写框架之——实现自己的AOP

我们来实现一个AOP，使用cglib配合Annotation

先看代码后解释：

public interface AOPInterface {

 public void BeforeMethod();

 public void AfterMethod();

}

上面的是AOP的接口，包含了方法执行前和执行后的方法

public class Caller implements MethodInterceptor {

 private AOPInterface Iaop;

 public Object intercept(Object obj, Method method, Object[] args,MethodProxy proxy) throws Throwable {

  Object result = null;

  /\*\*

   \* 当Annotation在类上方的时候，应该判断superclass是否有Annotation

   \* 因为cglib帮你创建出来的已经是动态继承了你的类，成了子类！！

   \* 关键！！

   \*/

  if(obj.getClass().getSuperclass().isAnnotationPresent(InterceptorHandler.class)){

   this.Iaop.BeforeMethod();

   result = invokeMethod(obj,args,proxy); //方法在下面

   this.Iaop.AfterMethod();

   return result;

  }else{

   if(method.isAnnotationPresent(InterceptorHandler.class)){

    this.Iaop.BeforeMethod();

    result = invokeMethod(obj,args,proxy); //方法在下面

    this.Iaop.AfterMethod();

    return result;

   }else{

    return invokeMethod(obj,args,proxy);

   }

  }

 }

 private Object invokeMethod(Object obj,Object[] args,MethodProxy proxy) throws Throwable {

  return proxy.invokeSuper(obj, args);

 }

// get、set省略...

}

这个是cglib代理方法的调用类，为它加入了一个AOP接口的引用，使得方法调用前后可以灵活的改变

（有点类似于策略模式）

public class TestHandler implements AOPInterface {

 public void AfterMethod() {

  System.out.println("Good Bye!");

 }

 public void BeforeMethod() {

  System.out.println("Hello!");

 }

}

这个是真实的AOP处理类，实现了AOPInterface接口，只要实现了这个接口，就可以在AOP的时候调用

里面的方法 After和Before

@Retention(RetentionPolicy.RUNTIME)

@Target(value={ElementType.METHOD,ElementType.TYPE})

public @interface InterceptorHandler {

 public Class<?> handlerClass();

}

这个是注解的开发！可以看到注解保持到RUNTIME，就可以用反射取得，

可以标注的地方为方法上（METHOD)和类上（类、接口、枚举都是使用TYPE类型）

唯一的属性是，处理这个AOP的类型

public class AOPFactory {

 public Map<String,Class<?>> beansInfo;

 /\*\*

  \* 这个方法是通过配置取得Bean信息

  \* 我在此处就省略，直接取得目标Bean,

  \* 对真实扫描不懂可以看我的博客“取得包内的类”

  \*/

 public AOPFactory(){

  this.beansInfo = new HashMap<String, Class<?>>();

  this.beansInfo.put("TrueClass", TrueClass.class);

  this.beansInfo.put("TrueClass2", TrueClass2.class);

  this.beansInfo.put("TestHandler", TestHandler.class);

 }

 /\*\*

  \* 这个方法我也简化,直接使用手动装配

  \*/

 public Object getBean(String beanName) throws InstantiationException, IllegalAccessException{

  //取得Bean的类型

  Class<?> clazz = this.beansInfo.get(beanName);

  //装配AOP

  AOPInterface handler = (AOPInterface)(this.beansInfo.get("TestHandler").newInstance());

  Caller c = new Caller();

  c.setIaop(handler);

  //cglib代理创建者

  Enhancer eh = new Enhancer();

  eh.setSuperclass(clazz);

  eh.setCallback(c);

  //创建代理

  return eh.create();

 }

}

上面这个是AOP的工厂类，由于篇幅关系，我做了非常大的简化，自动扫描包，改为了手动装配

如果想要了解像Spring一样自动扫描包的功能，在我博客上也有，有兴趣可以看一下

public class TrueClass {

 @InterceptorHandler(handlerClass=TestHandler.class)

 public void meetMM(){

  System.out.println("Just Go Around With MM!");

 }

 public void afterMeetMM(){

  System.out.println("Go Home!");

 }

}

@InterceptorHandler(handlerClass=TestHandler.class)

public class TrueClass2 {

 public void BuySomeThing(){

  System.out.println("我就要这些，多少钱？");

 }

 public void NoMoney(){

  System.out.println("我忘带钱了");

 }

}

上面两个类，是真实的业务类，是用来测试的，TrueClass类只有方法上有注解，而TrueClass2

类上有注解，代表了这个类的所有方法都被cglib拦截

下面是测试方法（JUnit4）

@Test

 public void testAOP(){

  AOPFactory factory =  new AOPFactory();

  try {

   TrueClass tc = (TrueClass)factory.getBean("TrueClass");

   tc.meetMM();

   tc.afterMeetMM();

   TrueClass2 tc2 = (TrueClass2)factory.getBean("TrueClass2");

   tc2.BuySomeThing();

   tc2.NoMoney();

  } catch (Exception e) {

   e.printStackTrace();

  }

 }

执行后输出的果是：

Hello!

Just Go Around With MM!

Good Bye!

Go Home!

Hello!

我就要这些，多少钱？

Good Bye!

Hello!

我忘带钱了

Good Bye!

可以看到我们注释上的方法都被拦截了，没有注释的就没有拦截（Go Home前后）

上面的方法要注意的地方再总结一下：

1）cglib方法代理方法的调用类(Caller)中 ，执行方法的函数不是invoke，而是invokeSuper

因为cglib动态继承了你的类，导致了invoke会不断执行Caller的方法，而不是代理类的

2）在判断类的上方是否有Annotation的时候

不是xxx.getClass.isAnnotationXXXXX()

而是xxx.getClass.getSuperClass.isAnnotationXXXXX()

原因也是cglib动态继承所造成的！

写完了，估计我在努力努力就可以写出个“有点实用”的Spring简易框架了

写给一直进步的人