AOP(Aspect-Oriented Programming), 即 **面向切面编程**, 它与 OOP( Object-Oriented Programming, 面向对象编程) 相辅相成, 提供了与 OOP 不同的抽象软件结构的视角.  
在 OOP 中, 我们以类(class)作为我们的基本单元, 而 AOP 中的基本单元是 **Aspect(切面)**

通知(Advice): 有五种通知，执行前，执行后，执行成功后，执行抛出异常后，环绕通知。就是切面执行的方法。  
目标(Target): 被通知的对象。  
连接点（Joinpoint）：连接点是一个应用执行过程中能够插入一个切面的点。  
切点（pointcut）：每个类都拥有多个连接点，即连接点是程序类中客观存在的事务。AOP 通过切点定位到特定的连接点

实现

基于XML的使用

<aop:advisor> 定义一个AOP通知者  
<aop:after> 后通知  
<aop:after-returning> 返回后通知  
<aop:after-throwing> 抛出后通知  
<aop:around> 周围通知  
<aop:aspect>定义一个切面  
<aop:before>前通知  
<aop:config>顶级配置元素，类似于<beans>这种东西  
<aop:pointcut>定义一个切点

JoinPoint

基于注解的使用

<aop:aspectj-autoproxy proxy-target-**class**="**true**"></**aop**:**aspectj**-**autoproxy**>

@**Aspect**

@**Before**("execution(\* com.zhangguo.Spring052.aop02.Math.\*(..))")

@**After**("execution(\* com.zhangguo.Spring052.aop02.Math.\*(..))")

@AfterReturning(value="execution(public \* \*(..))")

@AfterThrowing(value="execution(public \* \*(..))")

@Around(value="execution(public \* \*(..))")

基于动态代理

1. 静态代理
2. 动态代理
3. Cglib

静态代理：由程序员创建或特定工具自动生成源代码，再对其编译。在程序运行前，代理类的.class文件就已经存在了。   
动态代理：在程序运行时，运用反射机制动态创建而成。

动态代理也分为两类：基于接口的代理和基于继承的代理   
两类实现的代表是：JDK代理 与 CGlib代理

JDK动态代理主要涉及java.lang.reflect包下的两个类：Proxy类和InvocationHandler接口。

JDK代理实现的三个要点：

通过java.lang.reflect.Proxy类来动态生成代理类

代理类要实现InvocationHandler接口

JDK代理只能基于接口进行动态代理的

CGLib采用非常底层的字节码技术，可以为一个类创建子类，并在子类中采用方法拦截的技术拦截所有的父类方法的调用，并顺势织入横切逻辑。   
CGlib和JDK的原理类似，也是通过方法去反射调用目标对象的方法。

Enhancer 通过该对象设置代理对象和目标对象。

MethodInterceptor代理对象实现该接口。