作者：雨夜偷牛的人  
链接：https://www.zhihu.com/question/20794107/answer/23330381  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

最近正好在看，特来挖坟。  
关于动态代理设计模式很可能题主就在不知不觉中使用了，例如Spring中的AOP，Struts2中的拦截器等。  
  
先来看静态代理模式代码：

package test;

public interface Subject

{

public void doSomething();

}

package test;

public class RealSubject implements Subject

{

public void doSomething()

{

System.out.println( "call doSomething()" );

}

}

package test;

public class SubjectProxy implements Subject

{

Subject subimpl = new RealSubject();

public void doSomething()

{

subimpl.doSomething();

}

}

package test;

public class TestProxy

{

public static void main(String args[])

{

Subject sub = new SubjectProxy();

sub.doSomething();

}

}

刚开始我会觉得SubjectProxy定义出来纯属多余，直接实例化实现类完成操作不就结了吗？后来随着业务庞大，你就会知道，实现proxy类对真实类的封装对于粒度的控制有着重要的意义。但是静态代理这个模式本身有个大问题，如果类方法数量越来越多的时候，代理类的代码量是十分庞大的。所以引入动态代理来解决此类问题。  
  
先看代码：

package test;

public interface Subject

{

public void doSomething();

}

package test;

public class RealSubject implements Subject

{

public void doSomething()

{

System.out.println( "call doSomething()" );

}

}

package test;

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

import java.lang.reflect.Proxy;

public class ProxyHandler implements InvocationHandler

{

private Object tar;

//绑定委托对象，并返回代理类

public Object bind(Object tar)

{

this.tar = tar;

//绑定该类实现的所有接口，取得代理类

return Proxy.newProxyInstance(tar.getClass().getClassLoader(),

tar.getClass().getInterfaces(),

this);

}

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

{

Object result = null;

//这里就可以进行所谓的AOP编程了

//在调用具体函数方法前，执行功能处理

result = method.invoke(tar,args);

//在调用具体函数方法后，执行功能处理

return result;

}

}

public class TestProxy

{

public static void main(String args[])

{

ProxyHandler proxy = new ProxyHandler();

//绑定该类实现的所有接口

Subject sub = (Subject) proxy.bind(new RealSubject());

sub.doSomething();

}

}

看完代码，现在我来回答，动态代理的作用是什么：

1. Proxy类的代码量被固定下来，不会因为业务的逐渐庞大而庞大；
2. 可以实现AOP编程，实际上静态代理也可以实现，总的来说，AOP可以算作是代理模式的一个典型应用；
3. 解耦，通过参数就可以判断真实类，不需要事先实例化，更加灵活多变。