代理模式：

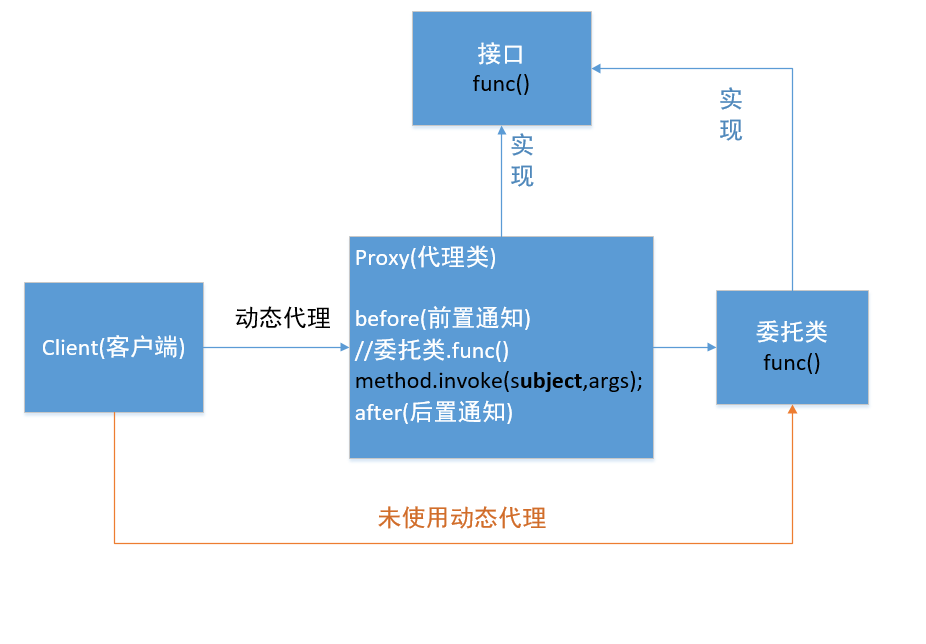
一个公共接口，一个委托类，一个代理类  
代理模式就是在访问实际对象时引入一定程度的间接性，因为这种间接性  
可以附加多种用途。例如前置通知，后置通知。  
代理类存在的意义就是为了增加一些公共代码逻辑。

## Java 代理模式实现方式，主要有如下五种方法

* 静态代理，工程师编辑代理类代码，实现代理模式；在编译期就生成了代理类。
* 基于 JDK 实现动态代理，通过jdk提供的工具方法Proxy.newProxyInstance动态构建全新的代理类(继承Proxy类，并持有InvocationHandler接口引用 )字节码文件并实例化对象返回。(jdk动态代理是由java内部的反射机制来实例化代理对象，并代理的调用委托类方法)
* 基于CGlib 动态代理模式 基于继承被代理类生成代理子类，不用实现接口。只需要被代理类是非final 类即可。(cglib动态代理底层是借助asm字节码技术
* 基于 Aspectj 实现动态代理（修改目标类的字节，织入代理的字节，在程序编译的时候 插入动态代理的字节码，不会生成全新的Class ）
* 基于 instrumentation 实现动态代理（修改目标类的字节码、类装载的时候动态拦截去修改，基于javaagent） -javaagent:spring-instrument-4.3.8.RELEASE.jar （类装载的时候 插入动态代理的字节码，不会生成全新的Class ）

## Notes

* 委托类 即指的是代理模式中的被代理对象
* 代理类 指的是生成的代表委托类的一个角色



1.静态代理：

工程师编辑代理类代码，实现代理模式；在编译期就生成了代理类。

2.动态代理

动态代理类是代理类在程序运行时创建的代理模式。

相比于静态代理， 动态代理的优势在于可以很方便的对代理类的函数进行统一的处理，而不用修改每个代理类中的方法。

动态代理涉及到的角色：

1. 公共接口（委托类，代理类都需要实现这个接口）
2. 委托类
3. InvocationHandler接口被Proxy类回调处理，一般实现 InvocationHandler 接口的类具有委托类引用，接口方法 invoke 中添加公共代码并调用委托类

的接口方法。

1. JDK提供生成动态代理类的核心类Proxy ( JDK 提供的Proxy.java)

动态代理使用场景，有哪些类别?

1. 基于接口的代理（jdk动态代理）

代理类继承了Proxy类，Java的继承机制决定了JDK动态代理类们无法实现对 类 的动态代理。所以也就决定了java动态代理只能对接口进行代理。

如果业务实现类是没有实现接口而是直接定义业务方法的话，就无法使用JDK动态代理了。

1. 基于类的代理（字节类增强的类代理，cglib,javassist）

基于继承。

Cglib动态代理，它的原理是对代理的委托类生成一个子类，并覆盖其中方法实现增强，因为底层是基于创建被代理类的一个子类，所以它避免了动态代理的缺陷。

但因为采用的是继承，所以不能对final修饰的类进行代理。final修饰的类不可继承。

Cglib 总结

* CGlib可以传入接口也可以传入普通的类，接口使用实现的方式,普通类使用会使用继承的方式生成代理类.
* 由于是继承方式,如果是 static方法,private方法,final方法等描述的方法是不能被代理的
* 做了方法访问优化，使用建立方法索引的方式避免了传统JDK动态代理需要通过Method方法反射调用.
* 提供callback 和filter设计，可以灵活地给不同的方法绑定不同的callback。编码更方便灵活。
* CGLIB会默认代理Object中equals,toString,hashCode,clone等方法。比JDK代理多了clone。

#### 基于 AspectJ 的编译时增强进行 AOP POM 依赖

原生 AspectJ 不依赖Spring案例, 基于 AspectJ 的编译时增强进行 AOP 它是在编译期修改字节码，增强功能；并不会生成新的代理类字节码。

<!-- AspectJ begin--><dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjrt</artifactId>

<version>1.9.2</version></dependency><dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.2</version></dependency><!-- AspectJ stop-->

## 动态代理 使用场景

* 日志集中打印
* 事务
* 权限管理
* AOP