JDK动态代理

```
public static UserService createService(){

    //1.目标类
    final UserService userService = new UserServiceImpl();
    //2.切面类
    final MyAspect myAspect = new MyAspect();
    /*3.代理类：将目标类（切入点）和切入面（通知）结合-->切面
     *  Proxy.newProxyInstance
     *      参数1：loader，类加载器，动态代理类，运行时创建任何类都需要加载器加载到内存
     *              一般情况下：当前类.class.getClassLoader();
     *                      目标类实例.getClass().get...
     *      参数2：interfaces 代理类需要实现的所有接口
     *          方式1：目标类实例.getClass().getInterfaces();       注意：只能获得自己的接口，不能获得父元素接口
     *          方式2：new Class[]{UserService.class}
     *              例如:jdbc 驱动-->DriverManager 获得接口Connection
     *      参数3：InvocationHandler 处理类，接口，必须进行实现类，一般采用匿名内部
     *          提供invoke方法，代理类的每一个方法执行时，都将调用一次invoke
     *              参数31：Object proxy：代理对象
     *              参数32：Method method：代理对象当前执行的方法的描述对象（反射）
     *                  执行方法名：method.getName()
     *                  执行方法：method.invoke(对象,实际参数)
     *              参数33：Object[] args：方法实际参数
     *
     */
    UserService proxService = (UserService)Proxy.newProxyInstance(
            MyBeanFactory.class.getClassLoader(),
            userService.getClass().getInterfaces(),
            new InvocationHandler(){

                public Object invoke(Object proxy, Method method,Object[] args) throws Throwable {

                    //前执行
                    myAspect.before();

                    //执行目标类方法
                    method.invoke(userService, args);

                    //后执行
                    myAspect.after();
```

应用：

1.声明目标类对象 userService

和切面类对象 myAspect

2.创建代理类对象

代理类：将目标类（切入点）和切入面结合（通知）-->切面

UserService proxService = (UserService)Proxy.newProxyInstance(1,2,3)

Proxy.newProxyInstance(1,2,3)

参数1：loader，类加载器，动态代理类，运行时创建任何类都需要加载器加载到内存

一般情况：当前类.class.getClassLoader();

目标类实例.getClass().get...

参数2：interfaces 代理类需要实现的所有接口

方式1：目标类实例.getClass().getInterfaces(); 只能获得自己的接口，不能获得父元素接口

方式2：new Class[]{UserService.class}

参数3：InvocationHandler 处理类，接口，必须进行实现，一般采用匿名内部（这就是final的原因）

参数31：Object proxy :代理对象

参数32：Method method:代理对象当前执行的方法的描述（反射）

执行方法名:method.getName()

执行方法：method.invoke(对象,实际参数)

参数33：Object[] args:方法实际参数

```
new InvocationHandler(){
        public Object invoke(Object proxy,
                Method method,
                Object[] args) throws Throwable {
    //前执行
    myAspect.before();
    //执行目标类方法
    method.invoke(userService, args);
    //后执行
```

```
                myAspect.after();
                return null;
            }
        }
```

一个实例

****

```
public class MyBeanFactory {
 public static UserService createService(){
  final UserService userService = new UserServiceImpl();
  final MyAspect myAspect = new MyAspect();
  UserService proxService = (UserService)Proxy.newProxyInstance(
    MyBeanFactory.class.getClassLoader(),
    userService.getClass().getInterfaces(),
    new InvocationHandler(){
     public Object invoke(Object proxy, Method method,Object[] args) throws Throwable {
      myAspect.before();
      method.invoke(userService, args);
      myAspect.after();
      return null;
     }
    });
  return proxService;
 }
}
```