动态代理（在运行时扩展类的功能）

一、基本介绍

在程序运行期间根据需要动态创建代理类及其实例来完成具体的功能

动态代理主要分为JDK动态代理和cglib动态代理

二、JDK动态代理（基于接口的动态代理）

1、基本概念

jdk动态代理是jdk原生就支持的一种代理方式，它的实现原理，就是通过让target类和代理类实现同一接口，代理类持有target对象，来达到方法拦截的作用，这样通过接口的方式有两个弊端，一个是必须保证target类有接口，第二个是如果想要对target类的方法进行代理拦截，那么就要保证这些方法都要在接口中声明，实现上略微有点限制。

涉及的类：Proxy

提供者：JDK官方

2、 创建代理对象的要求：

被代理类最少实现一个接口，如果没有则不能使用

3、实例

必须实现接口

```
public interface IProducer {
    //销售
    public void saleProduct(float money);
    //售后
    public void afterService(float money);
}
```

```
    public static void main(String[] args) {
        //匿名内部类的调用必须声明为final
        final Producer producer = new Producer();
        /**
         * 如何创建代理对象：
         *      使用Proxy类中的newProxyInstance方法
         * newProxyInstance方法的参数：
         *      ClassLoader：类加载器
         *          它是用于加载代理对象字节码的。和被代理对象使用相同的类加载器。固定写法。
```

```
        *      Class[]：字节码数组
        *          它是用于让代理对象和被代理对象有相同方法。固定写法。
        *      InvocationHandler：用于提供增强的代码
        *          它是让我们写如何代理。我们一般都是些一个该接口的实现类，通常情况下都是匿名内部类，但不是必须的。
        *          此接口的实现类都是谁用谁写。
        */
       IProducer proxyProducer = (IProducer) Proxy.newProxyInstance(producer.getClass().getClassLoader(),
               producer.getClass().getInterfaces(),
               new InvocationHandler() {
                  /**
                   * 作用：执行被代理对象的任何接口方法都会经过该方法
                   * 方法参数的含义
                   * @param proxy   代理对象的引用
                   * @param method  当前执行的方法
                   * @param args    当前执行方法所需的参数
                   * @return        和被代理对象方法有相同的返回值
                   * @throws Throwable
                   */
                  @Override
                  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                     //可以增加一些增强的代码
                     。。。。。。。。。。
                     returnValue = method.invoke(producer, money*0.8f);
                     return returnValue;
                  }
               });

    }
}
```

三、基于子类的动态代理（基于子类的动态代理）

1、基本概念

Cglib是一个优秀的动态代理框架，它的底层使用ASM在内存中动态的生成被代理类的子类，使用CGLIB即使代理类没有实现任何接口也可以实现动态代理功能。CGLIB具有简单易用，它的运行速度要远远快于JDK的Proxy动态代理

涉及的类：Enhancer

提供者：第三方cglib库

2、创建代理对象的要求：

被代理类不能是最终类

3、实例

不必实现接口

```
public class Client {
    public static void main(String[] args) {
        //匿名内部类的调用必须声明为final
        final Producer producer = new Producer();

        /**
         * 动态代理：
         *  特点：字节码随用随创建，随用随加载
         *  作用：不修改源码的基础上对方法增强
         *  如何创建代理对象：
         *      使用Enhancer类中的create方法
         *  create方法的参数：
         *      Class：字节码
         *          它是用于指定被代理对象的字节码。
         *
         *      Callback：用于提供增强的代码
         *          它是让我们写如何代理。我们一般都是些一个该接口的实现类，通常情况下都是匿名内部类，但不是必须的。
         *          此接口的实现类都是谁用谁写。
         *          我们一般写的都是该接口的子接口实现类：MethodInterceptor
         */
        Producer cglibProducer = (Producer)Enhancer.create(producer.getClass(),
            new MethodInterceptor() {

            /**
             * 执行被代理对象的任何方法都会经过该方法
             * @param proxy
             * @param method
             * @param args
             *    以上三个参数和基于接口的动态代理中invoke方法的参数是一样的
             * @param methodProxy ： 当前执行方法的代理对象
             * @return
             * @throws Throwable
             */
            @Override
            public Object intercept(Object proxy, Method method, Object[] args,
MethodProxy methodProxy) throws Throwable {
                //提供增强的代码
```

```
            Object returnValue = null;

            //1.获取方法执行的参数
            Float money = (Float)args[0];
            //2.判断当前方法是不是销售
            if("saleProduct".equals(method.getName())) {
                returnValue = method.invoke(producer, money*0.8f);
            }
            return returnValue;
        }
    });
    cglibProducer.saleProduct(12000f);
  }
}
```