# [Java动态代理之JDK实现和CGlib实现（简单易懂）](https://www.cnblogs.com/ygj0930/p/6542259.html)

**转载请注明原文地址：http://www.cnblogs.com/ygj0930/p/6542259.html**

      一：代理模式（静态代理）

          代理模式是常用设计模式的一种，我们在软件设计时常用的代理一般是指静态代理，也就是在代码中显式指定的代理。

          静态代理由 业务实现类、业务代理类 两部分组成。业务实现类 负责实现主要的业务方法，业务代理类负责对调用的业务方法作拦截、过滤、预处理，主要是在方法中首先进行预处理动作，然后调用业务实现类的方法，还可以规定调用后的操作。我们在需要调用业务时，不是直接通过业务实现类来调用的，而是通过业务代理类的同名方法来调用被代理类处理过的业务方法。

          静态代理的实现：

          1：首先定义一个接口，说明业务逻辑。

[IMG_256](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

package net.battier.dao;

/\*\*

\* 定义一个账户接口

\* @author Administrator

\*/

public interface Count {

// 查询账户

public void queryCount();

// 修改账户

public void updateCount();

}

[IMG_257](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

           2：然后，定义业务实现类，实现业务逻辑接口

[IMG_258](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

import net.battier.dao.Count; /\*\*

\* 委托类(包含业务逻辑)

\*

\* @author Administrator

\*

\*/ public class CountImpl implements Count {

@Override

public void queryCount() {

System.out.println("查看账户...");

}

@Override

public void updateCount() {

System.out.println("修改账户...");

}

}

[IMG_259](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

       3：定义业务代理类：通过组合，在代理类中创建一个业务实现类对象来调用具体的业务方法；通过实现业务逻辑接口，来统一业务方法；在代理类中实现业务逻辑接口中的方法时，进行预处理操作、通过业务实现类对象调用真正的业务方法、进行调用后操作的定义。

[IMG_260](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public class CountProxy implements Count {

private CountImpl countImpl; //组合一个业务实现类对象来进行真正的业务方法的调用

/\*\*

\* 覆盖默认构造器

\*

\* @param countImpl

\*/

public CountProxy(CountImpl countImpl) {

this.countImpl = countImpl;

}

@Override

public void queryCount() {

System.out.println("查询账户的预处理——————");

// 调用真正的查询账户方法 countImpl.queryCount();

System.out.println("查询账户之后————————");

}

@Override

public void updateCount() {

System.out.println("修改账户之前的预处理——————");

// 调用真正的修改账户操作 countImpl.updateCount();

System.out.println("修改账户之后——————————");

}

}

[IMG_261](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

       4：在使用时，首先创建业务实现类对象，然后把业务实现类对象作构造参数创建一个代理类对象，最后通过代理类对象进行业务方法的调用。

[IMG_262](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public static void main(String[] args) {

CountImpl countImpl = new CountImpl();

CountProxy countProxy = new CountProxy(countImpl);

countProxy.updateCount();

countProxy.queryCount();

}

[IMG_263](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

       静态代理的缺点很明显：一个代理类只能对一个业务接口的实现类进行包装，如果有多个业务接口的话就要定义很多实现类和代理类才行。而且，如果代理类对业务方法的预处理、调用后操作都是一样的（比如：调用前输出提示、调用后自动关闭连接），则多个代理类就会有很多重复代码。这时我们可以定义这样一个代理类，它能代理所有实现类的方法调用：根据传进来的业务实现类和方法名进行具体调用。——那就是动态代理。

    二：动态代理的第一种实现——JDK动态代理

**JDK动态代理所用到的代理类在程序调用到代理类对象时才由JVM真正创建，JVM根据传进来的 业务实现类对象 以及 方法名 ，动态地创建了一个代理类的class文件并被字节码引擎执行，然后通过该代理类对象进行方法调用。**我们需要做的，只需指定代理类的预处理、调用后操作即可。

       1：首先，定义业务逻辑接口

public interface BookFacade {

public void addBook();

}

       2：然后，实现业务逻辑接口创建业务实现类

[IMG_264](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public class BookFacadeImpl implements BookFacade {

@Override

public void addBook() {

System.out.println("增加图书方法。。。");

}

}

[IMG_265](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

       3：最后，实现 调用管理接口InvocationHandler  创建动态代理类

[IMG_266](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public class BookFacadeProxy implements InvocationHandler {

private Object target;//这其实业务实现类对象，用来调用具体的业务方法

/\*\*

\* 绑定业务对象并返回一个代理类

\*/

public Object bind(Object target) {

this.target = target; //接收业务实现类对象参数

//通过反射机制，创建一个代理类对象实例并返回。用户进行方法调用时使用

//创建代理对象时，需要传递该业务类的类加载器（用来获取业务实现类的元数据，在包装方法是调用真正的业务方法）、接口、handler实现类

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

target.getClass().getInterfaces(), this); }

/\*\*

\* 包装调用方法：进行预处理、调用后处理

\*/

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

throws Throwable {

Object result=null;

System.out.println("预处理操作——————");

//调用真正的业务方法

result=method.invoke(target, args);

System.out.println("调用后处理——————");

return result;

}

}

[IMG_267](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

       4：在使用时，首先创建一个业务实现类对象和一个代理类对象，然后定义接口引用（这里使用向上转型）并用代理对象.bind(业务实现类对象)的返回值进行赋值。最后通过接口引用调用业务方法即可。（接口引用真正指向的是一个绑定了业务类的代理类对象，所以通过接口方法名调用的是被代理的方法们）

[IMG_268](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public static void main(String[] args) {

BookFacadeImpl bookFacadeImpl=new BookFacadeImpl();

BookFacadeProxy proxy = new BookFacadeProxy();

BookFacade bookfacade = (BookFacade) proxy.bind(bookFacadeImpl);

bookfacade.addBook();

}

[IMG_269](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

        JDK动态代理的代理对象在创建时，需要使用业务实现类所实现的接口作为参数（因为在后面代理方法时需要根据接口内的方法名进行调用）。如果业务实现类是没有实现接口而是直接定义业务方法的话，就无法使用JDK动态代理了。并且，如果业务实现类中新增了接口中没有的方法，这些方法是无法被代理的（因为无法被调用）。

   三：动态代理的第二种实现——CGlib

**cglib是针对类来实现代理的，原理是对指定的业务类生成一个子类，并覆盖其中业务方法实现代理。因为采用的是继承，所以不能对final修饰的类进行代理。**

       1：首先定义业务类，无需实现接口（当然，实现接口也可以，不影响的）

public class BookFacadeImpl1 {

public void addBook() {

System.out.println("新增图书...");

}

}

       2：实现 MethodInterceptor方法代理接口，创建代理类

[IMG_270](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public class BookFacadeCglib implements MethodInterceptor {

private Object target;//业务类对象，供代理方法中进行真正的业务方法调用

//相当于JDK动态代理中的绑定

public Object getInstance(Object target) {

this.target = target; //给业务对象赋值 Enhancer enhancer = new Enhancer(); //创建加强器，用来创建动态代理类 enhancer.setSuperclass(this.target.getClass()); //为加强器指定要代理的业务类（即：为下面生成的代理类指定父类）

//设置回调：对于代理类上所有方法的调用，都会调用CallBack，而Callback则需要实现intercept()方法进行拦  
 enhancer.setCallback(this);   
 // 创建动态代理类对象并返回   
 return enhancer.create();   
 }  
 // 实现回调方法   
 public Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy) throws Throwable {   
 System.out.println("预处理——————");  
 proxy.invokeSuper(obj, args); //调用业务类（父类中）的方法  
 System.out.println("调用后操作——————");  
 return null;   
 }

[IMG_271](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

       3：创建业务类和代理类对象，然后通过  代理类对象.getInstance(业务类对象)  返回一个动态代理类对象（它是业务类的子类，可以用业务类引用指向它）。最后通过动态代理类对象进行方法调用。

[IMG_272](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

public static void main(String[] args) {

BookFacadeImpl1 bookFacade=new BookFacadeImpl1()；

BookFacadeCglib cglib=new BookFacadeCglib();

BookFacadeImpl1 bookCglib=(BookFacadeImpl1)cglib.getInstance(bookFacade);

bookCglib.addBook();

}

[IMG_273](https://www.cnblogs.com/ygj0930/p/javascript:void(0);)

    四：比较

    静态代理是通过在代码中显式定义一个业务实现类一个代理，在代理类中对同名的业务方法进行包装，用户通过代理类调用被包装过的业务方法；

    JDK动态代理是通过接口中的方法名，在动态生成的代理类中调用业务实现类的同名方法；

    CGlib动态代理是通过继承业务类，生成的动态代理类是业务类的子类，通过重写业务方法进行代理；