# AOP概念

AOP（Aspect Orient Programming），也就是面向切面编程。可以这样理解，面向对象编程（OOP）是从静态角度考虑程序结构，面向切面编程（AOP）是从动态角度考虑程序运行过程。

切面 - 通知 - 代理

“目标类”被“切面类”“通知”

“代理类” = “目标类” + “增强处理”

1. 切面(aspect)：用来切插业务方法的类。（切面类中，包含了所有的通知）
2. 连接点(joinpoint)：是切面类和业务类的连接点，其实就是封装了业务方法的一些基本属性，作为通知的参数来解析。（将业务类方法的参数传递给代理类，由切面类来调用业务类方法？）
3. 切入点(pointcut)：业务类中指定的方法，作为切面切入的点。
4. 通知(advice)：在切面类中，声明对业务方法做额外处理的方法。
5. 前置通知(before advice)：在切入点之前执行。
6. 后置通知(after returning advice)：在切入点执行完成后，执行通知。
7. 环绕通知(around advice)：包围切入点，调用方法前后完成自定义行为。
8. 异常通知(after throwing advice)：在切入点抛出异常后，执行通知。
9. 目标对象(target object)：被代理对象。被一个或者多个切面所通知的对象。（业务类）
10. AOP代理(aop proxy)：代理对象。在Spring AOP中有两种代理方式，JDK动态代理和CGLIB代理。默认情况下，TargetObject实现了接口时，则采用JDK动态代理。一旦定义了合适的切入点和增强处理，AOP框架将会自动生成 AOP代理，即：代理对象的方法 = 增强处理 + 被代理对象的方法。
11. 切入点表达式 ：通常情况下，表达式中使用”execution“就可以满足大部分的要求。

Spring3常见异常：

http://blog.csdn.net/sjzs5590/article/details/7348836

# Spring对AOP的支持

**需要引入的包：**

**aspectjweaver.jar**

**aopalliance.jar**

**asm.jar**

**cglib.jar**

**commons-logging-1.1.1.jar**

**spring.\*.jar**

1. Spring中对AOP的支持：

代理类生成的方法有2种：

**JDK √**

CGLIB

1. Spring中AOP的实现

Spring有如下两种选择来定义切入点和增强处理：

基于 Annotation 的“零配置”方式：

√SpringAOPTest4（aspectj注解）“@\*\*\*”：\*\*\*注解

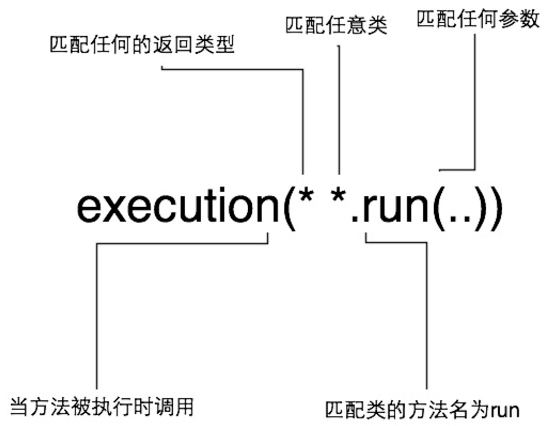
基于 XML 配置文件的管理方式：

配置切入点：

**schema配置 √**SpringAOPTest3

不配置切入点：

不配置切入点则需要在每一个通知中加入具体的切入点属性：<aop:after pointcut="execution(\* com.wicresoft.app.service.impl.\*.\*(..))" method="doAfter" />



通知写不写在同一个切面类中与使用什么配置方式不冲突

不同的通知定义在不同的切面类中：√SpringAOPTest2

Column注解：@Column注解来标识实体类中属性与数据表中字段的对应关系。

纯POJO类：POJO（Plain Ordinary Java Object）简单的Java对象，实际就是普通JavaBeans，是为了避免和EJB混淆所创造的简称。