Java动态代理详解

说到动态代理，顾名思义就是动态的代理(真是废话)。

关于代理：想必大家都并不陌生，GOF的23种设计模式之一(结构型模式)。这里暂不多做介绍，有兴趣的可以关注我关于设计模式的文章。

什么是动态代理：

说起动态，其实不如先说什么是静态。所谓静态代理，个人理解为自己手写的代理类，或者用工具生成的代理类，或者别人帮你写的代理类(没说一样...)。总之，就是程序运行前就已经存在的编译好的代理类。

相反，如果代理类程序运行前并不存在，需要在程序运行时动态生成(无需手工编写代理类源码)，那就是今天要说的动态代理了。

如何生成的：根据Java的反射机制动态生成。

不多说了，上程序。

目标接口TargetInterface:

**Java代码  [IMG_256](http://shensy.iteye.com/blog/javascript:void())**

1. **public** **interface** TargetInterface {
2. **public** **int** targetMethodA(**int** number);
3. **public** **int** targetMethodB(**int** number);
4. }

很简单，一个普通的接口，里面有若干方法(此处写2个示范一下)

实现该接口的委托类ConcreteClass:

**Java代码  [IMG_257](http://shensy.iteye.com/blog/javascript:void())**

1. **public** **class** ConcreteClass **implements** TargetInterface{
3. **public** **int** targetMethodA(**int** number) {
4. System.out.println("开始调用目标类的方法targetMethodA...");
5. System.out.println("操作-打印数字:"+number);
6. System.out.println("结束调用目标类的方法targetMethodA...");
7. **return** number;
8. }
10. **public** **int** targetMethodB(**int** number){
11. System.out.println("开始调用目标类的方法targetMethodB...");
12. System.out.println("操作-打印数字:"+number);
13. System.out.println("结束调用目标类的方法targetMethodB...");
14. **return** number;
15. }
17. }

很简单，一个普通的类，实现了目标接口。

代理处理器类ProxyHandler:

**Java代码  [IMG_258](http://shensy.iteye.com/blog/javascript:void())**

1. **public** **class** ProxyHandler **implements** InvocationHandler{
2. **private** Object concreteClass;
4. **public** ProxyHandler(Object concreteClass){
5. **this**.concreteClass=concreteClass;
6. }
8. **public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {
9. System.out.println("proxy:"+proxy.getClass().getName());
10. System.out.println("method:"+method.getName());
11. System.out.println("args:"+args[0].getClass().getName());
13. System.out.println("Before invoke method...");
14. Object object=method.invoke(concreteClass, args);//普通的Java反射代码,通过反射执行某个类的某方法
15. //System.out.println(((ConcreteClass)concreteClass).targetMethod(10)+(Integer)args[0]);
16. System.out.println("After invoke method...");
17. **return** object;
18. }
20. }

该类实现了Java反射包中的InvocationHandler接口。代理实例调用方法时，将对方法调用指派到它的代理处理器程序的invoke方法中。invoke方法内部实现预处理，对委托类方法调用，事后处理等逻辑。

最后是入口程序:

**Java代码  [IMG_259](http://shensy.iteye.com/blog/javascript:void())**

1. **public** **class** DynamicProxyExample {
2. **public** **static** **void** main(String[] args){
3. ConcreteClass c=**new** ConcreteClass();//元对象(被代理对象)
4. InvocationHandler ih=**new** ProxyHandler(c);//代理实例的调用处理程序。
5. //创建一个实现业务接口的代理类,用于访问业务类(见代理模式)。
6. //返回一个指定接口的代理类实例，该接口可以将方法调用指派到指定的调用处理程序，如ProxyHandler。
7. TargetInterface targetInterface=
8. (TargetInterface)Proxy.newProxyInstance(c.getClass().getClassLoader(),c.getClass().getInterfaces(),ih);
9. //调用代理类方法,Java执行InvocationHandler接口的方法.
10. **int** i=targetInterface.targetMethodA(5);
11. System.out.println(i);
12. System.out.println();
13. **int** j=targetInterface.targetMethodB(15);
14. System.out.println(j);
15. }
16. }

首先创建委托类对象，将其以构造函数传入代理处理器，代理处理器ProxyHandler中会以Java反射方式调用该委托类对应的方法。然后使用Java反射机制中的Proxy.newProxyInstance方式创建一个代理类实例，创建该实例需要指定该实例的类加载器，需要实现的接口(即目标接口)，以及处理代理实例接口调用的处理器。

最后，调用代理类目标接口方法时，会自动将其转发到代理处理器中的invoke方法内，invoke方法内部实现预处理，对委托类方法调用，事后处理等逻辑。

使用Java动态代理机制的好处：

1、减少编程的工作量：假如需要实现多种代理处理逻辑，只要写多个代理处理器就可以了，无需每种方式都写一个代理类。

2、系统扩展性和维护性增强，程序修改起来也方便多了(一般只要改代理处理器类就行了)。

使用Java动态代理机制的限制：

目前根据GOF的代理模式，代理类和委托类需要都实现同一个接口。也就是说只有实现了某个接口的类可以使用Java动态代理机制。但是，事实上使用中并不是遇到的所有类都会给你实现一个接口。因此，对于没有实现接口的类，目前无法使用该机制。有人说这不是废话吗，本来Proxy模式定义的就是委托类要实现接口的啊！但是没有实现接口的类，该如何实现动态代理呢？

当然不是没有办法，这也是我后面抽时间要继续整理和总结原先使用过的一件神器，相关Blog会不定期发上来。那就是大名鼎鼎的CGLib...

PS：CGLib中的动态代理已经新鲜出炉，欢迎访问：<http://shensy.iteye.com/blog/1873155>