实验06 Spring框架：Spring AOP应用

**【实验目的及要求】**

1. 了解AOP的基本概念；
2. 理解动态代理及基于代理类的AOP实现方法；
3. 基于XML和注解的AOP实现（AspectJ）；
4. 要求所有回答的文本格式：**五号，宋体、1.5倍行距，保留段单元格背景。**

**【实验步骤】**

**1、JDK动态代理**

（1）在IDEA中新建一个Maven项目，取名为ex06\_proxy。

（2）在com.ex06.tools包中新建接口Axe，该接口含一个方法void chop()。

|  |
| --- |
| package com.ex06.tools;  public interface Axe {  void chop(); } |

（3）在com.tools.impl包中新建Axe的实现类StealAxe。

|  |
| --- |
| package com.ex06.tools.impl; import com.ex06.tools.Axe;  public class SteelAxe implements Axe {  public void chop() {  System.*out*.println("用钢斧砍柴。");  } } |

（4）在com.ex06.aspect包中，编写工具保养类ToolUpkeep（即切面），其包含两个增强方法（用于模拟增强功能）。

|  |
| --- |
| package com.ex06.aspect;  public class ToolUpkeep {  public void grind() {  System.*out*.println("工具已处于最佳工作状态。");  }  public void repair() {  System.*out*.println("工具已维修好。");  } } |

（5）在com.ex06.proxy 包中，编写JDK代理类AxeProxyJdk，完成对**void** chop()方法的增强，即执行前的磨刀及执行后的维护功能。

|  |
| --- |
|  |

（6）在test/java目录中，创建测试类com.ex06.AxeTest，添加proxyJdkTest测试方法，测试利用JDK代理增加方法的效果。

|  |
| --- |
|  |

**说明**：本实验仅仅是实现了JDK动态代理，与教材中的相关内容类似。通过实验，加深了解JDK动态代理的基本思想，同时理解在实现过程中对接口的要求。

**2、CGLIB代理**

（1）接第1个实验，在com.ex06.proxy 包中，编写CGLIB代理类AxeProxyCglib，完成对**void** chop()方法的增强，即执行前的磨刀及执行后的维护功能。

|  |
| --- |
|  |

（2）在测试类com.ex06.AxeTest中，添加proxyCGLIBTest测试方法，测试利用CGLIB代理增加方法的效果。

|  |
| --- |
|  |

**说明**：CGLIB动态代理对被增强类没有实现接口的要求，这一点是与JDK动态代理的主要差别。

**3、基于XML的AOP实现**

（1）在IDEA中新建一个Maven项目，取名为ex06\_xml\_aspectj。

（2）参考第1个实验，创建Axe接口和SteelAxe实现类。

（3）除了核心依赖及JUnit依赖外，添加以下依赖。

|  |
| --- |
| <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>1.9.9.1</version> </dependency> <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.9.9.1</version> </dependency> |

（4）在com.ex06.aspectj包中，创建ToolUpkeep切面类，在类中依次添加前置通知、后置通知、环绕通知、异常通知及最终通知等5个方法（可参考教材P106页的内容编写代码，同时要求能够输出图5.5中的信息）。

|  |
| --- |
|  |

（5）在main/resources目录下，创建文件夹config，并在其中创建beans.xml文件，将下列代码复制进去（请留意<beans>…</beans>标签中约束信息的变化，详见下面的粗体字），并完善相关配置。

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  **xmlns:aop="http://www.springframework.org/schema/aop"**  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.3.xsd  **http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-4.3.xsd**">  <bean id="axe" class="com.ex06.tools.impl.SteelAxe" />  <bean id="toolUpkeep" class="com.ex06.aspect.ToolUpkeep" />  *<!-- aop配置-->* <aop:config>  <!-- 指定切入点 -->  <!-- 在这里配置切入点，前置、返回、环绕、最终及异常通知。 --> |
|  |
| </aop:config>  **<aop:aspectj-autoproxy proxy-target-class="true"/>** </beans> |

**注意**：当被增强的类为接口的实现时，需要在beans.xml中增加<aop:aspectj-autoproxy proxy-target-class="**true**"/>元素，并确保proxy-target-class属性的值为true。

（6）在test/java文件夹中，创建com.ex06.tools.AxeTest测试类，编写测试方法chopTest方法，运行结果如图6.1所示。

|  |
| --- |
| package com.ex06.tools;  import com.ex06.tools.impl.SteelAxe; import org.junit.Test; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class AxeTest {  @Test  public void chopTest(){  ApplicationContext applicationContext = new ClassPathXmlApplicationContext("classpath:config/beans.xml");  SteelAxe axe = applicationContext.getBean(SteelAxe.class);  axe.chop();  } } |

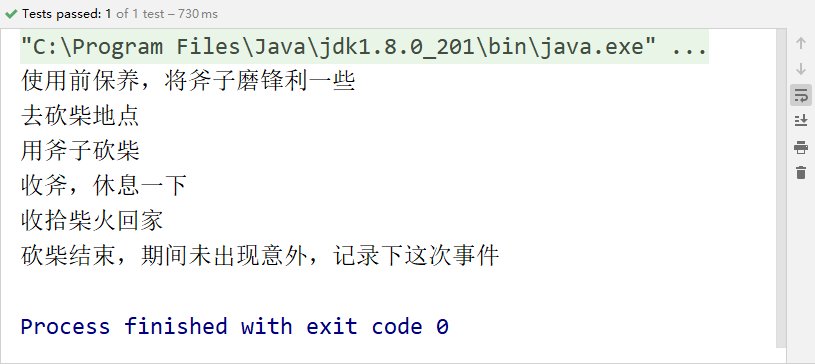


图6.1 程序的输出

**思考题1：**请结合实验情况，分析并列出5种通知的执行顺序。

|  |
| --- |
|  |

（7）如果在SteelAxe类的chop方法中添加语句“int I = 1/0;”，再次运行测试方法chopTest,给出运行结果的截图。

|  |
| --- |
|  |

**思考题2**：第（7）的输出与图6.1有何不同（即多了什么或少了什么）？请说明原因。

|  |
| --- |
|  |

**4、基于注解声明的AspectJ**

（1）在第3个实验的基础上，修改beans.xml文件，内容如下（请留意文件中的粗体信息）。

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  **xmlns:context="http://www.springframework.org/schema/context"**  xmlns:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.3.xsd  **http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.3.xsd**  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-4.3.xsd"> **<context:component-scan base-package="com.ex06.tools.impl,com.ex06.aspectj"/>** <aop:aspectj-autoproxy proxy-target-class="true"/> </beans> |

**说明**：beans.xml文件中，<context:component-scan/>元素的属性base-package值也可以改为"com.ex06"，即两个包路径的公共包路径。

（2）根据beans.xml文件中配置信息的变化，在ToolUpkeep类中添加注解信息，但需要保持切面、切入点配置不变。

|  |
| --- |
|  |

（3）根据beans.xml文件中配置信息的要求，在项目中其他位置添加注解信息，内容如下：

|  |
| --- |
|  |
|  |

（4）运行程序，要求运行结果与图6.1所示一致。

**思考题**：对比基于XML的AOP实现和基于注解的AOP实现之间的差异及其优缺点。

|  |
| --- |
|  |