动力节点
POWER NODE



北京动力节点教育科技有限公司

动力节点课程讲义

DONGLIJIEDIANKECHENGJIANGYI

www.bjpowernode.com

代理模式

> 代理模式是指，为其他对象提供一种代理以控制对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户类和目标对象之间起到中介的作用。
>
> 百度百科《代理模式》

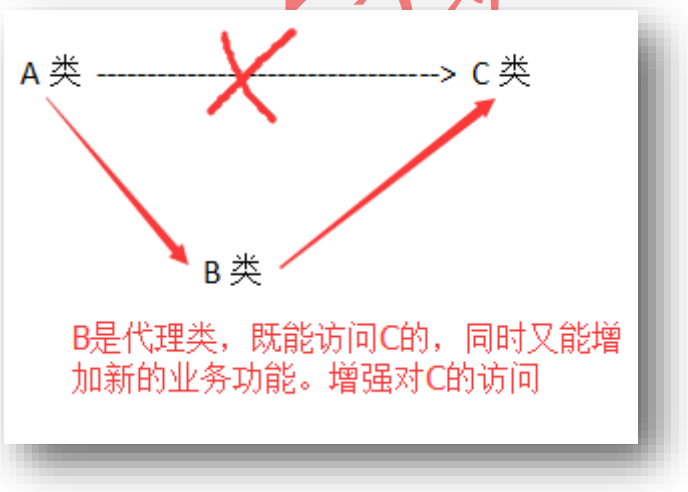
换句话说，使用代理对象，是为了在不修改目标对象的基础上，增强主业务逻辑。

客户类真正的想要访问的对象是目标对象，但客户类真正可以访问的对象是代理对象。客户类对目标对象的访问是通过访问代理对象来实现的。当然，代理类与目标类要实现同一个接口。

例如： 有 A，B，C 三个类， A 原来可以调用 C 类的方法， 现在因为某种原因 C 类不允许 A 类调用其方法，但 B 类可以调用 C 类的方法。A 类通过 B 类调用 C 类的方法。这里 B 是 C 的代理。 A 通过代理 B 访问 C.

原来的访问关系：

通过代理的访问关系：



Window 系统的快捷方式也是一种代理模式。快捷方式代理的是真实的程序，双击快捷方式是启动它代表的程序。

1.1 代理模式作用

A、 控制访问

B、 增强功能

1.2 代理模式分类

可以将代理分为两类：静态代理与动态代理

1.3 代理的实现方式

静态代理和动态代理

1.4 需求

需求：用户需要购买 u 盘， u 盘厂家不单独接待零散购买，厂家规定一次最少购买 1000 个以上，用户可以通过淘宝的代理商，或者微商哪里进行购买。

淘宝上的商品，微商都是 u 盘工厂的代理商， 他们代理对 u 盘的销售业务。

用户购买-------代理商（淘宝，微商）----- u 厂家（金士顿，闪迪等不同的厂家）

设计这个业务需要的类：

1. 商家和厂家都是提供 sell 购买 u 盘的方法。定义购买 u 盘的接口 UsbSell
2. 金士顿（King）对购买 1 千以上的价格是 85, 3 千以上是 80, 5 千以上是 75。 单个 120 元。定义 UsbKingFactory 类，实现 UsbSell
3. 闪迪（San）对购买 1 千以上的价格是 82, 3 千以上是 78, 5 千以上是 72。 单个 120 元。定义 UsbSanFactory 类，实现 UsbSell
4. 定义淘宝的代理商 TaoBao ，实现 UsbSell
5. 定义微商的代理商 WeiShang，实现 UsbSell
6. 定义测试类，测试通过淘宝， 微商购买 u 盘

1.5 静态代理

静态代理是指，代理类在程序运行前就已经定义好.java 源文件，其与目标类的关系在程序运行前就已经确立。在程序运行前代理类已经编译为.class 文件。

1.5.1 静态代理

在 idea 中创建 java 工程，
工程名称：ch01-staticproxy

（1） 定义业务接口

定义业务接口 UsbSell（目标接口），其中含有抽象方法 sell(int amount)， sell 是目标方法。

```
public interface UsbSell {
    float sell(int amount);
}
```

（2） 定义接口实现类

目标类 UsbKingFactory(金士顿 u 盘)，该类实现了业务接口。

```
//金士顿厂商
public class UsbKingFactory implements UsbSell {
    @Override
    public float sell(int amount){
        // 128G u盘代理商购买价格
        // 可以根据购买 amount 有其他优惠
        return 90.0F;
    }
}
```

（3） 代理商 TaoBao

TaoBao 就是一个代理类， 代理厂家销售 u 盘

```java
public class Taobao implements UsbSell {
    private UsbKingFactory factory = new UsbKingFactory();
    @Override
    public float sell(int amount)  {
        float price = factory.sell(amount);
        //在单价之上 ，增加25元作为利润
        return price + 25;
    }
}
```

（4） 代理商 WeiShang

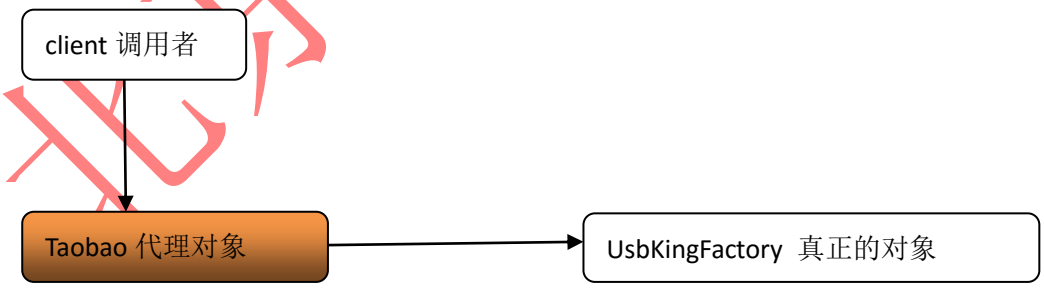
WeiShang 就是一个代理类， 代理厂家销售 u 盘

```java
public class WeiShang implements UsbSell {

    private UsbKingFactory factory = new UsbKingFactory();
    @Override
    public float sell(int amount) {
        float price = factory.sell(amount);
        //在单价之上 ，增加25元作为利润
        return price + 25;
    }
}
```

（5） 客户端调用者，购买商品类

```
public class ShopApplication {
    public static void main(String[] args) {
        float price = 0.0f;
        Taobao taobao = new Taobao();
        try {
            price = taobao.sell(1);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("taobao 购买价格price:"+price);
        System.out.println("================================");

        WeiShang wei = new WeiShang();
        try {
            price = wei.sell(1);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("weishang 购买价格price:"+price);
    }
}
```

使用代理的访问关系图：



1.5.2 静态代理的缺点

（1） 代码复杂，难于管理

代理类和目标类实现了相同的接口，每个代理都需要实现目标类的方法，这样就出现了大量的代码重复。如果接口增加一个方法，除了所有目标类需要实现这个方法外，所有代理类也需要实现此方法。增加了代码维护的复杂度。

（2） 代理类依赖目标类，代理类过多

代理类只服务于一种类型的目标类，如果要服务多个类型。势必要为每一种目标类都进行代理，静态代理在程序规模稍大时就无法胜任了，代理类数量过多。

1.6动态代理

动态代理是指代理类对象在程序运行时由 JVM 根据反射机制动态生成的。动态代理不需要定义代理类的.java 源文件。

动态代理其实就是 jdk 运行期间，动态创建 class 字节码并加载到 JVM。

动态代理的实现方式常用的有两种：使用 JDK 代理代理，与通过 CGLIB 动态代理。

1.6.1 jdk 的动态代理

jdk 动态代理是基于 Java 的反射机制实现的。使用 jdk 中接口和类实现代理对象的动态创建。

Jdk 的动态要求目标对象必须实现接口，这是 java 设计上的要求。

从 jdk1.3 以来，java 语言通过 java.lang.reflect 包提供三个类支持代理模式 Proxy, Method 和 InovcationHandler。

（1） InvocationHandler 接口

InvocationHandler 接口叫做调用处理器，负责完调用目标方法，并增强功能。

通过代理对象执行目标接口中的方法，会把方法的调用分派给调用处理器(InvocationHandler)的实现类，执行实现类中的 invoke()方法，我们需要把功能代理写在 invoke（）方法中 。

```
public interface InvocationHandler {
```

接口中只有一个方法：

```
public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable;
```

在 invoke 方法中可以截取对目标方法的调用。在这里进行功能增强。Java 的动态代理是建立在反射机制之上的。

实现了 InvocationHandler 接口的类用于加强目标类的主业务逻辑。这个接口中有一个方法 invoke()，具体加强的代码逻辑就是定义在该方法中的。通过代理对象执行接口中的方法时，会自动调用 invoke()方法。

invoke()方法的介绍如下：

public Object invoke (Object proxy, Method method, Object[] args) proxy：代表生成的代理对象 method：代表目标方法 args：代表目标方法的参数

第一个参数 proxy 是 jdk 在运行时赋值的，在方法中直接使用，第二个参数后面介绍，第三个参数是方法执行的参数，这三个参数都是 jdk 运行时赋值的，无需程序员给出。

（2） Method 类

invoke()方法的第二个参数为 Method 类对象，该类有一个方法也叫 invoke()，可以调用目标方法。这两个 invoke()方法，虽然同名，但无关。

public Object invoke (Object obj, Object... args) obj：表示目标对象 args：表示目标方法参数，就是其上一层 invoke 方法的第三个参数

该方法的作用是：调用执行 obj 对象所属类的方法，这个方法由其调用者 Method 对象确定。

在代码中，一般的写法为

method.invoke(target, args);

其中，method 为上一层 invoke 方法的第二个参数。这样，即可调用了目标类的目标方法。

（3） Proxy 类

通过 JDK 的 java.lang.reflect.Proxy 类实现动态代理，会使用其静态方法 newProxyInstance()，依据目标对象、业务接口及调用处理器三者，自动生成一个动态代理对象。

public static newProxyInstance (ClassLoader loader, Class<?>[] interfaces,
InvocationHandler handler)
loader：目标类的类加载器，通过目标对象的反射可获取
interfaces：目标类实现的接口数组，通过目标对象的反射可获取
handler：调用处理器。

1.6.2 jdk 动态代理实现

jdk 动态代理是代理模式的一种实现方式，其只能代理接口。
实现步骤
1、新建一个接口，作为目标接口
2、为接口创建一个实现类，是目标类
3、创建类实现 java.lang.reflect.InvocationHandler 接口，调用目标方法并增加其他功能代码
4、创建动态代理对象，使用 Proxy.newProxyInstance()方法，并把返回值强制转为接口类型。

idea 创建 java project
工程名称：ch02-dynamicproxy

（1） 定义目标接口

```
public interface UsbSell {
    float sell(int amount);
}
```

（2） 定义目标接口实现类

```
//金士顿厂商
public class UsbKingFactory implements UsbSell {
    @Override
    public float sell(int amount){
        // 128G u盘代理商购买价格
        // 可以根据购买 amount 有其他优惠
        return 90.0F;
    }
}
```

（3） 定义调用处理程序

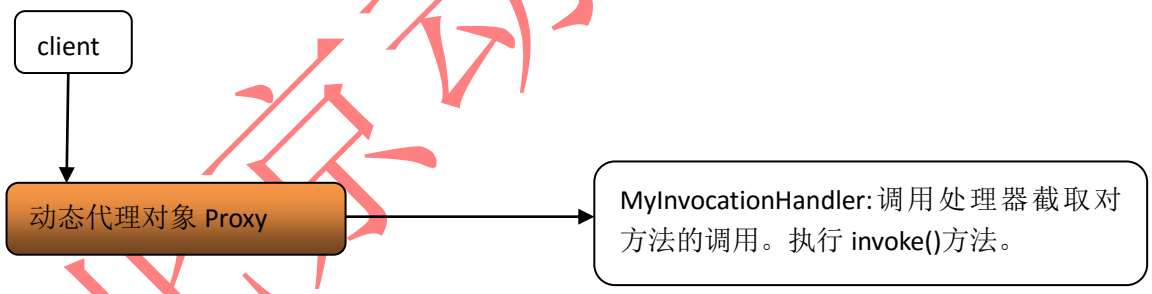
调用处理程序是实现了 InvocationHandler 的类，在 invoke 方法中增加业务功能。还需要创建有参构造，参数是目标对象。为的是完成对目标对象的方法调用。

```
public class MySellHandler implements InvocationHandler {
    private Object target = null;// 目标对象
    public MySellHandler() {  }
    public MySellHandler(Object target) {
        //使用构造方法传入目标对象，给目标对象提供代理功能
        this.target = target;
    }
    @Override
    public Object invoke(Object proxy, Method method, Object[] args)
                        throws Throwable {
        //执行目标方法
        Object ret  = method.invoke(target, args);
        float price = (float) ret;
        //在目标方法执行之后， 增强功能
        ret = price + 25;
        return ret;
    }
}
```

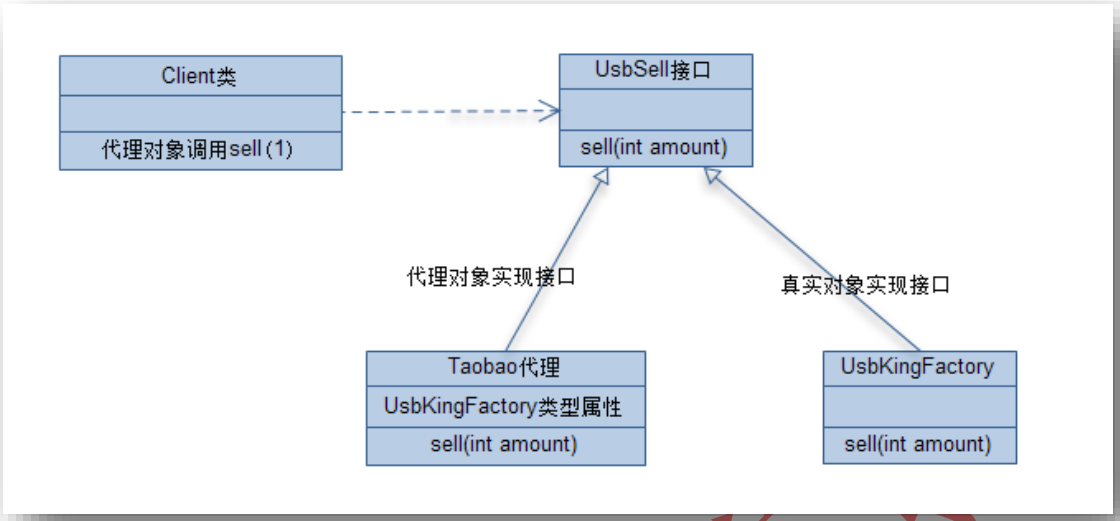
（4） 创建动态代理对象

```
public class ShopApplicationAgent2 {
    public static void main(String[] args) {
        //创建代理
        UsbKingFactory target = new UsbKingFactory();
        //创建调用处理器
        InvocationHandler handler = new MySellHandler(target);
        //创建jdk动态代理
        UsbSell taobao = (UsbSell) Proxy.newProxyInstance(
                target.getClass().getClassLoader(),
                target.getClass().getInterfaces(),
                handler);
        //通过代理对象执行业务方法，实现利润增加
        float price = taobao.sell(1);
        System.out.println("taobao 购买价格price:" + price);

    }
}
```

执行流程：



类图：



1.6.3 cgLib 代理

CGLIB(Code Generation Library)是一个开源项目。是一个强大的，高性能，高质量的 Code 生成类库，它可以在运行期扩展 Java 类与实现 Java 接口。它广泛的被许多 AOP 的框架使用，例如 Spring AOP。

使用 JDK 的 Proxy 实现代理，要求目标类与代理类实现相同的接口。若目标类不存在接口，则无法使用该方式实现。

但对于无接口的类，要为其创建动态代理，就要使用 CGLIB 来实现。CGLIB 代理的生成原理是生成目标类的子类，而子类是增强过的，这个子类对象就是代理对象。所以，使用 CGLIB 生成动态代理，要求目标类必须能够被继承，即不能是 final 的类。

cglib 经常被应用在框架中，例如 Spring ，Hibernate 等。Cglib 的代理效率高于 Jdk。对于 cglib 一般的开发中并不使用。做了一个了解就可以。