**面试官提问：什么是动态代理？**

<https://mp.weixin.qq.com/s/bCGncYhu41OJXKCzGeS5Nw>

**一、介绍**

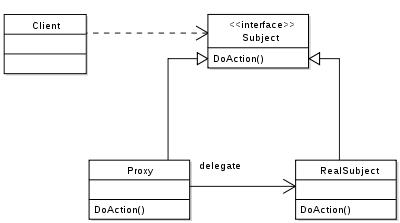
**何谓代理？**

**代理**这个词最早出现在**代理商**这个行业，所谓**代理商**，简而言之，其实就是**帮助企业或者老板打理生意，自己本身不做生产任何商品**。

举个例子，我们去火车站买票的时候，人少老板一个人还忙的过来，但是人一多的话，就会非常拥挤，于是就有了各种代售点，我们可以从代售点买车票，从而加快老板的卖票速度。

**代售点的出现，可以说，很直观的帮助老板提升了用户购票体验**。

站在软件设计的角度，其实效果也是一样的，采用代理模式的编程，能显著的增强原有的功能和简化方法调用方式。



在介绍动态代理之前，我们先来聊解静态代理。

### 二、静态代理

通过这种代理方式，最大的优点就是：**可以在不修改目标对象的前提下，扩展目标对象的功能**。

但也有缺点：需要**代理对象和目标对象实现一样的接口**，因此，当目标对象扩展新的功能时，代理对象也要跟着一起扩展，不易维护！

### 三、动态代理

动态代理，其实本质也是为了解决上面当目标对象扩展新功能时，代理对象也需要跟着一起扩展的痛点问题而生。

**那它是怎么解决的呢？**

以 JDK 为例，当需要给某个目标对象添加代理处理的时候，JDK 会在内存中动态的构建代理对象，从而实现对目标对象的代理功能。

下面，我们还是以两数相加为例，介绍具体的玩法！

#### 3.1、JDK 中生成代理对象的玩法

1. 首先定义一个 InvocationHandler 实例，它负责实现接口的方法调用

2. 通过 Proxy.newProxyInstance() 创建 interface 实例，它需要 3 个参数:

（1）使用的 ClassLoader，通常就是接口类的 ClassLoader

（2）需要实现的接口数组，至少需要传入一个接口进去；

（3）用来处理接口方法调用的 InvocationHandler 实例。

3. 将返回的 Object 强制转型为接口

#### 3.2、cglib 生成代理对象的玩法

除了 jdk 能实现动态的创建代理对象以外，还有一个非常有名的第三方框架：cglib，它也可以做到运行时在内存中动态生成一个子类对象从而实现对目标对象功能的扩展。

cglib 特点如下：

* cglib 不仅可以代理接口还可以代理类，而 JDK 的动态代理只能代理接口
* cglib 是一个强大的高性能的代码生成包，它广泛的被许多 AOP 的框架使用，例如我们所熟知的 Spring AOP，cglib 为他们提供方法的 interception（拦截）。
* CGLIB包的底层是通过使用一个小而快的字节码处理框架ASM，来转换字节码并生成新的类，速度非常快。

在使用 cglib 之前，我们需要添加依赖包，如果你已经有spring-core的jar包，则无需引入，因为spring中包含了cglib。

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>3.2.5</version>

</dependency>

**四、静态织入**

在上文中，我们介绍的代理方案都是在代码运行时动态的生成class文件达到动态代理的目的。

回到问题的本质，其实动态代理的技术目的，主要为了解决静态代理模式中当目标接口发生了扩展，代理类也要跟着一遍变动的问题，避免造成了工作伤的繁琐和复杂。

在 Java 生态里面，还有一个非常有名的第三方代理框架，那就是AspectJ，AspectJ通过特定的编译器可以将目标类编译成class字节码的时候，在方法周围加上业务逻辑，从而达到静态代理的效果。

采用AspectJ进行方法植入，主要有四种：

* 方法调用前拦截
* 方法调用后拦截
* 调用方法结束拦截
* 抛出异常拦截

使用起来也非常简单，首先是在项目中添加AspectJ编译器插件。

**五、小结**

看到上面的介绍静态织入方案，跟我们现在使用Spring AOP的方法极其相似，可能有的同学会发出疑问，我们现在使用的Spring AOP动态代理，**到底是动态生成的还是静态织入的呢**？

实际上，Spring AOP代理是对JDK代理和CGLIB代理做了一层封装，同时引入了AspectJ中的一些注解@pointCut、@after，@before等等，**本质是使用的动态代理技术**。

总结起来就三点：

* 如果目标是接口的话，默认使用 JDK 的动态代理技术；
* 如果目标是类的话，使用 cglib 的动态代理技术；
* 引入了AspectJ中的一些注解@pointCut、@after，@before，主要是为了简化使用，跟AspectJ的关系并不大；

那为什么Spring AOP不使用AspectJ这种静态织入方案呢？

虽然AspectJ编译器非常强，性能非常高，但是只要目标类发生了修改就需要重新编译，主要原因可能还是AspectJ的编译器太过于复杂，还不如动态代理来的省心！