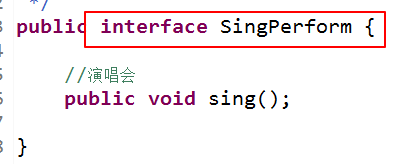
# Day13\_Cglib代理+AOP编程

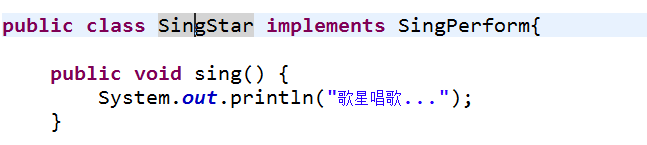
## 代理模式

### 静态代理

#### 定义一个接口:



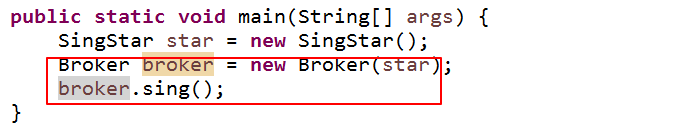
#### 定义目标类:



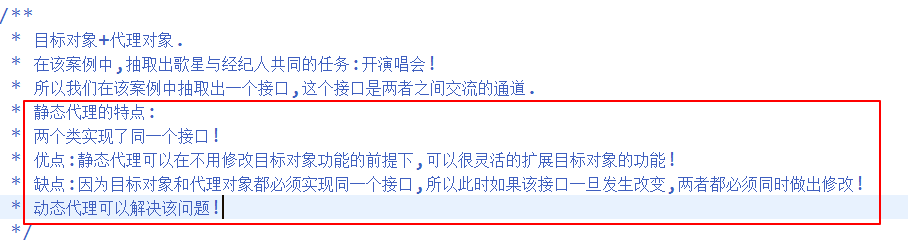
#### 定义代理类:



#### 测试:

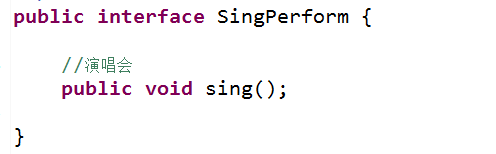


#### 静态代理的总结:

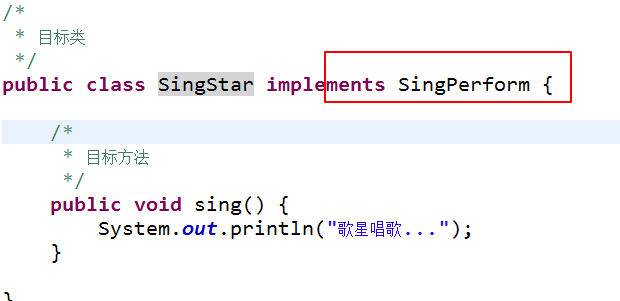


### 动态代理

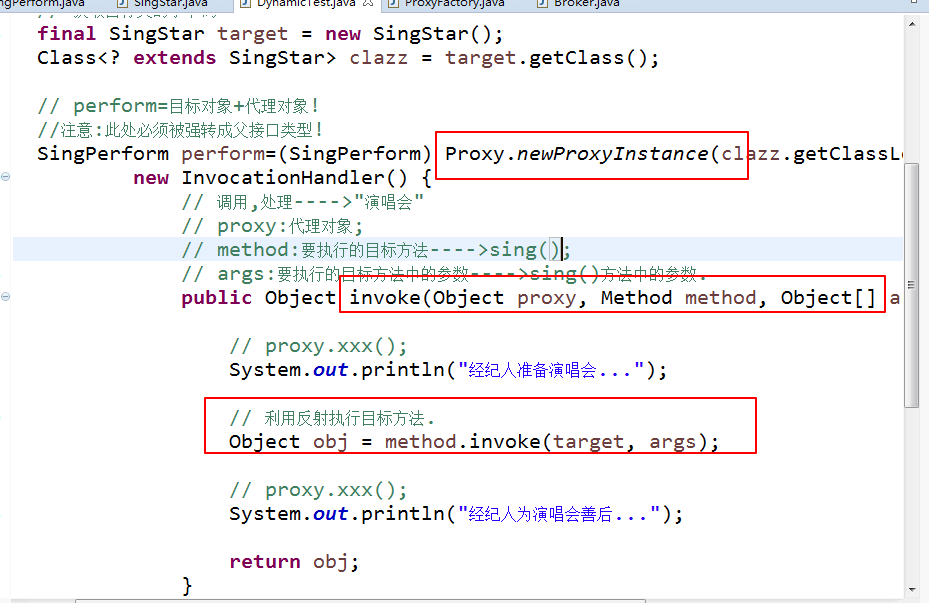
#### 定义目标类的接口:



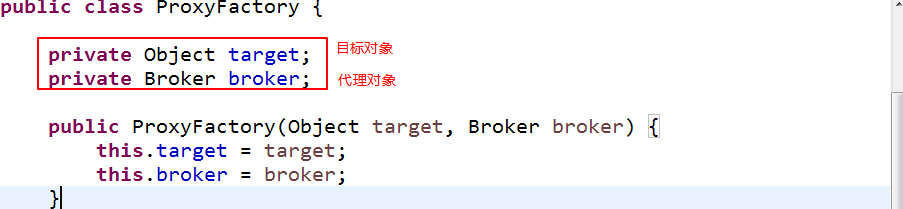
#### 目标类实现该接口:

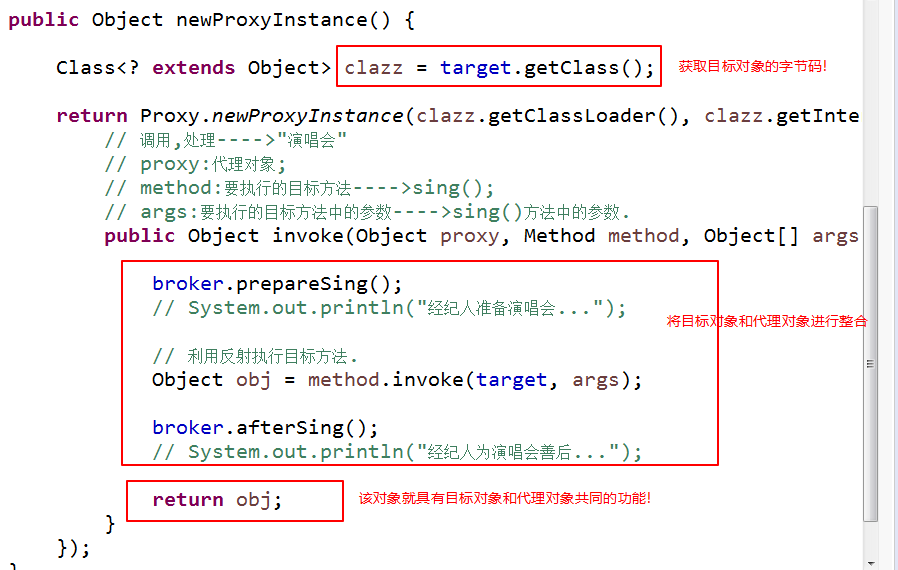


#### 不创建代理类:

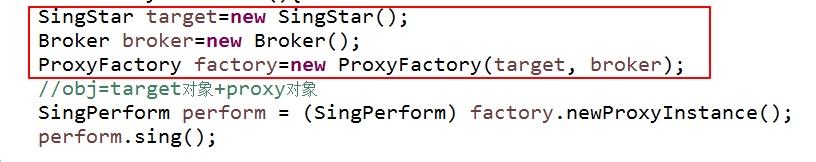


#### 创建代理类:

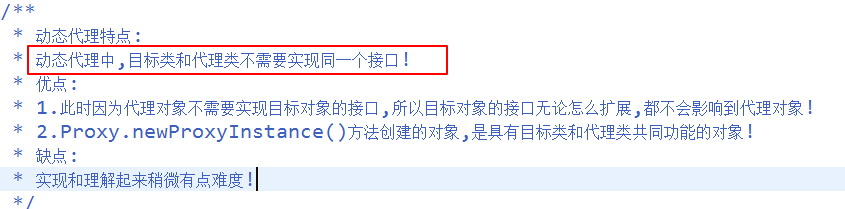




#### 测试:



#### 动态代理总结:

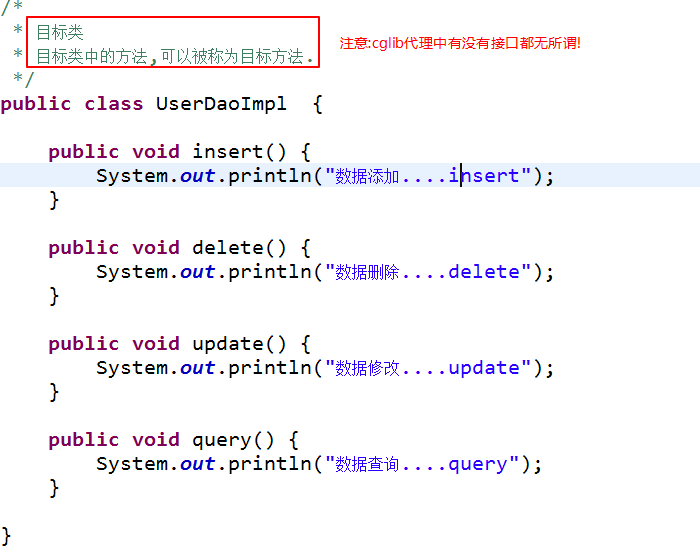


### Cglib(子)代理

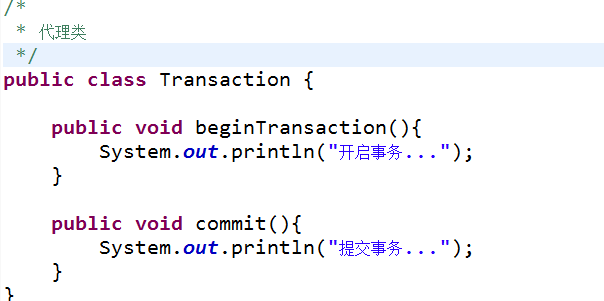
#### 1.添加依赖包:



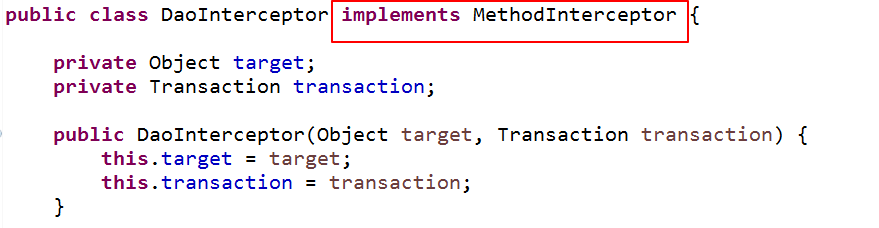
#### 2.定义一个目标类

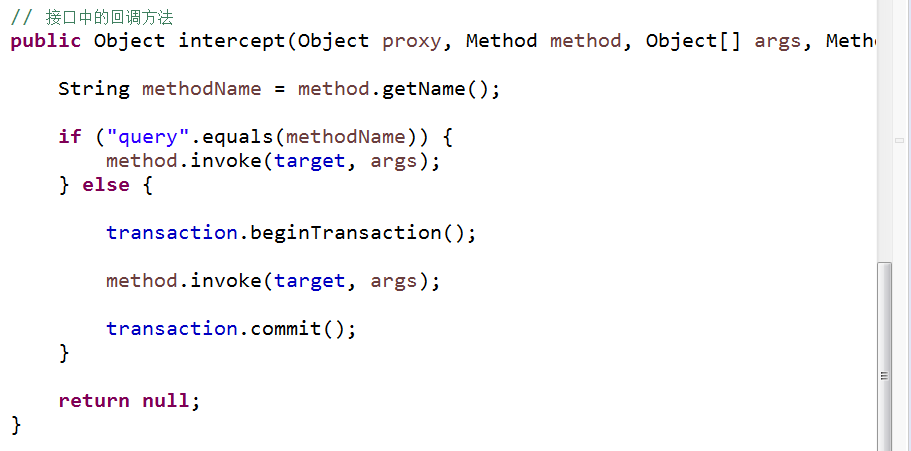


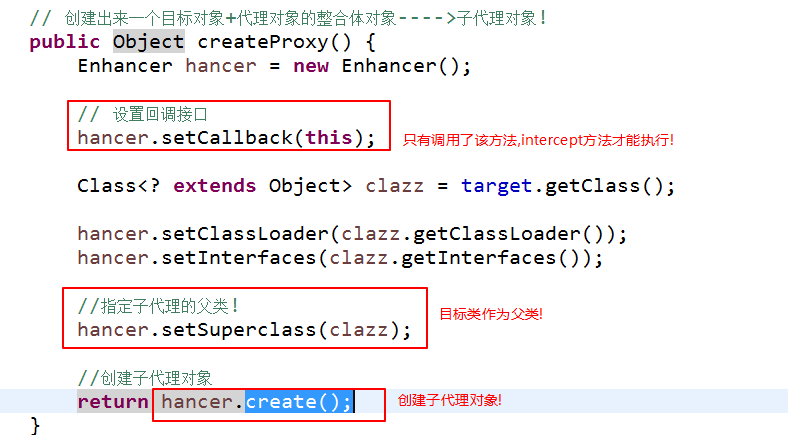
#### 3.定义代理类Transaction



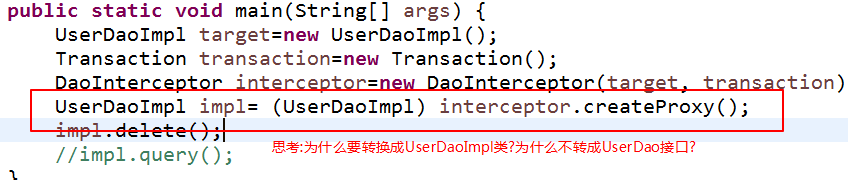
#### 4.实现MethodIntercept接口



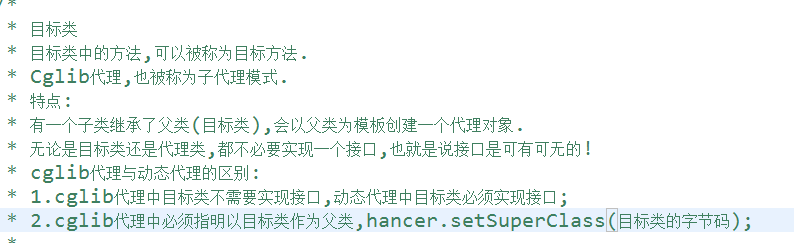




#### 测试类:



#### cglib代理的总结:



## 面向切面编程

### 1.AOP概念

AOP:Aspect Oriented Program,面向切面编程.AOP可以被看成是OOP的一个延续.AOP是一种通过预编译和运行期(动态和cglib)代理技术来对目标程序进行统一维护和扩展的技术.

### 2.AOP编程的优点

AOP编程能够把项目中不同的业务逻辑代码进行隔离,当需要整合在一起的时候,又可以很灵活的进行整合,从而降低了不同模块之间代码的耦合性,提高了代码的复用性,提高了开发效率.

### 3.AOP编程的使用场景

事务管理,日志记录,权限检查,缓存,性能检测等.

### AOP编程的特点

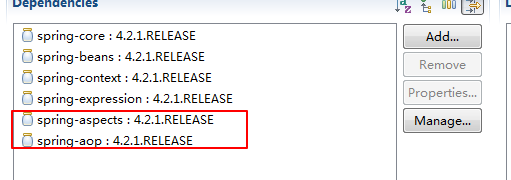
AOP编程最大的特点是它是采用横向的编程模式,取代了以往的纵向编程模式.

### AOP编程的实现原理

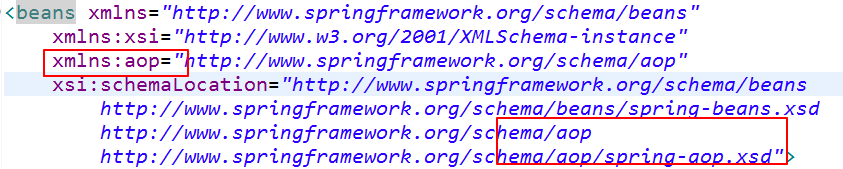
AOP采用Spring自带的API来实现,不需要额外的类加载器以及其他的编译过程,AOP是在程序运行期,通过(动态和cglib)代理方式,将辅助性的代码(weave)给织入到目标代码中,被织入的代码可以被称为增强(Advise通知/增强)代码.

## AOP编程的具体实现

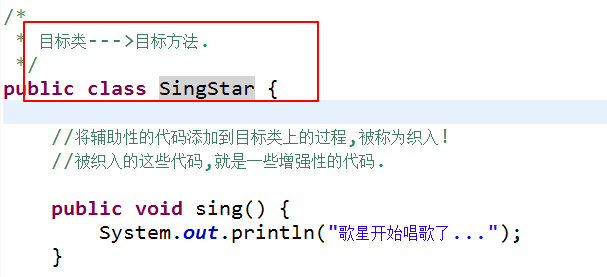
### 1.添加依赖包



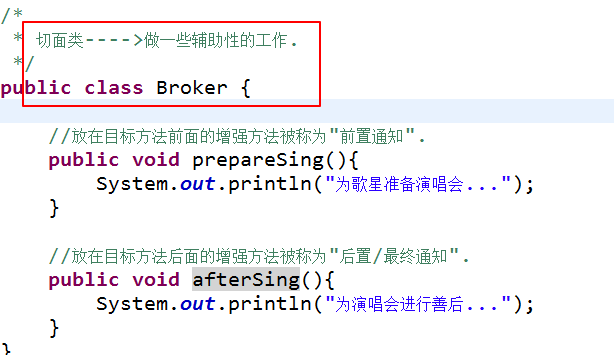
### 2.创建xml文件,添加aop的约束



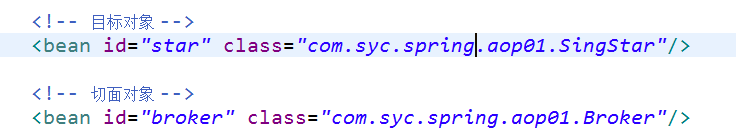
### 3.创建目标类



### 4.创建切面类

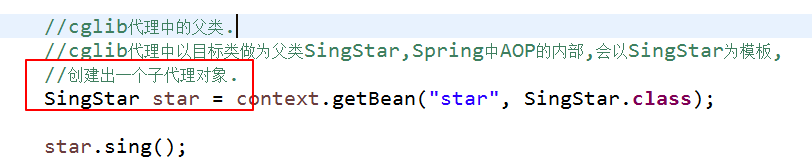


### 5.配置切面



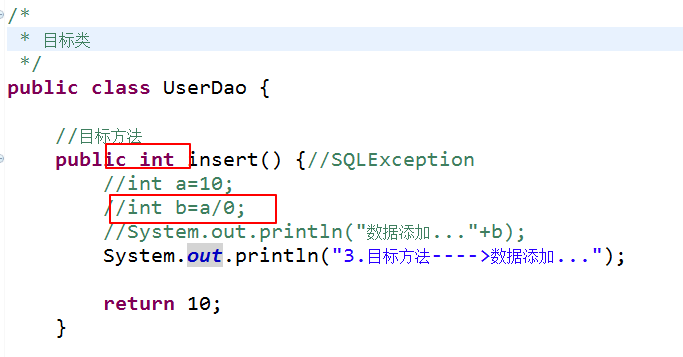


### 6.测试,得到子代理对象



## AOP编程详解

### 1.创建目标类

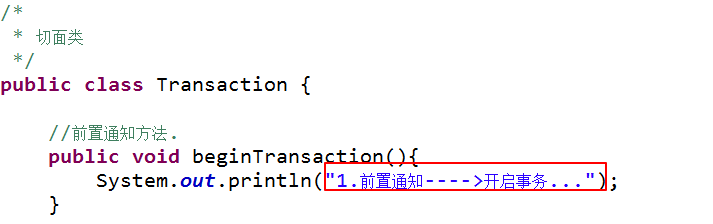


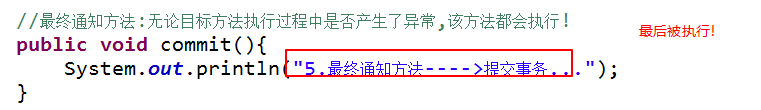
#### 注意:

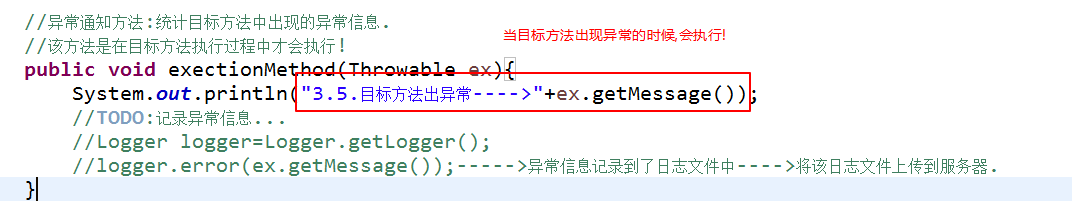
目标方法的返回值以及可能出现的异常问题.

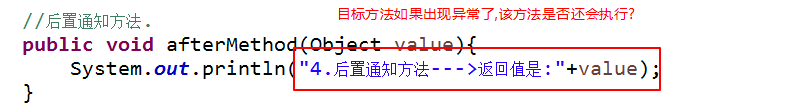
目标方法的返回值返回给环绕通知!

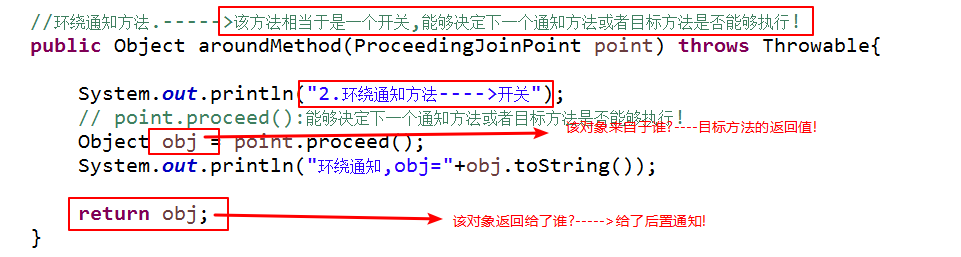
### 2.切面类



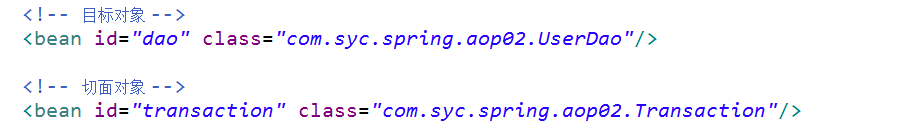






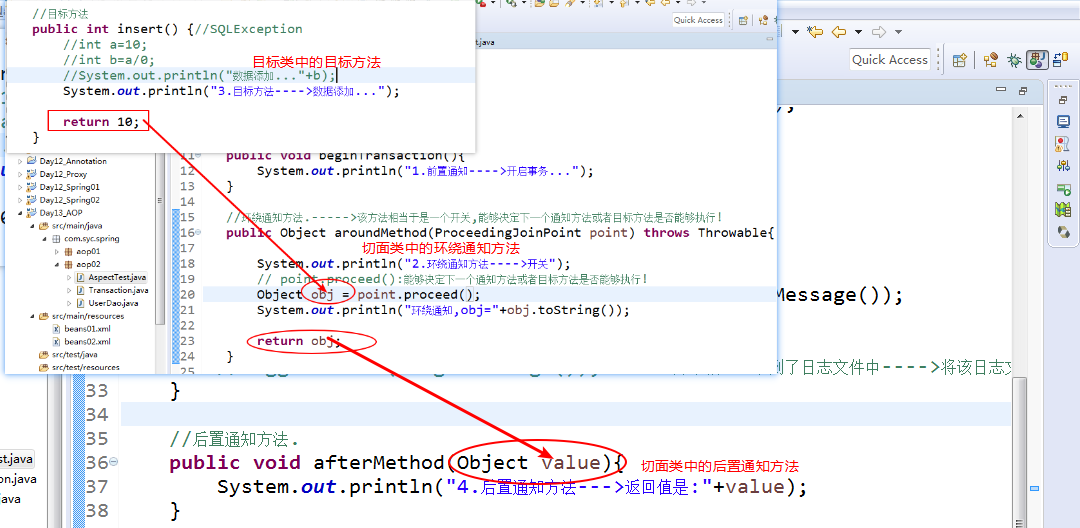


### 3.配置切面aop





### 4.目标方法,环绕方法,后置通知方法之间的关系



### 5.五种通知方法和目标方法的关系

try{

前置通知方法

if(环绕通知方法){

目标方法

后置通知

}

}catch(Exeception e){

异常通知

}finally{

最终通知

}