链滴

java 代理模式与 JDK 代理

作者：wthfeng

原文链接：https://ld246.com/article/1495945307833

来源网站：链滴

前言

代理模式是很常用的设计模式之一，一般可分为静态代理和动态代理两类。java利用反射也对动态代提供了支持。今天我们就来学习学习。

1. 定义

给某一个对象提供一个代理，并由代理对象控制对原对象的引用，称为代理模式。它是一种对象结构模式。

即可理解为，某个对象实例（记为Subject）不方便直接引用，我们就提供一个代理实例(记为Proxy)让这个代理实例去调用实例对象。我们直接与Proxy 打交道，由Proxy 负责与Subject 沟通。

这样说来，Proxy 相当于一个中间人的角色，负责**我们**与**实际对象**的沟通。

2. 情景示例

我们通过代码来描述这种模式。

假设一种情景，图片的加载工作。大图片加载很耗时，我们希望在大图片加载过程中先做一些操作（如显示张小图片、或给个文字提示），等大图片加载完毕后再显示大图片。

有一点需要注意，在显示大图片前做的操作不确定，这样就不能写死在大图片加载中。让我们考虑下理模式。

需要有一个共同的接口类。

是的。代理对象（Proxy）对外要表现实际对象(Subject)的功能，所以它们应该有一个共同的接口。

```
public interface ImageHandler {

    void loadImage();
}
```

下面是真正的图片加载类

```
public class ImageHandlerImpl implements ImageHandler {
    @Override
    public void loadImage() {
        try {
            //用休眠2秒表示图片加载过程
            TimeUnit.SECONDS.sleep(2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("图片加载完成。");
    }
}
```

该设计我们的代理类了。我们知道，代理类是要能执行我们的真实对象方法的。怎样执行？最简单的式当然是把真实对象的实例传给代理。

```
public class ImageHandlerProxy implements ImageHandler{
    private ImageHandler imageHandler;

    //将真实对象通过构造器传过来
    public ImageHandlerProxy(ImageHandler imageHandler) {
        this.imageHandler = imageHandler;
    }

    @Override
    public void loadImage() {
        //代理做一些预处理
        System.out.println("请等待，正在加载图片");
//        System.out.println("加载小图片");

        //调用真实对象方法
        imageHandler.loadImage();

        //可以做一些收尾工作
    }
}
```

这样代理就完成。测试一下

```
    @Test
    public void test(){
        ImageHandler handler = new ImageHandlerImpl();
        ImageHandlerProxy proxy = new ImageHandlerProxy(handler);

        proxy.loadImage();

    }
```

结果先出现提示信息，过一会后图片加载完毕

```
请等待，正在加载图片
图片加载完成。
```

3. 动态代理

在了解动态代理前，我们先思考一下，上面的例子有什么问题?

1. 动态代理只是实现对一种对象的代理，如上例 ImageHandler。这样导致复用性很差，比如我想视频也实现这样的代理，还要再写一个视频代理类。

2. 代理方法也写死在接口中（如上例的 loadImage()方法），这样我再代理一个方法就要在代理对再实现一遍。

当然，这些讨论是建立在你有这些需求的基础上。比如你就只需要一个图片加载的代理，静态代理也足够了。

我们需要一种方法来解决上面的问题。上面例子中，代理类是事先写好的，运行时早已确定。而我们望的是，能够在运行时动态生成某个类，这样就不必为每个真正需代理的对象都写一个代理类了。

JDK直接实现了这种需求。我们只需要做到：

1. 写一个动态代理类实现 InvocationHandler接口。
2. 使用 Proxy 类生成代理对象实例。

下面通过代码演示一下

上例中的ImageHandler与ImageHandlerImpl 仍保留。

```
public class MyDynamicProxy implements InvocationHandler {

    private Object subject;

    //通过构造函数传入实际对象实例
    public MyDynamicProxy(Object subject) {
        this.subject = subject;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("前置工作");
        Object result = method.invoke(subject,args);
        System.out.println("收尾工作");
        return result;
    }
}
```

MyDynamicProxy 与前面的静态代理模式中ImageHandlerProxy 相似，都是传入真正对象，最后用真实对象完成。不同的是，动态代理不耦合某个接口。

生成代理对象与测试

```
    @Test
    public void testDynamic(){

        //真实对象
        ImageHandler realObject = new ImageHandlerImpl();

        //代理对象的处理器
        InvocationHandler  handler = new MyDynamicProxy(realObject);

        //生成代理对象
        ImageHandler imageProxy = (ImageHandler) Proxy.newProxyInstance(handler.getClass()
getClassLoader(),
                new Class[]{ImageHandler.class},handler);

        imageProxy.loadImage();
    }
```

这就是JDK的动态代理实例。其中最关键的莫过newProxyInstance方法

```
  public static Object newProxyInstance(ClassLoader loader,Class<?>[] interfaces,InvocationH
ndler h)
```

参数分别为类加载器、接口数组及实现了InvocationHandler接口的对象实例。newProxyInstance()

可根据这些信息创建代理对象实例。从而实现动态代理。