链滴

# Spring 学习笔记 -AOP 入门

作者：pleaseok
原文链接：https://ld246.com/article/1544358938456
来源网站：链滴

# AOP的概述

# 什么是AOP

在软件业，AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式运行期动态代理实现程序功能的统一维护的一种技术。==AOP是OOP的延续==，是软件开发中的一热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。利用AOP可以对业务逻辑各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了发的效率。——————百度百科

# Spring底层的AOP实现原理

动态代理

- JDK动态代理：只能对实现了接口的类产生代理

- Cglib动态代理：对没有实现接口的类产生代理。生产子类对象

++动态代理相关详细请看《java代理模式的那些事》++

# Spring的AOP的开发

# Spring的AOP的简介

AOP思想最早由AOP联盟组织提出的，Spring是使用这种思想最好的框架。前期，Spring的AOP也自己的实现方式，但是非常繁琐。不过当时有一款非常好的AOP框架 - AspectJ，Spring便引入AspecJ作为自身的AOP。so,Spring有两套AOP开发方式：Spring传统方式、Spring基于AspectJ的方式。

# AOP开发中的术语

- Joinpoint:连接点，可以被拦截到的点(方法、类)。
- Pointcut:切入点，真正被拦截到的点(方法、类)。
- Advice:通知(或者叫增强)，对切入点进行权限校验等动作的方法称为通知(方法层面的增强)。
- Introduction:引介，类层面的增强
- Target:被增强的对象
- Weaving:织入，将通知(Advice)应用到目标(Target)过程。
- proxy:代理对象
- Aspect:切面，多个通知和多个切入点组合

```
Class UserDao{
    public void save();
    public void find();
    public void update();
    public void delete();
}
/**
*    save、find、update、delete方法都可以被称为连接点
```

```
*    如果在实际开发中，只对save方法进行增强，则save被称为切入点
*    如果现在要对save方法进行权限校验(checkPri();)，则权限校验(checkPri();)的方法则称为通知
*    Target是指被增强的对象，也就是UserDao
*/
```

# 进入AspectJ的XML方式开发

## 创建web项目,引入jar包

1. 引入Spring的 6 个基本jar包(commons-logging、log4j、beans、context、core、expression)
2. 引入AOP开发的相关jar包(aop、aspects)
3. 引入aop的XML约束

```
<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.xsd">
</beans>
```

## 编写目标类并完成配置

```
public class ProductDaoImpl implements ProductDao{
    public void save(){
        //TODO
    }
    public void update(){
        //TODO
    }
    public void find(){
        //TODO
    }
    public void delete(){
        //TODO
    }
}

<!-- 配置目标对象:被增强的对象 -->
<bean id="productDao" class="com.spring.demo.ProductDaoImpl" />
```

## 编写测试类

```
@RunWith(SpringJunit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class SpringDemo{
    @Resource(name="productDao")
    private ProductDao productDao;

    @Test
```

```
    public void demo1(){
        productDao.save();
        productDao.update();
        productDao.find();
        productDao.delete();
    }
}
```

## 编写切面类

```
//切面类
public class MyAspectXML{

    public void checkPri(){
        System.out.println("权限校验。。。。。。。。");
    }
}

<!-- 将切面类交给Spring管理 -->
<bean id="myAspect" class="com.spring.demo.MyAspectXML" />
```

## 通过AOP的配置来实现目标代理

```
<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.xsd">
    <!-- 配置目标对象:被增强的对象 -->
    <bean id="productDao" class="com.spring.demo.ProductDaoImpl" />
    <!-- 将切面类交给Spring管理 -->
    <bean id="myAspect" class="com.spring.demo.MyAspectXML" />

    <!-- 通过AOP的配置完成对目标类产生代理 -->
    <aop:config>
        <!-- experssion:通过表达式配置哪些类的哪些方法需要进行增强 -->
        <aop:pointcut experssion="execution(* com.spring.demo.ProductDaoImpl.save(..))" id="
ointcut1"/>

        <!-- 配置切面 -->
        <aop:aspect ref="myAspect">
            <aop:before method="checkPri" pointcut-ref="pointcut1"/>
        </aop:aspect>
    </aop:config>
</beans>
```

# Spring中的通知类型

前置通知：在目标方法执行之前进行操作

获得切入点信息

```
<aop:aspect ref="myAspect">
        <aop:before method="checkPri" pointcut-ref="pointcut1"/>
</aop:aspect>
```

后置通知：在目标方法执行之后进行操作

获得方法的返回值

```
<aop:aspect ref="myAspect">
        <aop:after-returning method="checkPri" pointcut-ref="pointcut1" returning="result"
>
</aop:aspect>
```

环绕通知：在目标方法执行前后进行操作

环绕通知可以阻止目标方法的执行（ProceedingJoinPoint.proceed() 执行切入点）

```
<aop:aspect ref="myAspect">
        <aop:around method="checkPri" pointcut-ref="pointcut1"/>
</aop:aspect>
```

异常抛出通知：在程序出现异常的时候，进行的操作

```
<aop:aspect ref="myAspect">
        <aop:after-throwing method="checkPri" pointcut-ref="pointcut1" throwing="ex"/>
```

最终通知：无论代码时候有异常，总是会执行。（相当于try catch中的finally）

```
<aop:aspect ref="myAspect">
        <aop:after method="checkPri" pointcut-ref="pointcut1"/>
```

# Spring的切入点表达式的写法

==基于execution的函数完成的==

基本语法:

- [访问修饰符] 方法返回值 包名.类名.方法名(参数)
- public void com.spring.demo.UserDao.save(..)
- public可省略。返回值可以是具体的String void,也可以是 *表示任意返回值。..表示任意参数