow。

MethodInterceptor接口没有方法，但是我们必须严格提供一个下面的方法：

- public void afterThrowing:必须相同
- 必须有Exception参数

也就是说，虽然ThrowsAdvice虽然没有方法，但是我们必须自己写一个和下面一样的方法。

```
package com.bjsxt.advice;

import org.springframework.aop.ThrowsAdvice;

public class MyThrow implements ThrowsAdvice {
    public void afterThrowing(Exception e){
        System.out.println("异常通知："+e.getMessage());
    }
}
```

配置切面

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:component-scan base-
package="com.bjsxt.mapper,com.bjsxt.service.impl"></context:component-scan>

    <bean id="mybefore" class="com.bjsxt.advice.MyBefore"></bean>
    <bean id="myafter" class="com.bjsxt.advice.MyAfter"></bean>
    <bean id="myaround" class="com.bjsxt.advice.MyAround"></bean>
    <!-- 异常通知 -->
    <bean id="mythrow" class="com.bjsxt.advice.MyThrow"></bean>
    <aop:config>
        <aop:pointcut id="mypoint" expression="execution(*
com.bjsxt.service.impl.PeopleServiceImpl.test())"/>
        <!-- 织入异常通知 -->
        <aop:advisor advice-ref="mythrow" pointcut-ref="mypoint"></aop:advisor>
    </aop:config>
</beans>
```

在切入点中写个异常

在PeopleServiceImpl里面随意写个会出现异常的代码

```
@Service("peopleService")
public class PeopleServiceImpl implements PeopleService {
    @Autowired
    private PeopleMapperImpl peopleMapperImpl123;

    @Override
    public void test() {
        System.out.println("join pointer 切入点。");
        int i = 5/0;// 算数异常
    }
}
```

下面继续看一下AspectJ实现AOP的方式

Aspectj方式实现AOP的通知类不需要实现任何的接口，直接声明一个普通java类即可，然后在类中直接定义通知方法即可，方法名随意，但是建议方法名见名知意。

```
package com.bjsxt.advice;

import org.aspectj.lang.ProceedingJoinPoint;

public class MyAdvice {
    //前置通知方法
    public void before(){
        System.out.println("我是前置通知方法...");
    }
    //后置通知方法
    public void after(){
        System.out.println("我是后置通知方法...");
    }
    //环绕通知方法
    public Object round(ProceedingJoinPoint pp) throws Throwable {
        System.out.println("环绕---前");
        //放行
        Object proceed = pp.proceed();

        System.out.println("环绕---后");
        return proceed;
    }
    //异常通知方法
    public  void  myThrow(Exception ex){
        System.out.println("我是异常通知......"+ex.getMessage());
    }
}
```

配置AOP，所有的配置都在`<aop:aspect>`标签中配置，不同的通知类型使用不同的子标签进行配置。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/aop
```

```
                            http://www.springframework.org/schema/aop/spring-
aop-4.0.xsd

        ">
    <!--配置真实对象的bean-->
    <bean id="us" class="com.bjsxt.service.impl.UserServiceImpl"></bean>
    <!--配置通知bean-->
    <bean id="advice" class="com.bjsxt.advice.MyAdvice"></bean>
    <!--配置AOP组装-->
    <aop:config>
        <!--基于Aspectj方式配置-->
        <aop:aspect ref="advice">
            <!--声明切点-->
            <aop:pointcut id="mp" expression="execution(* com.bjsxt.*.impl.*.*
(..))"/>
            <!--配置通知方法-->
            <aop:before method="before"  pointcut-ref="mp"></aop:before>
            <!--切点正常执行才会被执行-->
            <aop:after-returning method="after" pointcut-ref="mp"></aop:after-
returning>
            <!--切点是否正常执行都会执行-->
            <!--<aop:after method="after" pointcut-ref="mp"></aop:after>-->
            <aop:around method="round" pointcut-ref="mp"></aop:around>
            <!-- 必须使用throwing声明异常参数名 -->
            <aop:after-throwing method="myThrow" pointcut-ref="mp"
throwing="ex">                    </aop:after-throwing>
        </aop:aspect>
    </aop:config>
</beans>
```

**注意:**

after和after-returning,after无论切点是否出现异常都执行的后置通知，after-returning只有在切点正常执行完成才会触发的通知。

最后领小伙伴们看看，Schema-based和Aspectj的区别

Schema-based:基于模式的。基于接口实现的。每个通知都需要实现特定的接口类型，才能确定通知的类型。由于类已经实现了接口，所以配置起来相对比较简单。尤其是不需要在配置中指定参数和返回值类型。

AspectJ方式:是基于配置实现的。通过不同的配置标签告诉Spring通知的类型。AspectJ方式对于通知类写起来比较简单。但是在配置文件中参数和返回值需要特殊进行配置。

因为Schame-based是运行时增强，AspectJ是编译时增强。所以当切面比较少时，性能没有太多区别。但是当切面比较多时，最好选择AspectJ方式，因为AspectJ方式要快很多