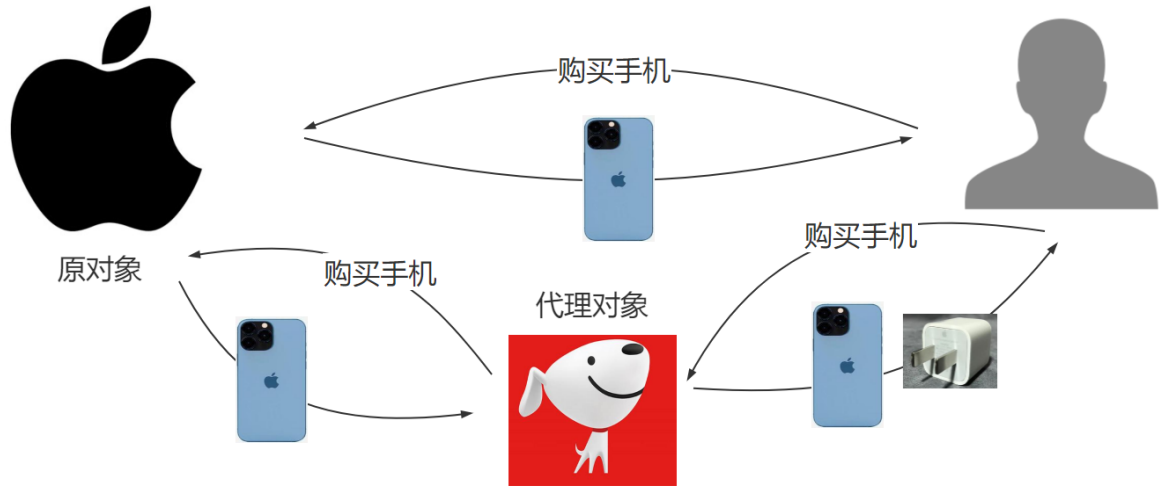
动态代理_代理模式简介



代理模式是23种设计模式之一。设计模式是前人总结的，在软件开发过程遇到常用问题的解决方案，常见的设计模式有单例模式、工厂模式、适配器模式等等。

代理模式的作用是在**不修改原对象的基础上增强该对象的方法。**比如官方购买苹果手机不赠送充电头，此时京东平台作为苹果的代理商，可以在代理销售苹果手机时赠送充电头。

代理模式分为静态代理、动态代理。静态代理会生成一个代理类，动态代理不会生成代理类，直接生成代理对象。

实时学习反馈

1. 关于动态代理和静态代理，说法正确的是?

A 动态代理和静态代理都会生成代理类。

B 动态代理和静态代理都不会生成代理类。

C 动态代理会生成代理类，静态代理不会生成代理类。

D 动态代理不会生成代理类，静态代理会生成代理类。

2. 代理模式的作用是?

A 在不修改原对象的基础上增强该对象的方法。

B 在一个系统只生成一个对象。

C 在系统中减少对象的生成。

D 使得原本接口不兼容的类能一起工作。

答案

1=>D 2=>A

动态代理_JDK动态代理

JDK动态代理是针对接口进行代理，所以我们要写被代理的接口和该接口的实现类。

```
// 被代理接口
public interface Apple {
    String sell(double price);//卖产品
    void repair();// 维修
}

// 被代理接口的实现类
public class AppleImpl implements Apple{
    @Override
    public String sell(double price) {
        System.out.println("产品卖了"+price+"元");
        return "iphone13";
    }

    @Override
    public void repair() {
        System.out.println("苹果售后维修");
    }
}

// 代理方式类，定义被代理方法的增强方式
// 该类实现InvocationHandler接口，重写invoke方法，定义方法的增强方式
public class ShoppingProxy implements InvocationHandler {
    private Apple apple;// 被代理对象
    public ShoppingProxy(Apple apple) {
```

```
        this.apple = apple;
    }

    /**
     * 定义原方法的增强方式
     * @param proxy 被代理对象
     * @param method 被代理对象调用的方法
     * @param args 被代理对象调用的方法时，传入的参数
     * @return 方法的返回值
     * @throws Throwable
     */
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
        String name = method.getName(); //被代理对象执行的方法名
        if("sell".equals(name)){
            double price = (double)args[0]*0.9; //增强参数
            Object result = method.invoke(apple, price); // 执行方法
            return result + "和充电头"; // 增强返回值
        }else if("repair".equals(name)){
            System.out.println("专属客服为您服务！"); // 增强方法流程
            return method.invoke(apple,args);
        }else{
            return method.invoke(apple,args); // 什么都不增强
        }
    }
}

public class Test {
    public static void main(String[] args) {
        // 被代理对象
        Apple apple = new AppleImpl();
        // 代理方式对象
        ShoppingProxy shoppingProxy = new ShoppingProxy(apple);
        // 生成代理对象
        Apple appleJD = (Apple) Proxy.newProxyInstance(
                apple.getClass().getClassLoader(), // 类加载器
                apple.getClass().getInterfaces(),//被代理接口
                shoppingProxy //代理方式对象
        );
        // 执行增强后的方法
        String sell = appleJD.sell(6000);
        System.out.println(sell);

        appleJD.repair();
    }
}
```

实时学习反馈

1. 关于JDK实现动态代理，说法不正确的是?

A 是针对接口进行代理。

B 是针对类进行代理。

C 要写被代理的接口和该接口的实现类。

D 不会生成代理类，只会生成代理对象。

答案

1=>B

动态代理_CGLib动态代理

CGLib动态代理简化了JDK动态代理的写法，JDK是针对接口代理，而CGLib是针对类代理。

```
<!-- 引入cglib依赖 -->
<dependencies>
    <dependency>
        <groupId>cglib</groupId>
        <artifactId>cglib</artifactId>
        <version>3.3.0</version>
    </dependency>
</dependencies>

// 被代理类
public class Apple{
    public String sell(double price) {
        System.out.println("产品卖了"+price+"元");
        return "iphone13";
    }
    public void repair() {
        System.out.println("苹果售后维修");
    }
}

// 代理方式类，实现MethodInterceptor接口，重写intercept方法
public class ShoppingProxy implements MethodInterceptor {
    private Apple apple; // 被代理对象
```

```
    public ShoppingProxy(Apple apple) {
        this.apple = apple;
    }

    /**
     * 定义原方法的增强方式
     * @param o 被代理对象
     * @param method 被代理对象调用的方法
     * @param objects 被代理对象调用的方法时，传入的参数
     * @param methodProxy 底层生成的代理类的引用
     * @return 方法的返回值
     * @throws Throwable
     */
    @Override
    public Object intercept(Object o, Method method, Object[] objects,
MethodProxy methodProxy) throws Throwable {
        String name = method.getName();
        if("sell".equals(name)){
            double price = (double)objects[0]*0.8;
            Object result = method.invoke(apple, price);
            return result+"和数据线";
        }else if("repair".equals(name)){
            System.out.println("专属客服为您服务！");
            return method.invoke(apple,objects);
        }else{
            return method.invoke(apple,objects);
        }
    }
}

public class Test {
    public static void main(String[] args) {
        // 被代理对象
        Apple apple = new Apple();
        // 代理方式
        ShoppingProxy shoppingProxy = new ShoppingProxy(apple);
        // 生成代理对象
        Apple appleTB = (Apple) Enhancer.create(Apple.class, shoppingProxy);

        // 执行增强后的方法
        String sell = appleTB.sell(9000);
        System.out.println(sell);
        appleTB.repair();
    }
}
```

实时学习反馈

1. 关于CGLib实现动态代理，说法正确的是?

A 是针对类进行代理。

B 是针对接口进行代理。

C 不需要额外引入依赖。

D 会生成代理类。

答案

1=>A