Java 动态代理

作者：tianma630
原文链接：https://ld246.com/article/1376437765964
来源网站：链滴

      代理模式是常用的java设计模式，他的特征是代理类与委托类有同样的口，代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后处理消息等。理类与委托类之间通常会存在关联关系，一个代理类的对象与一个委托类的对象关联，代理类的对象身并不真正实现服务，而是通过调用委托类的对象的相关方法，来提供特定的服务。按照代理的创建期，我们可以把代理分成静态代理和动态代理。这里我们主要介绍下动态代理的两种实现方式：jdk态代理和cglib动态代理。两者的区别在于，jdk动态代理是在代码层面上的处理，需要你定义特定的口，这样代理才能找到你需要被代理的类和方法；而cglib是在字节码层面上的操作，不需要你定义特的接口。下面是两者实现的简单实例：
代码下载地址：http://download.csdn.net/detail/tianma630/6002625
 一、jdk动态代理
1、定义接口

```
package jdk1;

/**
 * 定义接口
 * @author tianma630
 */
public interface Vehicle {
    public void run();
}
```

2、创建业务类，实现接口

```
package jdk1;

/**
 * 定义Car类 实现vehicle接口
 * @author tianma630
 */
public class Car implements Vehicle {

    @Override
    public void run() {
        System.out.println(&amp;quot;小汽车运行。。。&amp;quot;);
    }

}
```

3、代理类

```
package jdk1;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

/**
 * 定义vehicle的代理
 * @author tianma630
 */
public class VehicleProxy implements InvocationHandler {
```

```
    private Vehicle vehicle;

    public VehicleProxy(Vehicle vehicle){
        this.vehicle = vehicle;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println(&quot;代理前操作。。。&quot;);
        Object invoke = method.invoke(vehicle, args);
        System.out.println(&quot;代理后操作。。。&quot;);
        return invoke;
    }

}
```

4、业务工厂类

```
package jdk1;

import java.lang.reflect.Proxy;

/**
 * vehicle的工厂类，用来生产代理后的vehicle实例
 * @author lenovo
 */
public class VehicleFactory {

    public static Vehicle newInstance(Vehicle vehicle){
        return (Vehicle)Proxy.newProxyInstance(vehicle.getClass().getClassLoader(), vehicle.getCl
ss().getInterfaces(), new VehicleProxy(vehicle));
    }
}
```

5、客户端测试

```
package jdk1;

/**
 * 客户端测试
 * @author tianma630
 */
public class Client {

    public static void main(String[] args) {
        Vehicle car = VehicleFactory.newInstance(new Car());
        car.run();
    }

}
```

二、cglib动态代理
1、创建业务类

```
package cglib1;

/**
 * 定义car类
 * @author tianma630
 */
public class Car {
    public void run() {
        System.out.println(&quot;小汽车运行。。。&quot;);
    }
}
```

2、创建代理类

```
package cglib1;

import java.lang.reflect.Method;

import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

/**
 * car的代理类
 * @author tianma630
 */
public class CarProxy implements MethodInterceptor {

    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy) thr
ws Throwable {
        System.out.println(&quot;代理前操作。。。&quot;);
        Object invoke = proxy.invokeSuper(obj, args);
        System.out.println(&quot;代理后操作。。。&quot;);
        return invoke;
    }

}
```

3、创建业务工厂类

```
package cglib1;

import net.sf.cglib.proxy.Enhancer;

/**
 * car的工厂类，用来生产代理后的car类
 * @author tianma630
 */
public class CarFactory {
    public static Car newInstance(CarProxy carProxy){
        Enhancer enhancer = new Enhancer();
        enhancer.setSuperclass(Car.class);
        enhancer.setCallback(carProxy);
        return (Car) enhancer.create();
```

```
    }
}
```

4、客户端测试

```
package cglib1;

/**
 * 客户端测试
 * @author tianma630
 */
public class Client {

    public static void main(String[] args) {
        Car car = CarFactory.newInstance(new CarProxy());
        car.run();
    }

}
```