**JDK的动态代理实现**

jdk的动态代理，依赖的是反射包下的InvocationHandler接口，我们的代理类实现InvocationHandler，重写invoke()方法，每当我们的代理类调用方法时，都会默认先经过invoke()方法。

*UserService接口*

public interface UserService {

public String getName(int id);

public Integer getAge(int id);

}

*接口的实现类UserServiceImpl*

public class UserServiceImpl implements UserService {

public String getName(int id) {

System.out.println("------getName------");

return "Tom";

}

public Integer getAge(int id) {

System.out.println("------getAge------");

return 10;

}

}

*UserInvocationHandler.java*

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

public class UserInvocationHandler implements InvocationHandler {

private Object target;

UserInvocationHandler() {

super();

}

UserInvocationHandler(Object target) {

super();

this.target = target;

}

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

if("getName".equals(method.getName())){

System.out.println("++++++before " + method.getName() + "++++++");

Object result = method.invoke(target, args);

System.out.println("++++++after " + method.getName() + "++++++");

return result;

}else{

Object result = method.invoke(target, args);

return result;

}

}

}

*测试类TestUserInvocationHandler.java*

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Proxy;

public class TestUserInvocationHandler {

public static void main(String[] args) {

UserService userService = new UserServiceImpl();

InvocationHandler invocationHandler = new UserInvocationHandler(userService);

UserService userServiceProxy = (UserService) Proxy.newProxyInstance(

userService.getClass().getClassLoader(),

userService.getClass().getInterfaces(),

invocationHandler);

System.out.println(userServiceProxy.getName(1));

System.out.println(userServiceProxy.getAge(1));

}

}

运行结果：