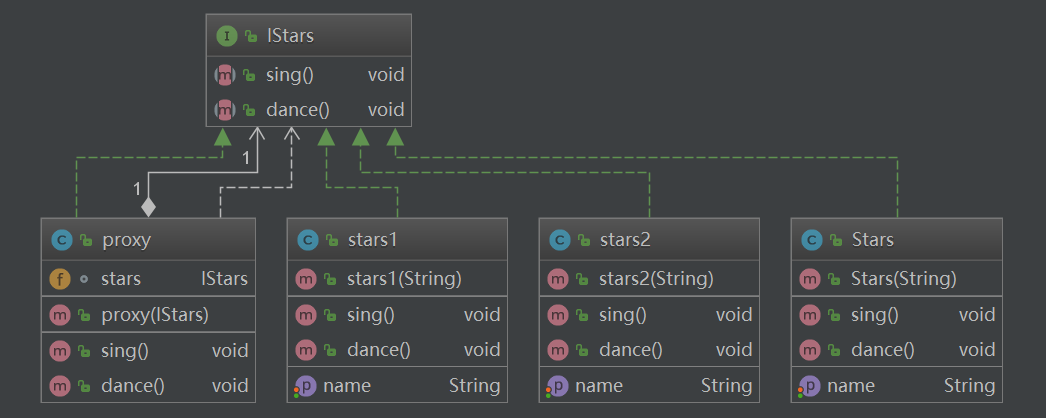
# 静态代理1

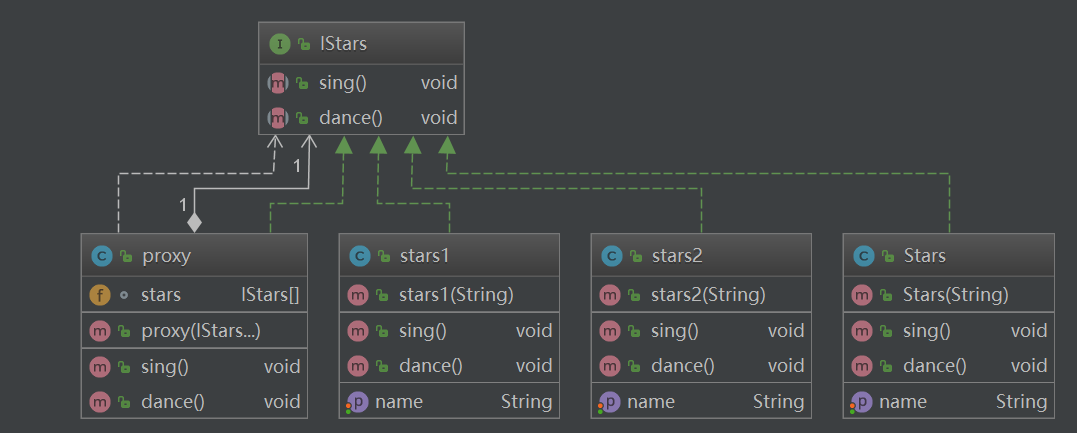


通过代理对象proxy间接调用真实对象的方法。

优点：代理人可以对被代理人做统一前置、后置处理，而无需修改被代理人的代码。方便拓展统一的业务需求。

缺点：一个代理人对应一个被代理人。如果被代理人有N个，这种方式无疑需要N个代理人，**静态代理2**针对这点进行了改进，即一个代理人可以同时代理N个代理对象。

# 静态代理2



对**动态代理1**进行改进。核心是代理对象的构造方法允许接受[0,N]个被代理对象。这更贴合实际业务场景。

优点：一个代理对象允许同时代理多个被代理对象。

缺点：当IStars添加新的方法接口后，代理对象必须跟着手动添加。动态代理模式能动态地生成代理对象，不需要人为去修改代理类的源代码。(无论动态代理还是静态代理，IStars和stars都需要跟着添加新的方法)。

# JDK动态代理

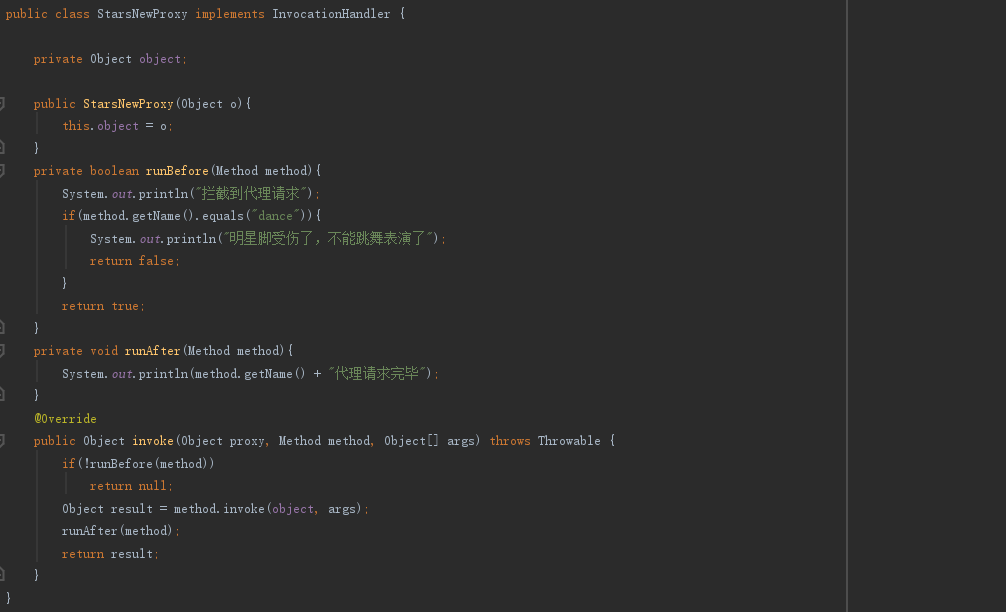
静态代理需要手动生成代理对象，而动态代理利用java机制可以自动生成对应的代理对象。

如果把代理对象看成是对被代理对象一层封装，那动态代理则对代理对象进行更高一层的封装（根据需要生成对象的代理对象，使用自动生成的代理对象完成业务需求）。

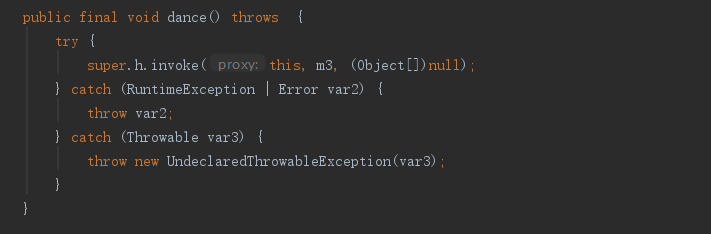
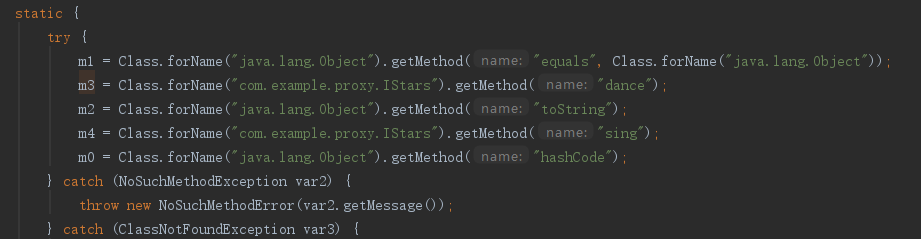
**自定义源码：**



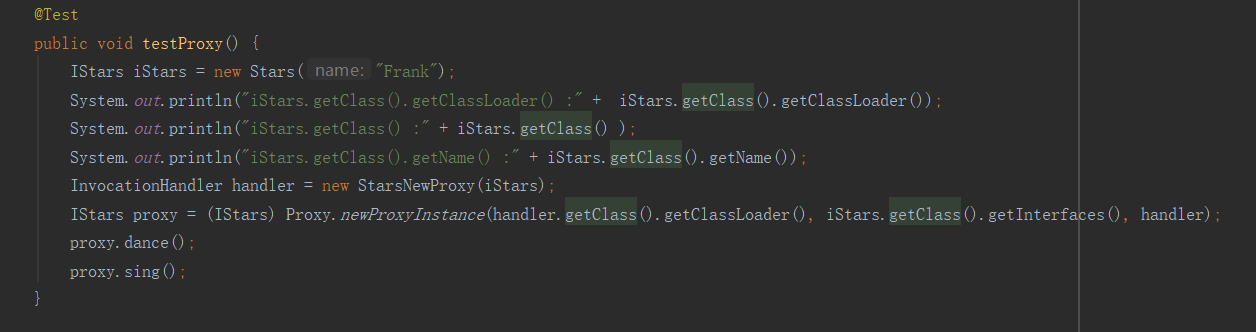




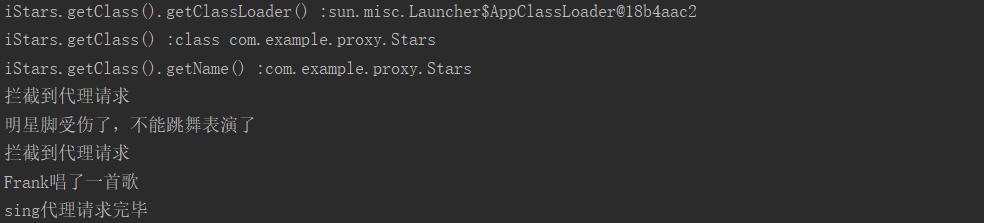
**动态生成的代理对象源码如下两图：**

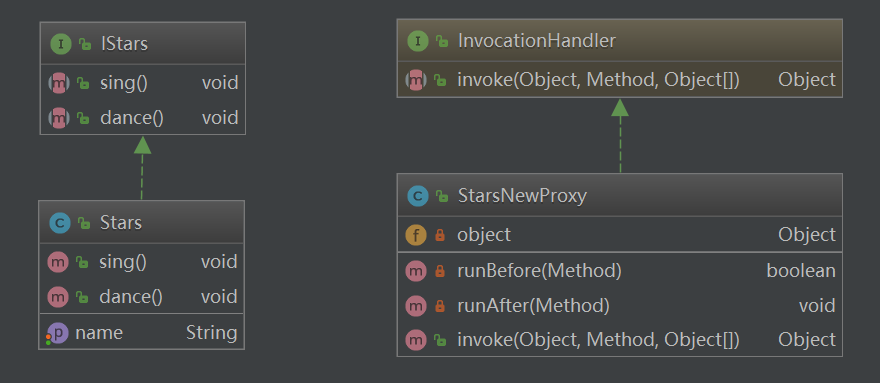
 

**测试类：**



**测试结果：**



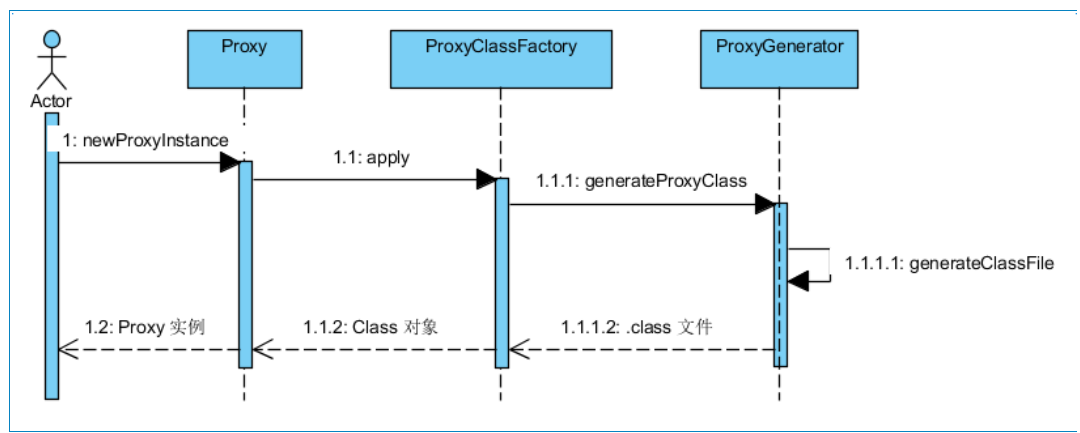


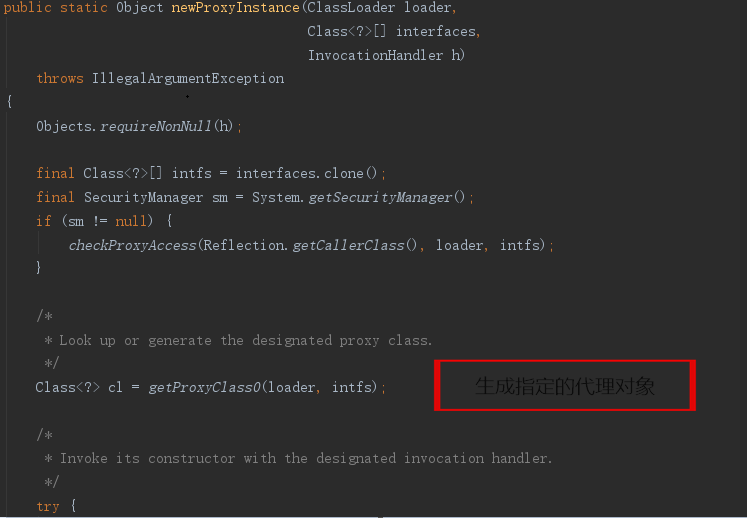
在IStars接口添加了新的方法时，相比静态代理模式，动态代理模式不需要修改代理类，代理类有invoke(…)方法就行了。

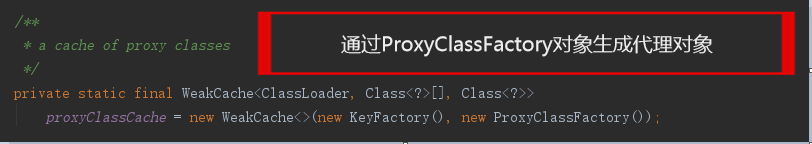
在IStars接口添加了新的方法时，无论静态代理和动态代理，都需要修改实现类Stars的源代码。区别在于是否需要手动修改代理类源代码。并且StarsNewProxy代理类可以为不同类型的类代理，一个足矣！而静态代理只能为某一类事物代理，多类事物必须实现多个代理类。

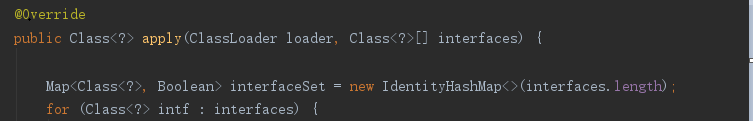
总结静态/动态代理就一句话：动态代理无须预先定义代理类，一个invoke(…)就够了。而静态代理需要预先定义代理类，因此在被代理对象的方法接口变更时，需要跟着修改代理类！

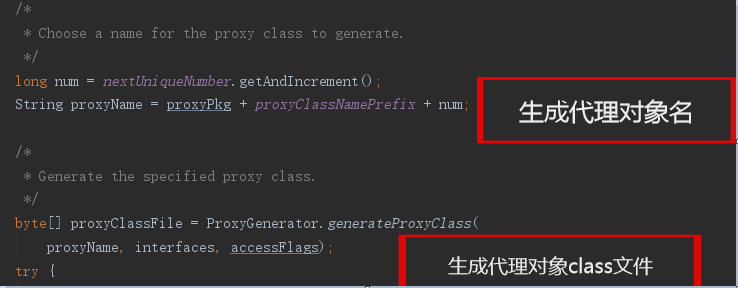
## 原理

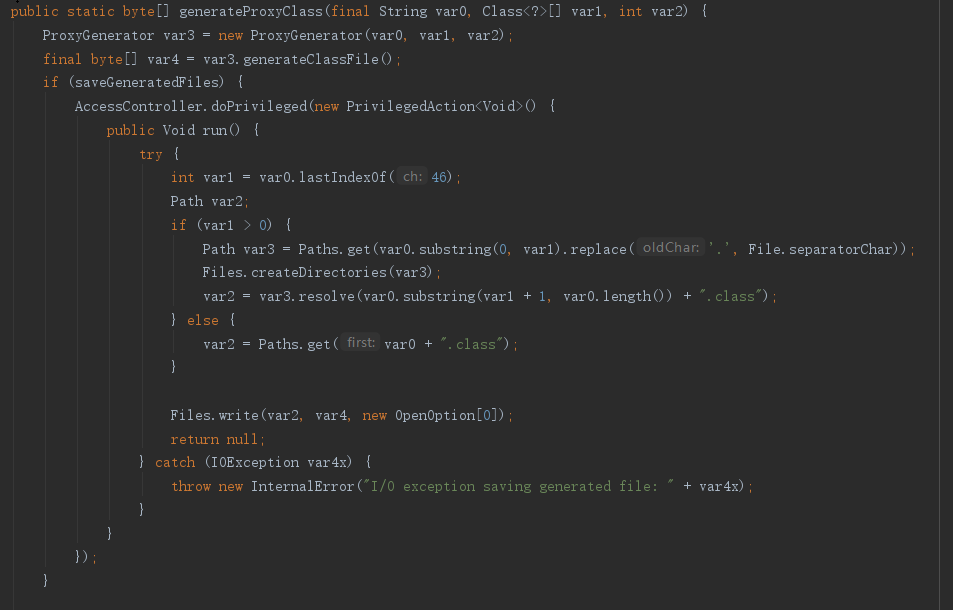










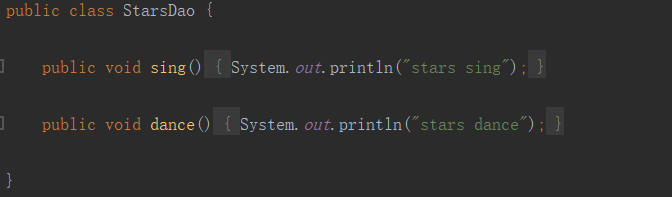
## 查看代理类源码

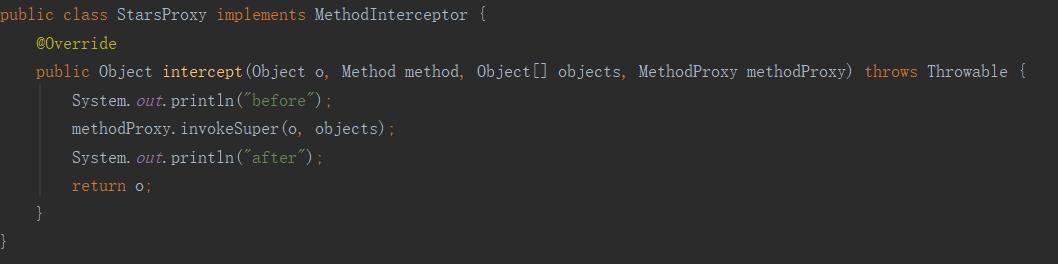
设置系统参数可以在项目类文件夹下找到动态生成的代理对象。

System.getProperties().put("sun.misc.ProxyGenerator.saveGeneratedFiles","true");

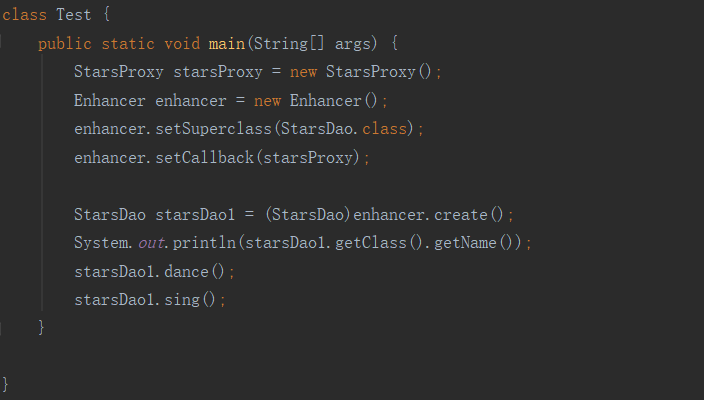
# 动态代理(Cglib)

**源码：**

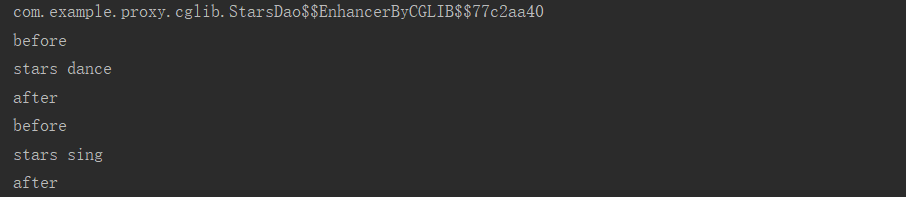




**测试类：**



**测试结果：**



**查看Cglib源码:**

System.setProperty(DebuggingClassWriter.DEBUG\_LOCATION\_PROPERTY, "target/cglib");

# JDK与Cglib区别

JDK与Cglib代理对比：

1：JDK只能针对有接口的类的接口方法进行动态代理；

2：Cglib基于继承来实现代理，无法对static、final类进行代理；

3：Cglib基于继承来实现代理，无法对private、static方法进行代理。