动态代理

一、掌握的程度

1. 知道什么是动态代理
    - 使用JDK的反射机制，创建对象的能力， 创建的是代理类的对象。 而不用你创建类文件。不用写java文件。
    - 动态：在程序执行时，调用JDK提供的方法才能创建代理类的对象。
    - JDK动态代理，必须有接口，目标类必须实现接口，没有接口时，需要使用 cglib 动态代理
2. 知道动态代理能做什么
    - 可以在不改变原来目标方法功能的前提下，可以在代理中增强自己的功能代码。
3. 后面会讲myBatis、Spring时会用到动态代理

二、什么是代理

代购，中介，商家等等....

比如有一家美国的大学，可以对全世界招生。留学中介（代理）

1. 代理特点：
    - 中介和代理他们要做的事情是一致的：招生。
    - 中介是学校代理， 学校是目标。
    - 家长 --- 中介（学校介绍，办入学手续）---- 美国学校。
    - 中介是代理，不能白干活，需要收取费用。
    - 代理不让你访问到目标。
2. 为什么要去找中介?
    - 中介是专业的，方便
    - 家长现在不能自己去找学校。家长没有能力访问学校。或者美国学校不接收个人来访。

卖东西都是商家卖，商家是某个商品的代理，你个人买东西，肯定不会让你接触到厂家的。

三、代理模式

- 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。
- 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。
- 实际的例子： 登录，注册有验证码， 验证码是手机短信。

  中国移动， 联通能发短信。

  中国移动， 联通能有子公司，或者关联公司，他们面向社会提供短信的发送功能

  张三项目发送短信 ----> 子公司，或者关联公司 -----> 中国移动， 联通

四、代理模式的作用

1. 功能增强：在你原有的功能上，增加了额外的功能。新增加的功能，叫做功能增强。
2. 控制访问：代理类不让你访问目标，例如商家不让用户访问厂家。

五、代理模式的分类

1. 静态代理
2. 动态代理

六、实现代理的方式

6.1 静态代理：

- 代理类是自己手工实现的，自己创建一个java类，表示代理类。
- 同时你所要代理的目标类是确定的。
- 特点：a. 实现简单 b.容易理解。

模拟一个用户购买u盘的行为。
用户：是客户端类
商家：代理，代理某个品牌的u盘。
厂家：目标类。

三者的关系： 用户（客户端）---> 商家（代理）---> 厂家（目标）
商家和厂家都是卖u盘的，他们完成的功能是一致的，都是卖u盘。

实现步骤：

1. 创建一个接口，定义卖u盘的方法，表示厂家和商家做的事情
2. 创建厂家类，实现1步骤的接口

3. 创建商家，就是代理，也需要实现1步骤中的接口
4. 创建客户端类，调用商家的方法买一个u盘。

```
//接口
package com.example.proxy.service;
//表示功能的，厂家和商家都要完成的功能
public interface UsbSell {
    //定义方法 参数 amount：表示一次购买的数量，暂时不用
    //返回值表示一个U盘的价格
    float sell(int amount);

    //可以定义多个其他的方法
}
```

```
//目标类 厂家类
package com.example.proxy.factory;
import com.example.proxy.service.UsbSell;
//目标类：金士顿厂家，不接受用户的单独购买
public class UsbKingFactory implements UsbSell {
    @Override
    public float sell(int amount) {
        System.out.println("目标类中的方法调用,UsbKingFactory中的sell ");
        //一个128G的U盘是85元
        //后期根据amount，可以实现不同的价格，例如10000个单价是80，50000个是75
        return 85.0f;
    }
}
```

```
//代理类 商家
package com.example.proxy.sahngjia;
import com.example.proxy.factory.UsbKingFactory;
import com.example.proxy.service.UsbSell;
//代理类：TaoBao是一个商家，代理金士顿U盘的销售
public class TaoBao implements UsbSell {
    //声明 商家代理的厂家具体是谁
    private UsbKingFactory factory = new UsbKingFactory();
    @Override
    public float sell(int amount) {
        //实现销售U盘功能
        //向厂家发送订单，告诉厂家，我买了U盘，厂家发货
        float price = factory.sell(amount);//厂家的价格
        //商家 需要加价，也就是代理要增加价格
        price += 25;
        //在目标类的方法调用后，你做的其它功能，都是增强的意思。
        System.out.println("淘宝商家，给你返一个优惠券，或者红包");
        //增加的价格
        return price;
    }
}

package com.example.proxy.sahngjia;
import com.example.proxy.factory.UsbKingFactory;
import com.example.proxy.service.UsbSell;
//代理类：WeiShang是一个商家，也代理金士顿U盘的销售
public class WeiShang implements UsbSell {
    //代理的是金士顿，定义目标厂家类
    private UsbKingFactory factory = new UsbKingFactory();
    @Override
    public float sell(int amount) {
        //实现目标方法
        float price = factory.sell(amount);//厂家的价格
        //只增加1元
        price += 1;
        //增加的价格
        return price;
    }
}
```

```
//用户客户端类
package com.example.proxy;
import com.example.proxy.sahngjia.TaoBao;
public class ShopMan {
    public static void main(String[] args) {
        //创建代理的商家TaoBao对象
        TaoBao taoBao = new TaoBao();
        //通过代理类，实现购买u盘，增加了优惠券，红包等等
        float price = taoBao.sell(1);
        System.out.println("通过淘宝的商家，购买U盘单价：" + price);
    }
}
/*
```

```
目标类中的方法调用,UsbKingFactory中的sell
淘宝商家，给你返一个优惠券，或者红包
通过淘宝的商家，购买U盘单价：110.0
*/
```

- 代理类完成的功能：
  1. 目标类中方法的调用
  2. 功能增强
- 静态代理缺点
  - 当你的项目中，目标类和代理类数量很多的时候，有以下的缺点：
  - 当目标类增加了，代理类可能也需要成倍的增加。代理类数量过多。
  - 当你的接口中功能增加了，或者修改了，会影响众多的实现类，厂家类，代理都需要修改。影响比较多。

6.2 动态代理

- 在静态代理中目标类很多时候，可以使用动态代理，避免静态代理的缺点。
- 动态代理中目标类即使很多：
  - 代理类数量可以很少
  - 当你修改了接口中的方法时，不会影响代理类。
- 动态代理：
  - 在程序执行过程中，使用JDK的反射机制，创建代理类对象，并动态的指定要代理目标类。换句话说：动态代理是一种创建 java 对象的能力，让你不用创建TaoBao类，就能创建代理类对象。
  - 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。
  - 特点：不用创建类文件，代理的目标是活动的，可设置的；1.不用创建代理类； 2.可以给不同的目标随时创建代理

- 动态代理的实现
  1. JDK动态代理（理解）
     - 使用java反射包中的类和接口实现动态代理的功能。
     - 反射包 java.lang.reflect， 里面有三个类：InvocationHandler, Method, Proxy
  2. cglib动态代理（了解）
     - cglib是第三方的工具库，创建代理对象。
     - cglib的原理是继承，cglib通过继承目标类，创建它的子类，在子类中重写父类中同名的方法，实现功能的修改。
     - 因为cglib是继承，重写方法，所以要求目标类不能是final的， 方法也不能是final的。cglib的要求目标类比较宽松，只要能继承就可以了。cglib在很多的框架中使用，比如mybatis，spring框架中都有使用。

- JDK 动态代理：
  1. 反射，Method类，表示方法。类中的方法；通过Method可以执行某个方法。

```java
public interface HelloService {
    //根 name 打招呼
    public void sayHello(String name);
}

public class HelloServiceImpl implements HelloService {
    @Override
    public void sayHello(String name) {
        System.out.println("Hello " + name);
    }
}

public class HelloServiceImpl2 implements HelloService {
    @Override
    public void sayHello(String name) {
        System.out.println("==========Hello " + name);
    }
}

public class TestApp {
    public static void main(String[] args) throws NoSuchMethodException,
InvocationTargetException, IllegalAccessException {
        HelloService service = new HelloServiceImpl();
        service.sayHello("张三");

        //使用反射机制执行sayHello方法，核心 Method 类中的方法
        HelloService tarGet = new HelloServiceImpl();
        HelloService tarGet2 = new HelloServiceImpl2();
        //获取sayHello名称对于的Method类对象
        Method m = HelloService.class.getMethod("sayHello", String.class);
        /*
        * invoke是Method类中的一个方法，表示执行方法的调用
        * 参数:
        *   1.Object  表示对象的，要执行这个对象的方法
```

```
         *   2.Object... args, 方法执行时的参数值
         * 返回值:
         *   Object:方法执行后的返回值
         * */
        //表达的意思就是 执行tarGet对象的sayHello()方法，参数是"李四"
        Object ret = m.invoke(tarGet, "李四");
        Object ret1 = m.invoke(tarGet2, "王五");
    }
}
/*
Hello 张三
Hello 李四
==========Hello 王五
*/
```

2. jdk动态代理的实现

反射包 java.lang.reflect , 里面有三个类： InvocationHandler，Method，Proxy

1、InvocationHandler 接口(调用处理器)

- 就一个方法 invoke()
- invoke()：表示代理对象要执行的功能代码。你的代理类要完成的功能就写在invoke()方法中。
- 代理类完成的功能：1.调用目标方法，执行目标方法的功能；2.功能增强，在目标方法调用时，增加功能。

```
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable;
/*
参数:
    proxy：jdk创建的代理对象，无需赋值。
    method：目标类中的方法，jdk提供method对象的
    args：目标类中方法的参数，jdk提供的。
*/
```

- 怎么用:

  1.创建类实现接口 InvocationHandler
  2.重写invoke() 方法， 把原来静态代理中代理类要完成的功能，写在这。

2、Method类:

- 表示方法的，确切的说就是目标类中的方法。
- 作用：通过Method可以执行某个目标类的方法，Method.invoke()

```
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

Object ret = method.invoke(service2, "李四");
```

- 说明:

  method.invoke()：就是用来执行目标方法的，等同于静态代理中的

```
//向厂家发送订单，告诉厂家，我买了u盘，厂家发货
float price = factory.sell(amount); //厂家的价格。
```

3、Proxy类:

- 核心的对象，创建代理对象。
- 之前创建对象都是 new 类的构造方法()
- 现在我们是使用Proxy类的方法，代替new的使用。

```
//方法：静态方法 newProxyInstance()
//作用是：创建代理对象，等同于静态代理中的TaoBao taoBao = new TaoBao();
public static Object newProxyInstance(ClassLoader loader,
                                      Class<?>[] interfaces,
                                      InvocationHandler h)
                                    throws IllegalArgumentException

参数:
1. ClassLoader loader 类加载器，负责向内存中加载对象的。使用反射获取对象的ClassLoader
    类a, a.getCalss().getClassLoader(), 目标对象的类加载器
2. Class<?>[] interfaces：接口，目标对象实现的接口，也是反射获取的。
3. InvocationHandler h：我们自己写的，代理类要完成的功能。

返回值：就是代理对象
```

3. 实现动态代理的步骤：
    - 创建接口，定义目标类要完成的功能
    - 创建目标类实现接口
    - 创建InvocationHandler接口的实现类，在invoke方法中完成代理类的功能
        1. 调用目标方法、
        2. 增强功能
    - 使用Proxy类的静态方法，创建代理对象。并把返回值转为接口类型。

```java
//接口
//表示功能的，厂家和商家都要完成的功能
public interface UsbSell {
    //定义方法 参数 amount：表示一次购买的数量，暂时不用
    //返回值表示一个U盘的价格
    float sell(int amount);

    //可以定义多个其他的方法
}
```

```java
//目标类实现接口
//目标类：金士顿厂家，不接受用户的单独购买
public class UsbKingFactory implements UsbSell {
    @Override
    public float sell(int amount) {
        System.out.println("目标类中的方法调用,UsbKingFactory中的sell ");
        //一个128G的U盘是85元
        //后期根据amount，可以实现不同的价格，例如10000个单价是80，50000个是75
        return 85.0f;
    }
}
```

```java
//必须实现InvocationHandler接口，完成代理类要做的功能(1.调用目标方法；2.增强功能)
public class MySellHandler implements InvocationHandler {
    private Object target = null;

    public MySellHandler(Object target) {
        this.target = target;
    }

    //动态代理：目标对象是活动的，不是固定的，需要传入进来
    //传入是谁，就给谁创建代理
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        Object res = null;
        //向厂家发送订单，告诉厂家，我买了U盘，厂家发货
        res = method.invoke(target, args);//厂家的价格
        //商家 需要加价，也就是代理要增加价格
        if(res != null){
            Float price = (Float) res;
            price += 25;
            res = price;
        }
        //在目标类的方法调用后，你做的其它功能，都是增强的意思。
        System.out.println("淘宝商家，给你返一个优惠券，或者红包");
        //增加的价格
        return res;
    }
}
```

```java
public class MainShop {
    public static void main(String[] args) {
        //创建代理对象，使用Proxy
        //1.创建目标对象
        UsbSell factory = new UsbKingFactory();
        //2.创建InvocationHandler对象
        InvocationHandler handler = new MySellHandler(factory);
        //3.创建代理对象
        UsbSell proxy = (UsbSell) Proxy.newProxyInstance(factory.getClass().getClassLoader(),
factory.getClass().getInterfaces(), handler);
        //4.通过代理执行方法
        float price = proxy.sell(1);
        System.out.println("通过动态代理对象，调用方法：" + price);
    }
}
```