慕课网Spring AOP视频：<https://www.imooc.com/learn/869>

AOP即面向切面编程，相比OOP-面向对象编程，由于面向对象中最基本的单位是类、实例，AOP中最基本的单位就是所谓的切面。

面向切面——Spring提供了面向切面编程的丰富支持，允许通过分离应用的业

务逻辑与系统级服务（例如审计（auditing）和事务（transaction）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们并不负责（甚至是意识）其它的系统级关注点，例如日志或事务支持。

一、AOP中的基本概念

#### 1、通知Advice

* Before
* After
* After-returning 在方法成功执行之后调用通知
* After-throwing 在方法抛出异常后调用通知
* Around

可以理解通知就是对应日常生活中所说的通知，比如“某某人，你2019年9月1号来学校报个到”，通知更多地体现一种告诉我们（告诉系统何）何时执行，规定一个时间，在系统运行中的某个时间点（比如抛异常啦！方法执行前啦！），并非对应代码中的方法！

#### 2、切点（Pointcut）

切点在Spring AOP中对应系统中的方法，但是这个方法是定义在切面中的方法，一般和通知一起使用，一起组成了切面。

#### 3、连接点（Join point）

比如：方法调用、方法执行、字段设置/获取、异常处理执行、类初始化、甚至是for循环中的某个点。理论上程序执行过程中的任何时点都可以作为织入点。但Spring AOP目前仅支持方法执行（method excution）也可以这样理解，连接点就是你准备在系统中执行切点和切入通知的地方（一般是一个方法，一个字段）

#### 4、切面（Aspect）

切面是切点和通知的集合，一般单独作为一个类。通知和切点共同动员了关于切面的全部内容，它是什么时候，在何时何处完成功能。

#### 5、引用（Introduction）

引用允许我们向现有的类添加新的方法或者属性。

#### 6、织入（Weaving）

组装方面来创建一个被通知对象。这可以在编译时完成（例如使用AspectJ编译器），也可以在运行时完成。Spring和其他纯Java AOP框架一样，在运行时完成织入。

二、Spring中对AOP的支持

首先AOP思想的实现一般都是基于代理模式，Spring AOP 会这样子来进行切换，因为Spring AOP 同时支持 CGLIB、ASPECTJ、JDK动态代理，当你的真实对象有实现接口时，Spring AOP会默认采用JDK动态代理，否则采用cglib代理。

三、代码实现

#### 1、定义主题接口

这些接口可以成为我们的连接点Join point

public interface Subject {

void login();  
void download();  
}

#### 2、定义实现类

代理模式中真正的被代理对象

public clsass SubjectImpl implements Subject {

@Override  
 public void login() {  
 System.*out*.println("登录中...");  
 }  
  
 @Override  
 public void download() {  
 System.*out*.println("下载中...");  
 }  
}

#### 3、定义切面

切面中有切点和通知

public class PermissionVerification {

*/\*\*  
 \* 权限校验  
 \*/* public void canLogin() {  
 System.*out*.println("正在校验...");  
 }  
  
 */\*\*  
 \* 校验之后做一些处理（无论是否成功都做处理）  
 \*/* public void saveMessage() {  
 //做一些后置处理  
 System.*out*.println("正在处理...");  
 }  
}

SpringAOP.xml文件

<bean id="SubjectImpl1" class="com.chance.spring.aop.SubjectImpl"/>

<bean id="SubjectImpl2" class="com.chance.spring.aop.SubjectImpl"/>  
<bean id="PermissionVerification" class="com.chance.spring.aop.aspectj.PermissionVerification"/>  
  
<aop:config>  
 <!--定义一个切面，切面是切点和通知的集合-->  
 <aop:aspect id="cut" ref="PermissionVerification">  
 <!--定义切点，expression语言，表示包括该接口中定义的所有方法都会被执行-->  
 <aop:pointcut id="point" expression="execution(\* com.chance.spring.aop.Subject.\*(..))"/>  
 <!--定义通知-->  
 <aop:before method="canLogin" pointcut-ref="point"/>  
 <aop:after method="saveMessage" pointcut-ref="point"/>  
 </aop:aspect>  
</aop:config>

<dependency>

<groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
  
<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.8.9</version>  
</dependency>

#### 4、测试

ApplicationContext ctx = new ClassPathXmlApplicationContext("SpringAOP.xml");

Subject subject1 = (Subject) ctx.getBean("SubjectImpl1");  
  
subject1.login();  
subject1.download();

可以看下JDK动态代理的源码分析（里边非常地巧妙，还运用了二级缓存）、Spring AOP源码分析等等，建议采用基于注解的方式进行AOP编程！

四、基于AspectJ注解实现AOP

#### 1、@AspectJ的由来

AspectJ是一个AOP框架，它能够对java代码进行AOP编译，让java代码具有AspectJ的AOP功能（当然需要特殊的编译器），可以这样说AspectJ是目前实现AOP框架中最成熟，功能最丰富的语言，更幸运的是，AspectJ与java程序完全兼容。

其实AspectJ单独就是一门语言，它需要专门的编译器(ajc编译器)。

Spring AOP 与AspectJ的目的一致，都是为了统一处理横切业务，但与AspectJ不同的是，Spring AOP并不尝试提供完整的AOP功能(即使它完全可以实现)，Spring AOP更注重的是与Spring IOC容器的结合，并结合该优势来解决横切业务的问题，因此在AOP的功能完善方面，相对来说AspectJ具有更大的优势，同时,Spring注意到AspectJ在AOP的实现方式上依赖于特殊编译器(ajc编译器)，因此Spring很机智回避了这点，转向采用动态代理技术的实现原理来构建Spring AOP的内部机制（动态织入），这是与AspectJ（静态织入）最根本的区别。在AspectJ 1.5后，引入@Aspect形式的注解风格的开发，Spring也非常快地跟进了这种方式，因此Spring 2.0后便使用了与AspectJ一样的注解。请注意，Spring只是使用了与AspectJ 5一样的注解，但仍然没有使用AspectJ的编译器，底层依是动态代理技术的实现，因此并不依赖于AspectJ的编译器。

所以，Spring AOP虽然是使用了那一套注解，其实实现AOP的底层是使用了动态代理(JDK或者CGLib)来动态植入。

#### 2、Spring AOP-AspectJ注解

* 切点：定位到具体方法的一个表达式
* 切面：切点+建言
* 建言（增强）：定位到方法后干什么事

#### 3、代码实现

<dependency>

<groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
  
<!--下面这两个aspectj的依赖是为了引入AspectJ的注解-->  
<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjrt</artifactId>  
 <version>1.9.2</version>  
</dependency>  
<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.2</version>  
</dependency>  
  
<!--Spring AOP底层会使用CGLib做动态代理-->  
<dependency>  
 <groupId>cglib</groupId>  
 <artifactId>cglib</artifactId>  
 <version>3.3.0</version>  
</dependency>

被代理对象：

public class Dog {

public void say() {  
 System.*out*.println("小狗在汪汪叫...");  
 }  
}

切面类：

@Aspect

public class MyAspect {  
   
 @Before("execution(\* com.chance.spring.aop.Dog.\*(..))")  
 public void before() {  
 System.*out*.println("前置通知");  
 }  
}

@Aspect声明自己是切面类，然后@Before(方位)，切入点可以定位到具体某个类的某个方法的方位。

配置文件：

<!--开启AspectJ功能-->

<aop:aspectj-autoproxy/>  
<bean id="dog" class="com.chance.spring.aop.Dog"/>  
<!--定义aspect类-->  
<bean name="myAspect" class="com.chance.spring.aop.annotation.MyAspect"/>

说白了，就是把切面类丢到容器，开启一个AspectJ的功能，Spring AOP就会根据切面类中的（@Before(切入点)）定位好具体的类的某个方法（这里定义的是com.chance.spring.aop包下类的所有方法），然后把增强before()切入进去。

Spring AOP支持的几种类似于@Before的AspectJ注解：

@Before()

@After()

@AfterReturning()

@AfterThrowing()

@Around()

然后说下一直用“...”忽略掉切入点表达式，这个表达式可以不是execution(..)，还有其他的一些，最常用的是execution：

//scope ：方法作用域，如public,private,protect

//returnt-type：方法返回值类型

//fully-qualified-class-name：方法所在类的完全限定名称

//parameters 方法参数

execution(<scope> <return-type> <fully-qualified-class-name>.\*(parameters))