Java的三种代理模式

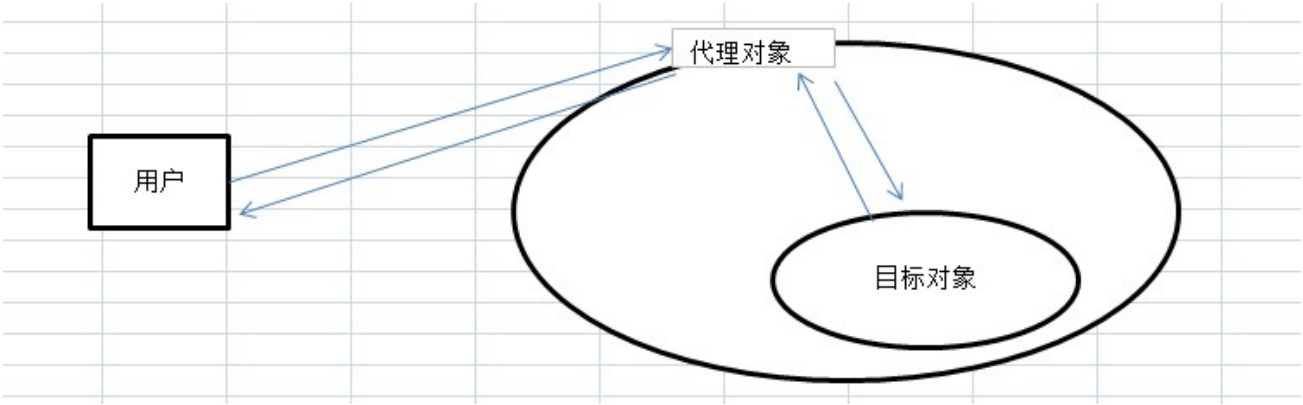
Java的三种代理模式

1.代理模式

代理(Proxy)是一种设计模式,提供了对目标对象另外的访问方式;即通过代理对象访问目标对象.这样做的好处是:可以在目标对象实现的基础上,增强额外的功能操作,即扩展目标对象的功能.
这里使用到编程中的一个思想:不要随意去修改别人已经写好的代码或者方法,如果需改修改,可以通过代理的方式来扩展该方法

举个例子来说明代理的作用:假设我们想邀请一位明星,那么并不是直接连接明星,而是联系明星的经纪人,来达到同样的目的.明星就是一个目标对象,他只要负责活动中的节目,而其他琐碎的事情就交给他的代理人(经纪人)来解决.这就是代理思想在现实中的一个例子

用图表示如下:



代理模式的关键点是:代理对象与目标对象.代理对象是对目标对象的扩展,并会调用目标对象

1.1.静态代理

静态代理在使用时,需要定义接口或者父类,被代理对象与代理对象一起实现相同的接口或者是继承相同父类.

下面举个案例来解释:
模拟保存动作,定义一个保存动作的接口:IUserDao.java,然后目标对象实现这个接口的方法UserDao.java,此时如果使用静态代理方式,就需要在代理对象(UserDaoProxy.java)中也实现IUserDao接口.调用的时候通过调用代理对象的方法来调用目标对象.
需要注意的是,代理对象与目标对象要实现相同的接口,然后通过调用相同的方法来调用目标对象的方法

代码示例:
接口:IUserDao.java

```
/**
 * 接口
 */
public interface IUserDao {

    void save();
}
```

目标对象:UserDao.java

```
/**
 * 接口实现
```

25 26 27 28 29 30 1
2 3 4 5 6 7 8

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

Java(32)
JavaWeb(12)
Struts(7)
Spring(6)
PHP(5)
python 学习笔记(4)
Hibernate(4)
django(3)
JavaScript(3)
Linux(3)
更多

随笔分类

道·理论(66)

```
 * 目标对象
 */
public class UserDao implements IUserDao {
    public void save() {
        System.out.println("----已经保存数据!----");
    }
}
```

代理对象:UserDaoProxy.java

```
/**
 * 代理对象,静态代理
 */
public class UserDaoProxy implements IUserDao{
    //接收保存目标对象
    private IUserDao target;
    public UserDaoProxy(IUserDao target){
        this.target=target;
    }

    public void save() {
        System.out.println("开始事务...");
        target.save();//执行目标对象的方法
        System.out.println("提交事务...");
    }
}
```

测试类:App.java

```
/**
 * 测试类
 */
public class App {
    public static void main(String[] args) {
        //目标对象
        UserDao target = new UserDao();

        //代理对象,把目标对象传给代理对象,建立代理关系
        UserDaoProxy proxy = new UserDaoProxy(target);

        proxy.save();//执行的是代理的方法
```


随笔档案

最新评论

1. Re:Python的hasattr() getattr() setattr() 函数使用方法详解
@豪门百里那应该用什么 不垃圾的平台?...
--阿谋

2. Re:Java的三种代理模式
cglib动态代理，执行目标对象的方法那行应该是Object returnValue = proxy.invokeSuper(obj, args);寻找了半天资料，发现原来是这么写，博主的写法会莫名的.....
--乖乖宁宝

```
    }
}
```

静态代理总结:

1.可以做到在不修改目标对象的功能前提下,对目标功能扩展.

2.缺点:

- 因为代理对象需要与目标对象实现一样的接口,所以会有很多代理类,类太多.同时,一旦接口增加方法,目标对象与代理对象都要维护.

如何解决静态代理中的缺点呢?答案是可以使用动态代理方式

1.2.动态代理

动态代理有以下特点:

1.代理对象,不需要实现接口

2.代理对象的生成,是利用JDK的API,动态的在内存中构建代理对象(需要我们指定创建代理对象/目标对象实现的接口的类型)

3.动态代理也叫做:JDK代理,接口代理

JDK中生成代理对象的API

代理类所在包:java.lang.reflect.Proxy

JDK实现代理只需要使用newProxyInstance方法,但是该方法需要接收三个参数,完整的写法是:

```
static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces,InvocationHandler h
)
```

注意该方法是在Proxy类中是静态方法,且接收的三个参数依次为:

- `ClassLoader loader,` :指定当前目标对象使用类加载器,获取加载器的方法是固定的
- `Class<?>[] interfaces,` :目标对象实现的接口的类型,使用泛型方式确认类型
- `InvocationHandler h` :事件处理,执行目标对象的方法时,会触发事件处理器的方法,会把当前执行目标对象的方法作为参数传入

代码示例:

接口类IUserDao.java以及接口实现类,目标对象UserDao是一样的,没有做修改.在这个基础上,增加一个代理工厂

类(ProxyFactory.java),将代理类写在这个地方,然后在测试类(需要使用到代理的代码)中先建立目标对象和代理对象的联系,然后代用代理对象的中同名方法

代理工厂类:ProxyFactory.java

```
/**
 * 创建动态代理对象
 * 动态代理不需要实现接口,但是需要指定接口类型
 */
public class ProxyFactory{

    //维护一个目标对象
    private Object target;
    public ProxyFactory(Object target){
        this.target=target;
    }

   //给目标对象生成代理对象
    public Object getProxyInstance(){
        return Proxy.newProxyInstance(
                target.getClass().getClassLoader(),
                target.getClass().getInterfaces(),
                new InvocationHandler() {
                    @Override
                    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
                        System.out.println("开始事务2");
                        //执行目标对象方法
                        Object returnValue = method.invoke(target, args);
                        System.out.println("提交事务2");
                        return returnValue;
                    }
                }
        );
    }

}
```

测试类:App.java

2. Python的hasattr() getattr() setattr() 函数使用方法详解(18)
3. java的会话管理：Cookie和Session(5)
4. Ognl表达式基本原理和使用方法(4)
5. Python的列表推导式，字典推导式，集合推导式使用方法(3)

```
/**
 * 测试类
 */
public class App {
    public static void main(String[] args) {
        // 目标对象
        IUserDao target = new UserDao();
        // 【原始的类型 class cn.itcast.b_dynamic.UserDao】
        System.out.println(target.getClass());

        // 给目标对象，创建代理对象
        IUserDao proxy = (IUserDao) new ProxyFactory(target).getProxyInstance();
        // class $Proxy0   内存中动态生成的代理对象
        System.out.println(proxy.getClass());

        // 执行方法    【代理对象】
        proxy.save();
    }
}
```

总结:

代理对象不需要实现接口,但是目标对象一定要实现接口,否则不能用动态代理

1.3.Cglib代理

上面的静态代理和动态代理模式都是要求目标对象是实现一个接口的目标对象,但是有时候目标对象只是一个单独的对象,并没有实现任何的接口,这个时候就可以使用以目标对象子类的方式类实现代理,这种方法就叫做:Cglib代理

Cglib代理,也叫作子类代理,它是在内存中构建一个子类对象从而实现对目标对象功能的扩展.

- JDK的动态代理有一个限制,就是使用动态代理的对象必须实现一个或多个接口,如果想代理没有实现接口的类,就可以使用Cglib实现.
- Cglib是一个强大的高性能的代码生成包,它可以在运行期扩展java类与实现java接口.它广泛的被许多AOP的框架使用,例如Spring AOP和synaop,为他们提供方法的interception(拦截)
- Cglib包的底层是通过使用一个小而块的字节码处理框架ASM来转换字节码并生成新的类.不鼓励直接使用ASM,因为它要求你必须对JVM内部结构包括class文件的格式和指令集都很熟悉.

Cglib子类代理实现方法:

1.需要引入cglib的jar文件,但是Spring的核心包中已经包括了Cglib功能,所以直接引入 `pring-core-3.2.5.jar` 即可.

2.引入功能包后,就可以在内存中动态构建子类

3.代理的类不能为final,否则报错

4.目标对象的方法如果为final/static,那么就不会被拦截,即不会执行目标对象额外的业务方法.

代码示例:

目标对象类:UserDao.java

```
/**
 * 目标对象,没有实现任何接口
 */
public class UserDao {

    public void save() {
        System.out.println("----已经保存数据!----");
    }
}
```

Cglib代理工厂:ProxyFactory.java

```
/**
 * Cglib子类代理工厂
 * 对UserDao在内存中动态构建一个子类对象
 */
public class ProxyFactory implements MethodInterceptor{
    //维护目标对象
    private Object target;

    public ProxyFactory(Object target) {
        this.target = target;
    }

    //给目标对象创建一个代理对象
    public Object getProxyInstance(){
        //1.工具类
        Enhancer en = new Enhancer();
        //2.设置父类
```

```
        en.setSuperclass(target.getClass());
        //3.设置回调函数
        en.setCallback(this);
        //4.创建子类(代理对象)
        return en.create();

    }

    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy)
throws Throwable {
        System.out.println("开始事务...");

        //执行目标对象的方法
        Object returnValue = method.invoke(target, args);

        System.out.println("提交事务...");

        return returnValue;
    }
}
```

测试类:

```
/**
 * 测试类
 */
public class App {

    @Test
    public void test(){
        //目标对象
        UserDao target = new UserDao();

        //代理对象
        UserDao proxy = (UserDao)new ProxyFactory(target).getProxyInstance();

        //执行代理对象的方法
        proxy.save();
    }
}
```

在Spring的AOP编程中:
如果加入容器的目标对象有实现接口,用JDK代理
如果目标对象没有实现接口,用Cglib代理

分类: 道·理论

标签: Java , Spring

« 上一篇：Spring与Struts框架整合
» 下一篇：Spring的AOP编程

posted @ 2017-01-16 12:46 岑宇 阅读(100681) 评论(21) 编辑 收藏

评论列表

#1楼 2017-03-09 00:31 **triumphofc**

文章写的不错 我理解很透彻。

支持(1) 反对(0)

#2楼 2017-06-17 21:55 **jiangxiong**

很好，简单易懂

支持(1) 反对(0)

#3楼 2017-07-23 13:58 **nele**

写的很简单明了

支持(0) 反对(0)

#4楼 2017-09-21 09:57 **babyHealthy**

好文章，通熟易懂

支持(0) 反对(0)

#5楼 2017-09-27 20:05 **liuyian**

刚好遇到一个cglib代理的问题，目标类有一个final方法，代理类代理不了；看到原理之后理解了很多

支持(0) 反对(0)

#6楼 2017-09-27 23:20 **林云枫**

6666

支持(1) 反对(0)

#7楼 2017-11-17 10:16 **Tur丶bo**

itcast..哈哈

支持(1) 反对(0)

#8楼 2017-11-26 00:21 **追风的狼**

很不错，总结的很清晰

支持(1) 反对(0)

#9楼 2017-11-26 00:22 **追风的狼**

@ 林云枫

引用

6666

支持(0) 反对(0)

#10楼 2018-01-10 11:28 **forget406**

理解

支持(0) 反对(0)

#11楼 2018-02-23 16:02 **frankchao1005**

很好

支持(0) 反对(0)

#12楼 2018-03-18 11:00 **optor**

在Spring的AOP编程中:
如果加入容器的目标对象有实现接口,用JDK代理
如果目标对象没有实现接口,用Cglib代理

支持(0) 反对(0)

#13楼 2018-03-18 11:00 **optor**

在Spring的AOP编程中:
如果加入容器的目标对象有实现接口,用JDK代理
如果目标对象没有实现接口,用Cglib代理
这两句话很实用啊，感觉面试时会被问到哈哈哈😆

支持(1) 反对(0)

#14楼 2018-03-21 09:55 **寻找灯塔**

这个很通俗

支持(1) 反对(0)

#15楼 2018-03-23 16:18 **jeff9571**

静态代理的代理对象为什么一定要实现接口？不实现也可以对目标对象功能的扩展

支持(1) 反对(0)

#16楼 2018-04-03 11:39 **Ethan Shan**

Mark.

支持(1) 反对(0)

#17楼 2018-04-19 10:07 **TryBest123**

讲的通俗易懂，赞一个！

支持(1) 反对(0)

#18楼 2018-04-26 10:59 **方头狼**

你的代码没运行过吧?
Cglib代理这里用target?

//执行目标对象的方法
Object returnValue = method.invoke(target, args);

支持(2) 反对(0)

#19楼 2018-04-28 00:52 **ShayneLee**

好文章，理解了好多

支持(0) 反对(0)

#20楼 2018-07-17 15:19 **liulehua**

简单易懂

支持(0) 反对(0)

#21楼 2018-08-06 17:50 **乖乖宁宝**

cglib动态代理，执行目标对象的方法那行应该是
Object returnValue = proxy.invokeSuper(obj, args);
寻找了半天资料，发现原来是这么写，博主的写法会莫名的调用若干次(输出了一下大约1万次)intercept方法，但是也不执行目标方法，才疏学浅，还不知道什么原因

支持(1) 反对(0)

相关博文:
· java设计模式------代理模式
· 静态代理与动态代理模式

· Java的三种代理模式(Spring动态代理对象)
· 设计模式——代理模式
· 设计模式之代理模式

最新新闻:
· GPU着火 AI业务受挑战 英伟达拐点已来?
· 微众银行张开翔：公众联盟链生态将呈现星星之火可以燎原之势
· ofo溃败英国市场：共享单车模式在英国困难重重
· Facebook申请数据挖掘新专利：可构建用户家族族谱图
· 全球首富贝索斯给儿女们的忠告：为后天的选择骄傲 而不是天赋
» 更多新闻...