java 代理模式的那些事

作者：pleaseok
原文链接：https://ld246.com/article/1544358568857
来源网站：链滴

java代理模式-登场

什么是代理模式?

代理模式是java中的一种设计模式，它其实就是设置一个中间环节来代理你要对原目标对象的访问。言之，代理模式就是有一个充当代理者身份的类或方法来控制原对象的引用。

还是不太理解，你能举个例子说明一下吗?

这里一个很好的例子(引用链接)： 一个公司是卖摄像头的，但公司不直接跟用户打交道，而是通过代商跟用户打交道。如果：公司接口中有一个卖产品的方法，那么公司需要实现这个方法，而代理商也须实现这个方法。如果公司卖多少钱，代理商也卖多少钱，那么代理商就赚不了钱。所以代理商在调公司的卖方法后，加上自己的利润然后再把产品卖给客户。而客户不直接跟公司打交道，或者客户根不知道公司的存在，然而客户最终却买到了产品。*

它用途应该挺大的吧?

它是java中常用的设计模式之一，并且在spring框架中有广泛应用(AOP)，它一般有三种模式：静态理、jdk1.6+中的动态代理、cglib动态代理。

java代理模式-静态代理

首先，动态代理是通过反射机制来处理相关业务方法。那么你也猜到了，静态代理则是通过编写代理来完成代理过程。

公司要卖的产品（实现接口）

```
package top.code666.porxy;
//公司中的业务接口
public interface ICompany {
```

```
    //卖摄像头
    void sellCamera();
}
```

公司的实现类（目标类）

```
package top.code666.porxy;
//公司类
public class CompanyImpl implements ICompany{
    @Override
    public void sellCamera() {
        System.out.println("一个很好的Camera，高清无码。(xx公司生产Camera并出售给代理商，
价￥1000)");
    }
}
```

代理商的实现类（代理类）

```
package top.code666.porxy;

//代理类
public class ProxyImpl implements ICompany{
    private CompanyImpl ci;

    public ProxyImpl(CompanyImpl ci){
        this.ci = ci;
    }

    @Override
    public void sellCamera() {
        System.out.println("(卖前打了一波广告)ss微商成为xx公司的最大代理商，现在正式售卖Camer
了！不要998，不要98，只要9998你就可以把它带回家~");
        ci.sellCamera();
        System.out.println("(卖后服务)恭喜你购买成功，三分钟内无条件退换哦~");
    }

}
```

客户（测试类）

```
package top.code666.porxy;

//一个有钱的人
public class RichMan {
    public static void main(String[] args) {
        CompanyImpl ci = new CompanyImpl();
        ProxyImpl pi = new ProxyImpl(ci);
        pi.sellCamera();
    }
}
```

运行结果

(卖前打了一波广告)ss微商成为xx公司的最大代理商，现在正式售卖Camera了！不要998，不要98，要9998你就可以把它带回家~
一个很好的Camera，高清无码。(xx公司生产Camera并出售给代理商，定价￥1000)
(卖后服务)恭喜你购买成功，三分钟内无条件退换哦~

你现在也感觉到了，静态代理就是这么的简单。不过它却存在许多弊端：

1. 如果要代理的产品多了，那么你的整个项目将变得非常冗余。
2. 如果要修改接口，那么你的目标对象和代理对象都要修改，所以不易于维护。

这时我们的动态代理就可以很好的解决这些弊端了。灯登瞪等~，让我们欢迎动态代理闪亮登场吧(方高能，如果你对java的反射机制不是很了解，请先去学习反射相关知识后再过来)

java代理模式-动态代理(JDK1.6+)

JDK1.6以上的版本自带了动态代理方法。我们只需要实现InvocationHandler就可以使用代理了。**前：你的目标对象必须实现接口，否则不能使用JDK动态代理**

接口与目标类是与上一样，所以这里不再重复

代理商的实现类（代理类）

```
package top.code666.jdkproxy;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

//代理对象
public class ProxyImpl implements InvocationHandler{
    //目标对象
    private Object target;

    public ProxyImpl(Object target){
        this.target = target;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args)
            throws Throwable {
        System.out.println("(卖前打了一波广告)ss微商成为xx公司的最大代理商，现在正式售卖Camer
了！不要998，不要98，只要9998你就可以把它带回家~");
        Object result = method.invoke(target, args); // 调用目标类的方法
        System.out.println("(卖后服务)恭喜你购买成功，三分钟内无条件退换哦~");
        return result;
    }

}
```

客户（测试类）

```
package top.code666.jdkproxy;
```

```
import java.lang.reflect.Proxy;

//一个有钱的人
public class RichMan {
    public static void main(String[] args) {
        ICompany ic = new CompanyImpl();
        ProxyImpl pi = new ProxyImpl(ic); //传入目标对象
        ICompany proxySubject = (ICompany) Proxy.newProxyInstance(
                CompanyImpl.class.getClassLoader(),
                CompanyImpl.class.getInterfaces(), pi); // new代理实例
        proxySubject.sellCamera();
    }
}
```

可以发现，在JDK的动态代理中，我们的代理类是不需要再重复写了的，可以共用同一个。但是它需定义接口，然后才能实现代理功能，所以还是存在一定的局限性。下面让我们来看看CGLIB是怎么实代理功能的吧，它会不会不需要就能实现呢?

java代理模式-动态代理(cglib.jar)

cglib可以在目标类不实现接口方法时，也能够给这样的类提供动态代理。而JDK中是不行的。Spring给某个类提供动态代理时会自动在JDK动态代理和CGLIB动态代理中动态的选择。(其实JDK中的动态理是要比CGLIB中动态代理效率要稍微高一点点的)

使用cglib需要导入cglib.jar+asm.jar

目标类(因为是直接复制的，类名啥的我就没改了，Impl或许对一些人来说有点碍眼^……^)

```
package top.code666.cglib;

//公司类
public class CompanyImpl{

    public void sellCamera() {
        System.out.println("一个很好的Camera，高清无码。(xx公司生产Camera并出售给代理商，
价￥1000)");
    }

}
```

代理类

```
package top.code666.cglib;

import java.lang.reflect.Method;

import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

//代理对象
public class ProxyImpl implements MethodInterceptor{
```

```
    @Override
    public Object intercept(Object arg0, Method arg1, Object[] arg2,
            MethodProxy arg3) throws Throwable {
        System.out.println("(卖前打了一波广告)ss微商成为xx公司的最大代理商，现在正式售卖Camer了！不要998，不要98，只要9998你就可以把它带回家~");
        Object result = arg3.invokeSuper(arg0, arg2);
        System.out.println("(卖后服务)恭喜你购买成功，三分钟内无条件退换哦~");
        return result;
    }
}
```

测试类

```
package top.code666.cglib;

import net.sf.cglib.proxy.Enhancer;

//一个有钱的人
public class RichMan {
    public static void  main(String ... args) {
        CompanyImpl target = new CompanyImpl();
          RichMan test = new RichMan();
        CompanyImpl proxyTarget = (CompanyImpl) test.createProxy(CompanyImpl.class);
        proxyTarget.sellCamera();
    }

    public Object createProxy(Class targetClass) {
        Enhancer enhancer = new Enhancer();
        enhancer.setSuperclass(targetClass);
        enhancer.setCallback(new ProxyImpl());
        return enhancer.create();
    }
}
```

JDK动态代理与CGLib动态代理均是实现Spring AOP的基础。

代理对象的生成过程由Enhancer类实现，大概步骤如下：

1. 生成代理类Class的二进制字节码；
2. 通过Class.forName加载二进制字节码，生成Class对象；
3. 通过反射机制获取实例构造，并初始化代理类对象。

总结

1. 静态代理实现较简单，只要代理对象对目标对象进行包装，即可实现增强功能，但静态代理只能为个目标对象服务，如果目标对象过多，则会产生很多代理类。
2. JDK动态代理需要目标对象实现业务接口，代理类只需实现InvocationHandler接口。
3. 动态代理生成的类为 class com.sun.proxy. Proxy4，cglib代理生成的类为class com.cglib.UseraoEnhancerByCGLIB $552188b6。
4. 静态代理在编译时产生class字节码文件，可以直接使用，效率高。

5. 动态代理必须实现InvocationHandler接口，通过反射代理方法，比较消耗系统性能，但可以减少理类的数量，使用更灵活。

6. cglib代理无需实现接口，通过生成类字节码实现代理，比反射稍快，不存在性能问题，但cglib会承目标对象，需要重写方法，所以目标对象不能为final类。