# 1、AOP

## 1.1 AOP概述

1. 在软件业，**AOP为Aspect Oriented Programming**的缩写，意为：**面向切面编程**，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。
2. AOP是OOP（面向对象编程）的延续，是软件开发中的一个热点，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。
3. 利用AOP可以对业务逻辑的各个部分进行隔离，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，同时提高了开发的效率。
4. AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码
5. **经典应用：事务管理、性能监视、安全检查、缓存 、日志等【画图】**
6. Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码
7. **AspectJ是一个基于Java语言的AOP框架**，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入

## 1.2 AOP术语

**1.target：**目标类，需要被代理的类。例如：UserService

**2.Joinpoint**(连接点):所谓连接点是指那些可能被拦截到的方法。例如：所有的方法

**3.PointCut** 切入点：已经被增强的连接点。例如：addUser()

4.**advice 通知/增强**，**增强代码**。例如：after、before

**5. Weaving**(织入):是指把增强advice应用到目标对象target来创建新的代理对象proxy的过程.

**6.proxy** 代理类

**7. Aspect**(切面): 是切入点pointcut和通知advice的结合

一个线是一个特殊的面。

一个切入点和一个通知，组成成一个特殊的面。

|  |
| --- |
|  |

## 1.3 AOP联盟通知类型

|  |
| --- |
| AOP联盟为通知Advice定义了org.aopalliance.aop.Advice  Spring按照通知Advice在目标类方法的连接点位置，可以分为5类  •**前置通知** org.springframework.aop.MethodBeforeAdvice  •在目标方法执行前实施增强  •**后置通知** org.springframework.aop.AfterReturningAdvice  •在目标方法执行后实施增强  •**环绕通知** org.aopalliance.intercept.MethodInterceptor  •在目标方法执行前后实施增强  •**异常抛出通知** org.springframework.aop.ThrowsAdvice  •在方法抛出异常后实施增强  •**引介通知** org.springframework.aop.IntroductionInterceptor  在目标类中添加一些新的方法和属性 |

|  |
| --- |
| **环绕通知**，**必须手动执行目标方法**  try{  //前置通知  //执行目标方法  //后置通知  } catch(){  //抛出异常通知  } |

# AspectJ

## 2.1 AspectJ简介

* AspectJ是一个基于Java语言的AOP框架
* Spring2.0以后新增了对AspectJ切点表达式支持
* @AspectJ 是AspectJ1.5新增功能，通过JDK5注解技术，允许直接在Bean类中定义切面
* **新版本Spring框架， AspectJ方式来开发AOP**
* **主要用途：自定义开发**

## 2.2 切入点表达式【掌握】

### execution()

用于描述方法 【掌握】

|  |
| --- |
| 语法：execution(修饰符 返回值 包.类.方法名(参数) throws异常) |

**修饰符，一般省略**

public 公共方法

\* 任意

**返回值，不能省略**

void 返回没有值

String 返回值字符串

\* 任意

**包，[省略]**

com.neuedu.aop.demo 固定包

com.neuedu.ws.\*.service ws包下面子包任意 （例如：com.neuedu.ws.staff.service）

com.neuedu.ws.. ws包下面的所有子包（含自己）

com.neuedu.ws.\*.service.. ws包下面任意子包，固定目录service，service目录任意包

**类，[省略]**

UserServiceImpl 指定类

\*Impl 以Impl结尾

User\* 以User开头

\* 任意

**方法名，不能省略**

addUser 固定方法

add\* 以add开头

\*Do 以Do结尾

\* 任意

**(参数)**

() 无参

(int) 一个整型

(int ,int) 两个

(..) 参数任意

**throws ,可省略，一般不写**。

**案例1：**

**execution(\* com.neuedu.ws.\*.service..\*.\*(..))//** ws包下面任意子包，固定目录service，service目录任意包所有类所有方法，参数个数为任意

**案例2：或**

**<aop:pointcut expression="execution(\* com.neuedu.ws.service.\*.\*(..))**

**id="myPointCut"/>**

## 2.3 AspectJ 通知类型

|  |
| --- |
| aop联盟定义通知类型，具有特性接口，必须实现，从而确定方法名称。  aspectj 通知类型，只定义类型名称，以及方法格式。  个数：6种，**知道5种，掌握1**中。  before:前置通知(应用：各种校验)  在方法执行前执行，如果通知抛出异常，阻止方法运行  afterReturning:后置通知(应用：常规数据处理)  方法正常返回后执行，如果方法中抛出异常，通知无法执行  必须在方法执行后才执行，所以可以获得方法的返回值。  around:环绕通知(应用：十分强大，可以做任何事情)  方法执行前后分别执行，可以阻止方法的执行  **必须手动执行目标方法**  afterThrowing:抛出异常通知(应用：包装异常信息)  方法抛出异常后执行，如果方法没有抛出异常，无法执行  after:最终通知(应用：清理现场)  方法执行完毕后执行，无论方法中是否出现异常 |

## 2.4 Aspect案例讲解【其于xml】

### 第一步：导包

|  |  |
| --- | --- |
|  | AOP联盟规范 |
| Spring-aop-5.0.5.jar | AOP实现 |
| Aspectjweaver-1.8.13.jar | 规范 |
| spring-aspects-5.0.5 | aspectj实现 |

|  |
| --- |
|  |

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>5.0.5.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>5.0.5.RELEASE</version>  </dependency>  注解:如何出现无法下载aspectjweaver，可以手动下载配置:  mvn install:install-file -DgroupId=org.aspectj -DartifactId=aspectjweaver -Dversion=1.8.13 -Dpackaging=jar -Dfile=aspectjweaver-1.8.13.jar |

### 第二步：实现类和切面类

|  |
| --- |
| **public** **class** UserServiceImpl **implements** UserService {  @Override  **public** **void** addUser() {  System.***out***.println("添加用户");  }  @Override  **public** **void** updateUser() {  System.***out***.println("更新用户");  }  @Override  **public** **void** deleteUser() {  System.***out***.println("删除用户");  }    **public** String testByUserId() {  **return** "120";  }  } |
| **package** cn.neuedu.aop.demo;  **import** org.aspectj.lang.JoinPoint;  **import** org.aspectj.lang.ProceedingJoinPoint;  **public** **class** MyAspect {  **public** **void** myBefore(JoinPoint joinPoint) {  System.***out***.println(joinPoint.getSignature().getName());  System.***out***.println("前置通知");  }    **public** **void** myAfterReturning(JoinPoint joinPoint,Object ret) {  //第二参数为返回值  System.***out***.println(joinPoint.getSignature().getName());  System.***out***.println("后置通知:"+ret);  }    **public** Object myAround(ProceedingJoinPoint joinPoint) **throws** Throwable {  System.***out***.println("环绕通知前");  Object obj = joinPoint.proceed();  System.***out***.println("环绕通知后");  **return** obj;  }  **public** **void** myAfterThrowing(JoinPoint joinPoint,Throwable e) {  System.***out***.println("异常通知:"+e.getMessage());  }  **public** **void** myAfter() {  System.***out***.println("最终通知");  }  } |

### 第三步：spring的xml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"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *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"*>  <bean id=*"myAspect"* class=*"cn.neuedu.aop.demo.MyAspect"*></bean>  <bean id=*"userService"* class=*"cn.neuedu.aop.demo.UserServiceImpl"*></bean>  <aop:config>  <aop:aspect ref=*"myAspect"*>  <aop:pointcut expression=*"execution(\* cn.neuedu..\*.\*(..))"* id=*"myPointCut"*/>  <aop:around method=*"myAround"* pointcut-ref=*"myPointCut"*/>  <aop:after method=*"myAfter"* pointcut-ref=*"myPointCut"*/>  </aop:aspect>  </aop:config>  </beans> |

### 第四步：测试

|  |
| --- |
| @Test  **public** **void** testAdd() {  ClassPathXmlApplicationContext application =  **new** ClassPathXmlApplicationContext("applicationContext.xml");  UserService userService=(UserService)application.getBean(UserService.**class**);  userService.addUser();  } |

## 2.5 Aspect案例讲解【其于xml】

### 第1步：声明使用注解

|  |
| --- |
| <?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"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *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"*>  <context:component-scan base-package=*"cn.neuedu.aop"*/>  <aop:aspectj-autoproxy/>  </beans> |

### 第2步：替换service和 切面 bean

|  |
| --- |
|  |
|  |
|  |

### 第3步：声明切面

|  |
| --- |
|  |
| @Component  @Aspect  **public** **class** MyAspect {  @Before("execution(\* cn.neuedu..\*.\*(..))")  **public** **void** myBefore(JoinPoint joinPoint) { |

### 第4步：声明前置通知

|  |
| --- |
| @Before("execution(\* cn.neuedu..\*.\*(..))")  **public** **void** myBefore(JoinPoint joinPoint) {  System.***out***.println(joinPoint.getSignature().getName());  System.***out***.println("前置通知");  } |

### 第5步：声明公共切入点

|  |
| --- |
| @Pointcut("execution(\* cn.neuedu..\*.\*(..))")  public void myPointCut() {  } |

### 第6步：声明后置通知

|  |
| --- |
| @AfterReturning(value="myPointCut()",returning="ret")  **public** **void** myAfterReturning(JoinPoint joinPoint,Object ret) {  //第二参数为返回值 |

### 第7步：声明环绕通知

|  |
| --- |
| @Around(value="myPointCut()")  **public** Object myAround(ProceedingJoinPoint joinPoint) **throws** Throwable |

### 第8步：声明异常通知

|  |
| --- |
|  |

### 第9步：声明最终通知

|  |
| --- |
|  |

### 注解总结

@Aspect 声明切面，修饰切面类，从而获得 通知。

通知

@Before 前置

@AfterReturning 后置

@Around 环绕

@AfterThrowing 抛出异常

@After 最终

切入点

@PointCut ，修饰方法 private void xxx(){} 之后通过“方法名”获得切入点引用