链滴

代理模式

作者：helly
原文链接：https://ld246.com/article/1510316852839
来源网站：链滴

代理模式（Proxy Pattern）

一、静态代理 - 代理对象和被代理对象一开始就确定，它们继承同一个抽象类或实现同一个接口

代理类一开始就写好

```
public interface ForSale() {
  public int price();
}
public class HouseA implements ForSale {
  public int price() {
    return 1000000;
  }
}
public class HouseProxy implements ForSale {
  private ForSale house;
  public HouseProxy(ForSale house) {
    this.house = house;
  }
  public int price() {
    return before() + houseA.price() + after();
  }
  public int before() {
    return 10000;
  }
  public int after() {
    return 20000;
  }
}
```

每一个代理类只能为一个接口服务，开发中必然会产生过多的代理。

所有的代理操作除了调用的方法不一样之外，其他的操作都一样时，会产生重复代码。

二、动态代理

1 JDK动态代理（接口的代理）

```
public interface Hello {
  public void say();
}
public class HelloImpl implements Hello {
  public void say() {
    System.out.println("hello world!");
  }
}

public class MyInvocationHandler implements  InvocationHandler {
  Hello target;
  public MyInvocationHandler(Hello target) {
    this.target = target;
  }
  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    ...
    method.invoke(target, args);
```

```
    ...
    return null;
  }
}

public void Test {
  public static void main(String[] args) {
     // 方法1
    InvocationHandler mih = new MyInvocationHandler(new HelloImp());
    Class proxyClass = Proxy.getProxyClass(Test.class.getClassLoader(), Hello.class);
    Constructor cons = proxyClass.getConstructor(InvocationHandler.class); // 基于组合的耦合
式
    Hello helloProxy = (Hello) cons.getInstance(mih);
    helloProxy.say();

    // 方法2
    InvocationHandler mih = new MyInvocationHandler(new HelloImp());
    Hello helloProxy = (Hello) Proxy.getInstance(Test.class.getClassLoader, new Class[]{Hello.cla
s}, mih);
    helloProxy.say();
  }
}
```

JDK的动态代理机制只能代理实现了接口的类，而不能实现接口的类就不能实现JDK的动态代理。

2 CGLib动态代理（类的代理）

```
public class Hello {
  public void say() {
    System.out.println("hello world!");
  }
}

public class Interceptor implements MethodInterceptor {
    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy) thr
ws Throwable {
        System.out.println("I am intercept begin");
        proxy.invokeSuper(obj, args);
        System.out.println("I am intercept end");
        return null;
    }
}

public class Test {
  public static void main(String[] args) {
        Enhancer eh = new Enhancer();
        eh.setSuperclass(Hello.class);
        eh.setCallback(new Interceptor());
        Hello hello = (Hello) eh.create();
        hello.say();
    }
}
```

对指定的目标类生成一个子类，并覆盖其中方法实现增强，但因为采用的是继承，所以不能对final修的类进行代理。

参考：

java动态代理（JDK和cglib）