Java静态代理与动态代理的区别

<https://blog.csdn.net/wangqyoho/article/details/77584832>

# Java静态代理

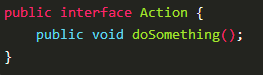
**静态代理通常用于对原有业务逻辑的扩充**。比如持有二方包的某个类，并调用了其中的某些方法。然后出于某种原因，比如记录日志、打印方法执行时间，但是又不好将这些逻辑写入二方包的方法里。所以可以创建一个代理类实现和二方方法相同的方法，通过让代理类持有真实对象，然后在原代码中调用代理类方法，来达到添加我们需要业务逻辑的目的。

**这其实也就是代理模式的一种实现，通过对真实对象的封装，来实现扩展性**。

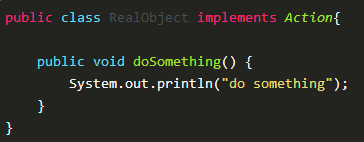
一个典型的代理模式通常有三个角色，这里称之为\*\***代理三要素**\*\*

共同接口、真实对象、代理对象

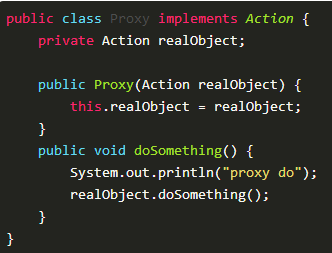
共同接口：

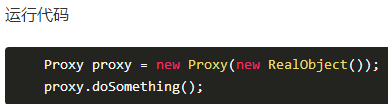
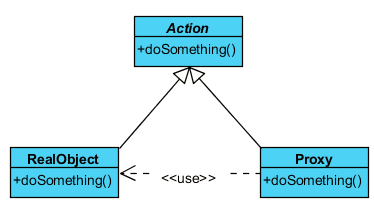


真实对象(被代理对象):



代理对象：



这种代理模式也最为简单，就是通过proxy持有realObject的引用，并进行一层封装。

静态代理的优点和缺点

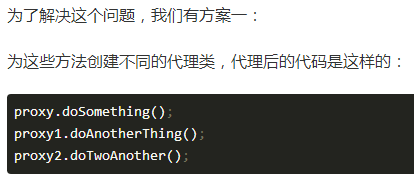
## 静态代理的优点：

扩展原功能，不侵入原代码。

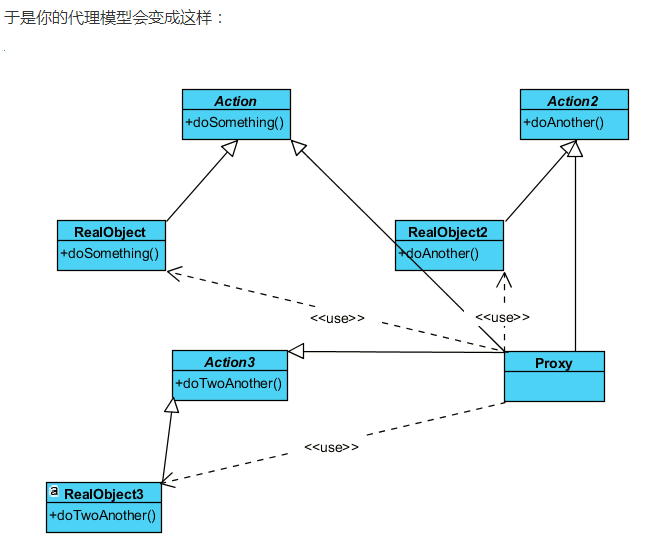
## 静态代理的缺点：

这种代理模式的缺点：

假如有这样一个需求，有十个不同的RealObject，同时我们要去代理的方法是不同的，比要代理方法：doSomething、doAnotherThing、doTwoAnotherThing，添加代理前，原代码可能是这样的：







毫无疑问，仅仅为了扩展同样的功能，在方案一种，我们会重复创建多个逻辑相同，仅仅RealObject引用不同的Proxy。

而在方案二中，会导致proxy的膨胀，而且这种膨胀往往是无意义的。此外，假如方法签名是相同的，更需要在调用的时候引入额外的判断逻辑。

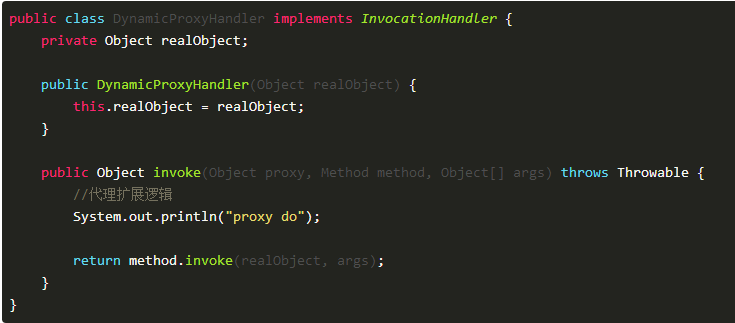
# Java动态代理

搞清楚静态代理的缺点十分重要，**因为动态代理的目的就是为了解决静态代理的缺点**。通过使用动态代理，我们可以通过在运行时，动态生成一个持有RealObject、并实现代理接口的Proxy，同时注入我们相同的扩展逻辑。哪怕你要代理的RealObject是不同的对象，甚至代理不同的方法，都可以动过动态代理，来扩展功能。

简单理解，动态代理就是我们上面提到的方案一，只不过这些proxy的创建都是自动的并且是在运行期生成的。

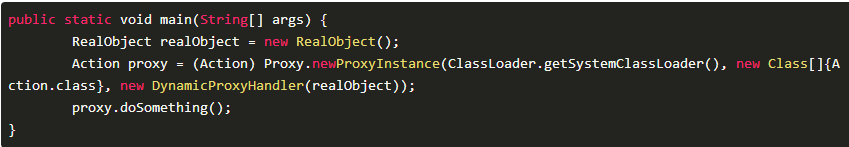
**动态代理基本用法**

使用动态代理，需要将要扩展的功能写在一个**InvocationHandler** 实现类里：



这个Handler中的invoke方法中实现了代理类要扩展的公共功能。

到这里，需要先看一下这个handler的用法：



…..

还有很多东西，请看博客继续对源码的分析。