**手绘6张图彻底搞懂动态代理**

<https://mp.weixin.qq.com/s/eXX6UUtQknYvMYHrRchx5g>

**动态代理实现之 JDK**

JDK 原生提供了动态代理的实现，主要是通过java.lang.reflect.Proxy和java.lang.reflect.InvocationHandler这两个类配合使用。

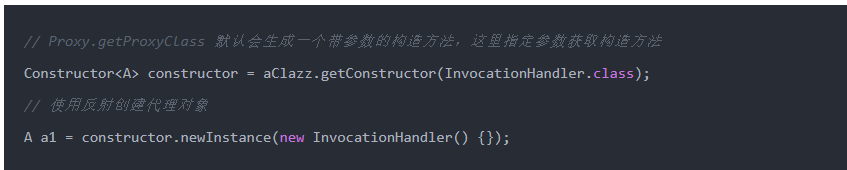
Proxy类有个静态方法，传入类加载器和一组接口就可以返回代理 Class 对象。

public static Class<?> getProxyClass(ClassLoader loader, Class<?>... interfaces)

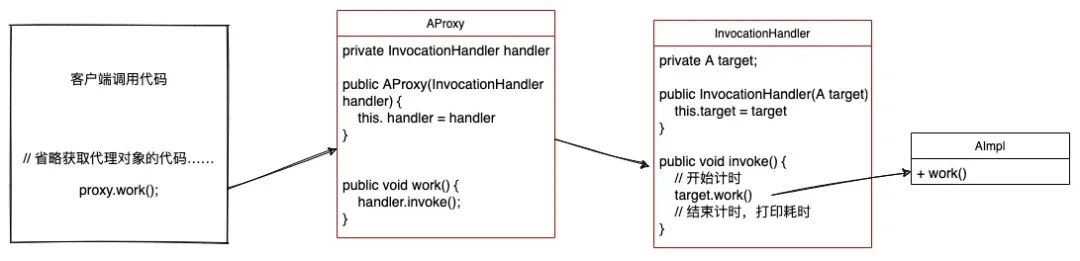
这个方法的作用简单来说就是，会将你传入一组接口类的结构信息"拷贝"到一个新的 Class 对象中，新的 Class对象带有构造器是可以创建对象的。

一句话总结：Proxy.getProxyClass() 这个静态方法的本质是**以 Class 造 Class**。

拿到了 Class 对象，就可以使用反射创建实例对象了：



眼尖的同学已经看到了，创建实例的时候需要传入一个 InvocationHandler 对象，说明代理对象中必然有一个成员变量去接收。在调用代理对象的方法时实际上会去执行 InvocationHandler 对象的 invoke方法，画个图理解一下：



invoke 方法里可以写增强代码，然后调用目标对象 work 方法。

总结一下流程：

（1）通过 Proxy.getProxyClass() 方法获取代理类 Class 对象；

（2）通过反射 aClazz.getConstructor() 获取构造器对象；

（3）定义InvocationHandler类并实例化，当然也可以直接使用匿名内部类；

（4）通过反射 constructor.newInstance() 创建代理类对象；

（5）调用代理方法；

看了上面的流程，是不是觉得比静态代理还要繁琐，有没有更加优雅的方法？当然有！

为了尽量简化操作，JDK Proxy 类直接提供了一个静态方法：

public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)

**问题 1：cglib 和 JDK 动态代理的区别？**

* JDK 动态代理：利用 InvocationHandler 加上反射机制生成一个代理接口的匿名类，在调用具体方法前调用InvokeHandler来处理
* cglib 动态代理：利用ASM框架，将目标对象类生成的class文件加载进来，通过修改其字节码生成代理子类

**问题 2：cglib 比 JDK快？**

* cglib底层是ASM字节码生成框架，在 JDK 1.6 前字节码生成要比反射的效率高
* 在 JDK 1.6 之后 JDK 逐步对动态代理进行了优化，在 1.8 的时候 JDK 的效率已经高于 cglib

**问题 3：Spring框架什么时候用 cglib 什么时候用 JDK 动态代理？**

* 目标对象生成了接口默认用 JDK 动态代理
* 如果目标对象没有实现接口，必须采用cglib
* 当然如果目标对象使用了接口也可以强制使用cglib