JDK动态代理实现

01、概述

JDK动态代理实现的原理是基于InvocationHandler接口，利用反射生成一个代理接口的匿名类，然后调用invoke方法。

02、核心要素

- 代理类 实现 InvocationHandler 接口
  - 覆盖invoke方法（通知）
- 代理对象
- 业务接口
- 业务接口的实现类（目标对象）

mybatis 为例

- 代理类：MapperProxy
- 代理对象 Proxy.newProxyInstance(this.mapperInterface.getClassLoader(), new Class[]{this.mapperInterface}, mapperProxy);
- 业务接口 -- UserMapper
- 业务接口的实现类（目标对象）----- sqlSession
  - mybatis
    - ibatis ---- sqlSession---curd
      - List userList = sqlSession.selectList("namspace + id",参数)

- mapper ----jdk动态（拿标准和拿规范，拿接口的规范，方法名组装一个与xml对应关系） --- sqlSession ---执行curd

```
UserMapper userProxy = sqlSessoion.getMapper(UserMapper.class);
userProxy.listUser();

userProxy :代理对象
- namspace : userProxy =  package + classname
- listUser : id = listuser

namespace + id = package+ classname + listuser
```

mybatis的mapper通过 jdk动态代理解决："namspace + id" 获取问题

```
List<User>  userList = sqlSession.selectList("namspace + id",参数)
```

第一步：定义业务接口和实现类

userservice接口

```
package com.kuangstudy.second;

//用户管理接口
public interface IUserService {

    // 1：查询用户
    void listUser();

    // 2：新增用户抽象方法
```

```
    void saveUser(String userName, String password);

    // 3: 修改用户
    void updateUser(String userName, String password);

    //4: 删除用户抽象方法
    void delUser(Long userId);
}
```

userservice实现类

```
package com.kuangstudy.second;

import lombok.extern.slf4j.Slf4j;

//用户管理接口
@Slf4j
public class UserServiceImpl implements IUserService {

    // 1: 查询用户
    public void listUser() {
        log.info("-----listUser------");
    }

    // 2: 新增用户抽象方法
    public void saveUser(String userName, String password) {
        log.info("-----saveUser------{},{}", userName,
password);
    }

    // 3: 修改用户
    public void updateUser(String userName, String password)
{
        log.info("-----updateUser------{},{}", userName,
password);
    }

    //4: 删除用户抽象方法
```

```
    public void delUser(Long userId) {
        log.info("-----delUser---{}---", userId);
    }
}
```

测试运行：

```
package com.kuangstudy.second;

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
 * 记得关注和三连哦!
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/22 20:48
 */
public class MainTest {

    public static void main(String[] args) {
        // 1 : 多态创建对象
        IUserService userService = new UserServiceImpl();
        // 2: 对象执行方法
        userService.listUser();
        userService.saveUser("yykk", "2212");
        userService.updateUser("yykk", "2212");
        userService.delUser(1L);
    }

}

```

第二步：创建代理类

```
package com.kuangstudy.second.jdkproxy;

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

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
 * 记得关注和三连哦!
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/22 20:50
 */
public class UserProxy implements InvocationHandler {

    // 目标执行对象 : 1: 生成代理对象 2：代理增强执行完毕最后还得自己的方法自己去执行
    private Object target;

    // 1：构造的作用：给具体的目标对象进行初始化
    public UserProxy(Object target){
        this.target = target;
    }

    //此方法就是把：目标对象转换成代理对象的方法
    // 2：代理对象创建的方法的作用：1：创建代理对象，建立目标对象和代理类关系， 2：给具体的目标对象进行初始化
    public Object getProxyObject(Object targetObject) {
        //为目标对象target赋值
        this.target = targetObject;
        //JDK动态代理只能针对实现了接口的类进行代理，newProxyInstance 函数所需参数就可看出
        // 参数1：目标对象的类加载器
        // 参数2：目标对象的实现的接口
```

```
33          // 参数3：切面类引用，那么this就代表当前UserProxy不就是代理类
34          // 含义是：把目标对象转和对应接口转换成代理对象返回，而代理对象执行方法进行到切面类中的invock方法。
35          return Proxy.newProxyInstance(targetObject.getClass().getClassLoader(), targetObject.getClass().getInterfaces(), this);
36      }
37
38
39      @Override
40      public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
41          // 日志处理完了
42          // target对象执行方法
43          Object invoke = method.invoke(target, args);
44          // 结果返回
45          return invoke;
46      }
47  }
48
```

第三步：测试代理

```
1  package com.kuangstudy.second.jdkproxy;
2
3  import java.lang.reflect.Proxy;
4
5  /**
6   * @author 飞哥
7   * @Title: 学相伴出品
8   * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
9   * 记得关注和三连哦!
```

```
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/22 20:48
 */
public class MainTest {

    public static void main(String[] args) {
        // 1:创建代理类实例
        UserProxy userProxy = new UserProxy();
        // 2: 获取代理对象
        IUserService userService = (IUserService) userProxy.getProxyObject(new UserServiceImpl());
        // 2: 代理对象执行方法
        userService.listUser();
        userService.saveUser("yykk", "2212");
        userService.updateUser("yykk", "2212");
        userService.delUser(1L);
    }


}

```

总结

1、核心代码，根据目标对象创建代理对象。目标对象必须要传递进来，必须要有接口。

```
return Proxy.newProxyInstance(target.getClass().getClassLoader(), target.getClass().getInterfaces(), this);
```

2、代理对象，必须用接口去接收

```
IUserService userService = (IUserService)
userProxy.getProxyObject(new UserServiceImpl());
```

3、产生代理对象的目的：其实就是去执行代理类中的invoke方法

```

    @Override
    public Object invoke(Object proxy, Method method,
Object[] args) throws Throwable {
        // 日志处理完了
        // target对象执行方法
        Object invoke = method.invoke(target, args);
        System.out.println("这里插入日志！！！！");
        // 结果返回
        return invoke;
    }
```

4：代理对象执行上面的invoke方法以后，其实它的作用就已经完毕了。所有你invoke方法中必须把具体方法的执行回归目标对象上去执行。

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

这也就是为什么我们在代理类一定要传递一个target对象。

> 一句话：代理对象执行方法，其实就让你去进入代理类的*invoke*方法，做增强，做完了就让出执行权限。