1. AOP
2. 概述

Aspect Oriented Programming,面向切面编程，是通过预编译方式和运行期间动态代理实现程序功能的统一维护的一种技术。动态代理的优点是不改变源码，进而增强代码的功能，它具有松耦合的优点。因此AOP可以对业务逻辑的各个部分进行松耦合，提高代码的可重用性。

1. AOP的作用

AOP通过配置文件的方式，使得两个功能在运行时能够结合到一起。

1. AOP的底层实现

AOP的底层是通过Spring提供的动态代理技术来实现的。在运行期间，Spring通过动态代理技术动态地生成代理对象，代理对象方法执行时进行增强功能，再去调用目标对象的方法，从而完成功能的增强。

1. AOP的动态代理技术
2. JDK代理

JDK代理技术是基于接口的动态代理技术，它要求原始的目标对象必须有接口，JDK代理才能基于该接口对目标对象进行增强动态生成代理对象。

1. cglib代理

cglib代理是基于父类的动态代理技术，它没有要求接口，只是在运行时动态地生成一个代理对象，增强原始目标对象的功能。

1. AOP术语
2. Target（目标对象）

动态代理要代理的目标对象，即要进行增强的对象。

1. Proxy（代理）

一个类被AOP织入增强后，就产生了一个结果代理类

1. Joinpoint（连接点）

连接点是指那些被拦截到的点，在Spring中，这些点指的是方法，因为Spring只支持方法类型的连接点。即有哪些方法是可以被增强的即为连接点。

1. Pointcut（切入点）

所谓的切入点是指我们要对哪些连接点进行拦截，即我们实际要对哪些方法进行增强，这些方法就叫做切入点。

1. Advice（通知/增强）

所谓的通知指的是拦截到连接点（方法）后要做的事情，即增强该方法。

1. Aspect(切面)

切入点和通知的结合

1. Weaving（织入）

织入是指将增强应用到目标对象上来创建新的代理对象的过程。Spring采用动态代理织入，而AspectJ采用编译期织入和类装载期织入。

1. AOP开发的内容
2. AOP需要编写的内容
3. 编写核心业务代码，即目标类中的目标方法。
4. 编写切面类，切面类中有通知（增强的功能方法）
5. 在配置文件中，配置织入关系，即将通知和切入点结合在一起。
6. AOP技术实现的内容

Spring框架监控切入点方法的执行，一旦监控到了切入点方法被执行，则使用代理机制，动态地创建目标对象的代理对象，根据通知类别，在代理对象的对应位置，将通知对应的功能织入，完成完整的代码逻辑运行。

1. AOP底层代理方式

在Spring框架中，它会根据目标类是否实现了接口来决定采用哪种动态代理的方式。若有接口，则使用JDK；若没有接口，则使用cglib代理。

1. 基于XML的AOP开发
2. 开发步骤
3. 导入AOP相关坐标

spring-context是依赖于spring-aop的，因此我们只需要导入spring-context即可，同时，我们还要导入aspectjweaver，这是一个轻量级的aop框架，一般使用它来配置aop。

1. 创建目标接口和目标类（内部有切点）
2. 创建切面类（内部有增强方法）
3. 将目标类和切面类的对象创建权交给spring容器管理，即在applicationContext配置文件中将目标类和切面类实例化Bean对象，放入Spring容器中

<!-- 目标对象-->  
 <bean id="target" class="com.aop.com.aop.Target"></bean>  
<!-- 切面对象-->  
 <bean id="myAspect" class="com.aop.com.aop.MyAspect"></bean>

1. 在applicationContext.xml中配置织入关系，即告诉Spring容器哪些方法需要进行哪些增强。
2. 配置的第一步就是在XML中引入aop的命名空间，复制主命名空间xmlns，将beans改为aop，并在xmlns后面添加”:aop”；同时复制最下面的schemeLocation，将beans改为aop即可。
3. 引入aop命名空间后就可以配置织入，告诉Spring框架哪些切点需要进行哪些通知（前置，后置…）。

首先，使用<aop:config>标签，然后再创建子标签<aop:aspect>声明切面，ref属性导入切面类的id名称；切面中有切点和通知，因此在切面标签内再创建通知子标签<aop:before>，这是前置增强，method属性为切面类中的通知方法名称，pointcut属性为切点表达式，意思为执行该切点，public void为该切点的签名，后面的为目标类的全类名，后面跟着切点方法，还要加上括号。

<aop:config>

<aop:aspect ref=”myAspect”>

<aop:before method=”before” pointcut=”execution(public void com.aop.Target.save())” />

</aop:aspect>

</aop:config>

1. 测试AOP

导入junit，spring-test的坐标，利用注解依赖注入的方式，创建测试类测试aop的情况，如下所示，注入目标类，调用目标类的切点，得出了通知增强后的代理对象结果。

@RunWith(SpringJUnit4ClassRunner.class)  
@ContextConfiguration("classpath:applicationContext.xml")  
public class AopTest {  
  
 @Autowired  
 private TargetInterface target;  
  
 @Test  
 public void test1(){  
 target.save();  
 }  
}

1. 切点表达式的写法

切点表达式为applicationContext.xml配置文件中通知标签的切点pointcut属性的值。该表达式的意思是有哪些切点会被增强。

1. 表达式的语法

execution([修饰符] 返回值类型 包名.类名.方法名(参数))

1. 表达式写法
2. 访问修饰符可以省略
3. 返回值类型，包名，类名，方法名可以使用星号\*代表任意
4. 包名与类名之间一个点 . 代表当前包下的类，两个点 .. 表示当前包及子包下的类
5. 参数列表可以使用两个点 .. 表示任意个数，任意类型的参数列表
6. 实例解析

execution(void com.aop.Target.\*(..))

代表Target包下的所有方法都能被增强，参数为任意值。

execution(\* com.aop..\*.\*(..))

代表aop的所有子包的所有方法都能被增强，参数为任意值。

1. 通知类型
2. 通知的标签语法

<aop:通知类型 method=”切面类中方法名” pointcut=”切点表达式”></aop:通知类型>

1. 通知的类型
2. 前置通知

<aop:before>，指定增强的方法在切入点方法之前执行

1. 后置通知

<aop:after-returning>，指定增强的方法在切入点方法之后执行

1. 环绕通知

<aop:around>，指定增强的方法在切入点方法之前和之后都执行

环绕通知比较特殊，通知方法在中间会执行原有的切点的方法。要在通知中传入一个参数ProceedingJoinPoint，代表着正在执行的连接点，即切点，在通知中运行切点方法。

如，设置一个环绕通知方法：

public Object around(ProceedingJoinPoint pjp) throws Throwable{

System.out.println(“环绕前增强”);

Object proceed = pjp.proceed(); //切点方法

System.out.println(“环绕后增强”);

return proceed;

}

1. 异常抛出通知

<aop:throwing>，指定增强的方法在切点出现异常时执行

1. 最终通知

<aop:after>，无论增强方式是否有异常都会执行

1. 切点表达式的抽取

当多个增强的切点表达式相同时，可以将切点表达式进行抽取到一个<aop:pointcut>标签内，该标签在<aop:config>内。在通知标签中使用pointcut-ref属性代替pointcut属性，来引用抽取后的切点表达式，如：

<aop:config>

<aop:aspect ref=”myAspect”>

<aop:pointcut id=”myPointcut” expression=”execution(\* com.aop.\*.\*(..))”/>

<aop:before method=”before” pointcut-ref=”myPointcut”></aop:before>

</aop:aspect>

</aop:config>

1. 基于注解的AOP开发
2. 实现步骤
3. 创建目标接口和目标类（内部有切点）
4. 创建切面类（内部有通知）
5. 将目标类和切面类的对象创建权交给spring，即通过注解@Component将两个类都注入Spring容器中
6. 在切面类中使用注解配置织入关系

在切面类上用注释@Aspect标注这个类是切面类，同时要用对应的通知注解标注通知，如前置增强用@Before()，括号的参数为切点表达式。如：

@Before(“execution(\* com.aop.\*.\*(..))”)

public void before(){ System.out.println(“前置增强”);}

1. 在配置文件中开启组件扫描和AOP的自动代理。组件扫描用于扫描注解，在context命名空间中，首先要在applicationContext.xml配置文件中导入context命名空间，然后添加组件扫描：

<context:component-scan base-package=”com.aop”/>

然后添加aop的自动代理，用于识别注解中的aop注解：

<aop:aspect-autoproxy>

1. 注解通知的类型
2. 通知注解的语法

@通知注解(“切点表达式”)

1. 通知注解的类型
2. 前置通知

@Before

1. 后置通知

@AfterReturning

1. 环绕通知

@Around

1. 异常抛出通知

@AfterThrowing

1. 最终通知

@After

1. 注解中切点表达式的抽取

在注解中抽取切点表达式，可以在切面类中定义方法，方法体可以为空，在该方法上使用@Pointcut()注解定义切点表达式，括号内为切点表达式：

@Pointcut(“execution(\* com.aop.\*.\*(..))”)

public void myPoint(){}

然后在增强方法的注解中进行引用，可以直接用方法名()引用，也可以用切面类名.方法名()进行引用：

@Before(“MyAspect.myPoint()”)

public void before(){ System.out.println(“前置增强”);}

@Before(“myPoint()”)

public void before(){ System.out.println(“前置增强”);}