目录

**键入章标题(第 1 级)1**

键入章标题(第 2 级)2

键入章标题(第 3 级)3

**键入章标题(第 1 级)4**

键入章标题(第 2 级)5

键入章标题(第 3 级)6

# 1.动态代理

## Jdk动态代理

Java基于接口的动态代理是一种在运行时动态生成代理类的技术。它是通过反射机制在运行时生成代理对象，对被代理对象的方法进行拦截处理，实现增强功能。

Java基于接口的动态代理需要实现两个接口：InvocationHandler和Proxy。InvocationHandler接口中定义了一个invoke方法，该方法接收一个代理对象、被代理对象的方法和参数，并返回代理对象对被代理对象的方法进行增强后的返回值。Proxy类中则提供了一个静态方法newProxyInstance，用于创建代理对象

用途:记录方法调用时长、打印日志、权限校验

## Cglib动态代理

Java CGLIB 动态代理是一种在运行时生成代理对象的技术，它可以在不修改原始类型（类）的基础上，为该类型创建一个代理子类，该代理子类可以拦截原始类中的方法调用。相较于Java JDK动态代理，它可以代理没有实现接口的类。

## Jdk动态代理为什么只能代理有接口的类

Jdk动态代理本身机制决定的。

Jdk中的代理类是通过Proxy.newProxyInstance方法实现的，该方法需要传入动态代理接口类。之所以要传入接口而不能传入类，取决于底层实现。Jdk会在运行时，会生成一个代理类叫$proxy0，这个动态生成的代理类回去继承一个java.lang.reflect.Proxy类，同时还会实现被代理类的接口，java里面不支持多继承，而每个动态代理类都集成了一个Proxy，所以就导致jdk里面的动态代理只能代理接口，而不能代理实现类。Proxy只是保存了动态代理的一个处理器叫InvocationHandler，如果不抽出来，直接设置到$Proxy0这个动态代理里面，我认为也是可以得，如果这么去实现的话，就可以针对实现类来做动态代理了。

作者为什么这么设计，我认为还是有几个方面的原因：

1. 动态代理本身的使用场景或者需求，只是对原始实现的一个拦截，然后去做一些功能过的增强或者扩展。而实际的开发模式都是基于面向接口来开发的，所以基于接口来实现动态代理，从需求和场景，我认为都是吻合的。当然确实会存在一些没有实现接口的实现类，这个时候去实现动态代理，jdk很显然是无法满足的。
2. Java里面类的继承关系的设计更多的是考虑到共性能力的抽象，从而去提高代码的重用性和扩展性，动态代理也在做这样一件事，它封装了动态代理类的生成的抽象逻辑以及判断一个类是否是动态代理类，把这些抽象逻辑放到Proxy这个父类里面，很显然也是一个比较正常的思路。技术的方案设计是解决特定的场景问题的，如果一定要正对普通类做动态代理，我们可以选择cglib，它会动态生成一个被代理类的子类，子类重写父类的所有非final修饰的方法

上层的框架可能会随着业务的发展而变化，但是底层技术原理和设计理念是不会变化的。底层逻辑是对一个事务本质上的一个认识，而不是停留在表面上。

## 区别

JDK 代理 : 基于接口的动态代理技术·：利用拦截器（必须实现InvocationHandler）加上反射机制生成

一个代理接口的匿名类，再调用具体方法前调用InvokeHandler来处理，从而实现方法增强

CGLIB 代理 : 基于父类的动态代理技术：动态生成一个要代理的子类，子类重写要代理的类的所有不是

final的方法。在子类中采用方法拦截技术拦截所有的父类方法的调用，顺势织入横切逻辑，对方法进行

增强