实验十 **Sping** 的面向切面编程（**AOP**）——用户登录模块的

增强处理

一、基础实验——使用**@AspectJ** 实现 **AOP**

# （一）实验目的

1. 学习理解 AOP（Aspect Orient Programming）即面向切面编程的基本概念，重点掌握切面、增强处理、切点的概念；
2. 掌握使用@AspectJ 实现 Spring AOP 的基本步骤和配置方法，会使用基于 Annotation 的注解方式或基于 XML 配置文件的方式来定义切入点和增强处理。

# （二）基本知识与原理

1. AOP（Aspect Orient Programming），即面向切面编程，作为面向对象编程的一种补充，用于处理系统中分布于各个模块中共同关注的服务问题，如事物管理、安全检查、缓存、对象池管理等；
2. 面向切面编程的相关术语有：
3. 切面（Aspect）：业务流程运行的某个特定步骤，也就是应用运行过程的关注点，关注点可能横切多个对象，因此也被称为横切关注点；
4. 连接点（Joinpoint）：程序执行过程中明确的点，如方法的调用或异常的抛出，Spring AOP 中，连接点总是方法的调用；
5. 增强处理（Advice）：是切面的具体实现，在前面的某个特定的连接点上执行动作，Spring 中执行的动作往往就是调用某类的具体方法，如实现保存日志功能的类就是通知；
6. 切入点（Pointcut）：可以插入增强处理的连接点，即某连接点将被添加增强处理，则该连接点也就变成了切入点；

3、在@AspectJ 方式下，Spring 有两种途径来定义切入点和增强处理：

1. 基于 Annotation 的注解方式：使用@Aspect、@Pointcut 等 Annotation 来标注切入点和增强处理；
2. 基于 XML 配置文件的方式：使用 Sping 配置文件来定义切入点和增强处理。

# （三）实验内容及步骤

1. 在 Eclipse 中新建 Java 工程 spring-prj3，并添加 common-logging-1.2.jar 和 Spring 的 4 个基础 JAR 包到工程中（可参考实验八 SSH 整合的基础实验中的实验步骤二）；
2. 在工程 spring-prj3 中添加 Spring 框架中与 AOP 相关的 JAR 包：spring-aop-4.0.0.

RELEASE.jar；

1. 在站点 http://www.eclipse.org/aspectj/downloads.php 下载 aspectj-1.7.1.jar，并添加其中的 aspectjrt.jar、aspectjweaver.jar 到工程 spring-prj3 中；
2. 在站点 http://sourceforge.net/projects/aopalliance/下载 aopalliance.jar，并将其添加到工程 spring-prj3 中；
3. 在 spring-prj3 中新建 cn.edu.zjut.po 包，并在其中创建持久化类 Customer.java，代码片段如下：

|  |
| --- |
| package cn.edu.zjut.po;  public class Customer implements java.io.Serializable { private int customerId; private String account; private String password;  ……  //省略getters/setters方法 } |

1. 在 spring-prj3 中新建 cn.edu.zjut.dao 包，并在其中创建 ICustomerDAO 接口定义数据持久层的操作（代码略），以及实现类 CustomerDAO 实现数据持久层的操作，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.dao; import cn.edu.zjut.po.Customer;  public class CustomerDAO implements ICustomerDAO{ public void save(Customer transientInstance) {  System.out.println("execute --save()-- method.");  }  public void delete(Customer transientInstance) {  System.out.println("execute --delete()-- method.");  }  public void update(Customer transientInstance) {  System.out.println("execute --update()-- method.");  }  public void findByHQL(String hql) {  System.out.println("execute --findByHQL()-- method."); |
| }  } |

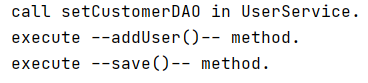
1. 在 spring-prj3 中新建 cn.edu.zjut.service 包，并在其中创建 IUserService 接口（代码略），以及实现类 UserService，代码片段如下：

|  |
| --- |
| package cn.edu.zjut.service; import cn.edu.zjut.dao.ICustomerDAO; import cn.edu.zjut.po.Customer;  public class UserService implements IUserService { private ICustomerDAO customerDAO = null;  public void setCustomerDAO(ICustomerDAO customerDAO) { System.out.println("call setCustomerDAO in UserService."); this.customerDAO = customerDAO;  }  public void addUser(Customer cust) {  System.out.println("execute --addUser()-- method."); customerDAO.save(cust);  }  public void delUser(Customer cust) { …… } public void updateUser(Customer cust) { …… } public void findUser(Customer cust) { …… }  } |

1. 在 spring-prj3 中创建 Spring 配置文件 applicationContext.xml，并在其中配置 CustomerDAO 实例和 UserService 实例，代码略；
2. 在 spring-prj3 中新建 cn.edu.zjut.app 包，并在其中创建测试类 SpringEnvTest，调用 UserService 实例的 addUser()方法，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.app;  ……  public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  IUserService userService = (IUserService) ctx.getBean("userService"); Customer cust = new Customer(); cust.setAccount("SPRING"); cust.setPassword("SPRING"); userService.addUser(cust); }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 在 spring-prj3 中新建 cn.edu.zjut.aspect 包，并在其中创建 SecurityHandler.java，用于实现权限检查，并用基于 Annotation 的注解方式定义切面，其中用@Aspect 修饰的类是切面类，用@Pointcut 定义切点，用

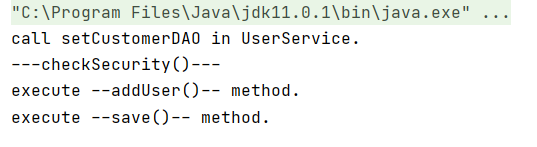
@Before 定义 Before 增强处理，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.aspect; import org.aspectj.lang.annotation.Before; import org.aspectj.lang.annotation.Aspect; import org.aspectj.lang.annotation.Pointcut;  **@Aspect** public class SecurityHandler {  /\*\* 定义Pointcut,Pointcut的名称是modify，  \* 此方法不能有返回值和参数，该方法只是一个标识\*/  **@Pointcut("execution(\* add\*(..)) || execution(\* del\*(..)) || execution(\* update\*(..))")** private void modify(){};  /\*\* 定义Advice，标识在那个切入点何处织入此方法 \*/  **@Before("modify()")**  private void checkSecurity() {  System.out.println("---checkSecurity()---"); } } |

1. 修改 Spring 配置文件 applicationContext.xml，在头文件中添加“xmlns：aop” 的命名申明，并在“xsi：schemaLocation”中指定 aop 配置的 schema 的地址，同时增加对 SecurityHandler 实例的配置，并启动注解配置 AOP 支持，具体代码如下：

|  |
| --- |
| <?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:p="http://www.springframework.org/schema/p" **xmlns:aop="http://www.springframework.org/schema/aop"** xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd **http://www.springframework.org/schema/aop**  **http://www.springframework.org/schema/aop/spring-aop-4.0.xsd"**>  <!-- 启动使用注解配置AOP支持 -->  **<aop:aspectj-autoproxy />**  **<bean id="securityHandler" class="cn.edu.zjut.aspect.SecurityHandler" />** |
| ……  </beans> |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



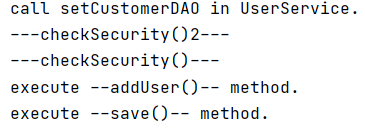
1. 在 spring-prj3 的 cn.edu.zjut.aspect 包中创建 SecurityHandler2.java，用于实现权限检查，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.aspect;  public class SecurityHandler2 { private void checkSecurity() {  System.out.println("---checkSecurity()2---"); } } |

1. 修改 Spring 配置文件 applicationContext.xml，增加对 SecurityHandler2 实例的配置，并使用 XML 配置文件的方式定义切面，代码片段如下：

|  |
| --- |
| <bean id="securityHandler2" class="cn.edu.zjut.aspect.SecurityHandler2" />  <!-- 配置文件的方式 -->  <aop:config>  <aop:aspect id="security" ref="securityHandler2">  <aop:pointcut id="modify" expression="execution(\* \*.add\*(..))" />  <aop:before method="checkSecurity" pointcut-ref="modify" />  </aop:aspect>  </aop:config> |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果。



# （四）实验要求

1、填写并上交实验报告，报告中应包括：

1. 运行结果截图；
2. 根据实验过程，查找相关资料，整理 Spring AOP 中的基本概念（如切面、增强处理、切点等），并记录下来；

切面是通知和切点的集合，通知和切点共同定义了切面的全部功能——它是什么，在何时何处完成其功能。

切点会匹配通知所要织入的一个或者多个连接点，定义通知被应用的位置

增强处理定义了切面是什么以及何时使用。除了描述切面要完成的工作，通知还解决何时执行这个工作。

1. 根据实验过程，总结 Spring AOP 的基本步骤，以及使用基于 Annotation 的注解方式或基于 XML 配置文件的方式来定义切入点和增强处理的基本方法，并记录下来；
2. 定义XXHandlerSupport的实现类
3. 要启用AOP，在Spring里面配置<aop:aspectj-autoproxy/>

**基于 Annotation 的 注解方式**

@Before： 标识一个前置增强方法，相当于BeforeAdvice的功能.  
@After： final增强，不管是抛出异常或者正常退出都会执行.  
@AfterReturning： 后置增强，似于AfterReturningAdvice, 方法正常退出时执行.  
@AfterThrowing： 异常抛出增强，相当于ThrowsAdvice.  
@Around： 环绕增强，相当于MethodInterceptor.  
execution：用于匹配方法执行的连接点；

任意公共方法的执行：execution(public \* \*(..))

任何一个以“set”开始的方法的执行：execution(\* set\*(..))

AccountService 接口的任意方法的执行：execution(\* com.xyz.service.AccountService.\*(..))

定义在service包里的任意方法的执行： execution(\* com.xyz.service.\*.\*(..))

定义在service包和所有子包里的任意类的任意方法的执行：execution(\* com.xyz.service..\*.\*(..))

**基于 XML 配置文件的方式来定义切入点和增强处理的基本方法**

配置<aop:config../>元素时，实质是将已有的Spring Bean转换成切面Bean，所以需要先定义一个普通的Spring Bean。因为切面Bean可以当成一个普通的Spring Bean来配置，所以我们完全可以为该切面Bean配置依赖注入。当切面Bean的定义完成后，通过<aop:congig../>元素中是哟个ref属性来引用该Bean，就可以将该Bean转换成切面Bean了。配置<aop:config../>元素时可以指定如下三个属性：

* id：该切面Bean的标识名
* ref：指定将要被转换成切面Bean的的普通Bean的id
* order：指定该切面Bean的优先级，值越小，优先级越高

1. 根据实验步骤 11 或 15，查找 AspectJ 切入点表达式的相关资料，记录其中 @Pointcut 注解中切入点表达式的含义；若切入点是 cn.edu.zjut.service 包下

所有实现类中的增删改方法，思考切入点表达式应该怎么写，并记录下来；

**切入点是 cn.edu.zjut.service 包下 所有实现类中的增删改方法：**

@Pointcut("execution(\* cn.zjut.edu.service.\*.add\*(..)) || execution(\* cn.zjut.edu.service.\*.del\*(..))"||\* cn.zjut.edu.service.\*.upd\*(..)))

1. 对比实验步骤 10 与 13，观察运行后的控制台输出以及相应的程序代码，总结 Spring AOP 优点或作用、适用场景，并记录下来；

**优点：**解耦：实现低耦合高内聚；实现代码复用，提高使用效率。

**适用场景：**Authentication 权限、Caching 缓存、Context passing 内容传递、Error handling 错误处理、Lazy loading　懒加载、Debugging　　调试、logging, tracing, profiling and monitoring　记录跟踪　优化　校准、Performance optimization　性能优化、Persistence　　持久化、Resource pooling　资源池、Synchronization　同步、Transactions 事务

（6）碰到的问题及解决方案或思考；

（7）实验收获及总结。

2、上交程序源代码，代码中应有相关注释。

二、提高实验——**SpringAOP** 实现事务管理

# （一）实验目的

1. 进一步理解 AOP 的基本概念与作用；
2. 理解 Spring 框架基于 AOP 实现声明式事务管理的基本机制，掌握进行声明式事务管理的基本方法；
3. 掌握 Spring 配置文件为实现声明式事务管理所涉及的主要元素及属性，理解其作用，并能进行正确的配置；
4. 了解 Spring 事务管理中的七种事务传播行为（propagation），并理解不同的事务传播行为起到的不同作用。

# （二）基本知识与原理

1. 事务管理是企业应用中非常重要的部分，Spring 框架对事务管理进行了高层次的抽象，定义了各种类型的事务管理器，用来实现事务管理功能；
2. Spring 框架支持编程式事务管理，也就是可以通过编写代码实现事务管理； Spring 同时也支持声明式事务管理，声明式事务管理基于 Spring AOP 实现，不在源文件中编写代码管理事务，而是使用 AOP 框架，在 IoC 容器中装配；
3. Spring 配置文件中提供了<tx:advice.../>元素来配置事务增强处理，并使用

<aop:advisor.../>为 IoC 容器中的 Bean 配置自动事务代理。

# （三）实验内容及步骤

1. 在 MySQL 中创建一个名称为 hibernatedb 的数据库，并在该数据库中创建一个名称为 customer 的数据表（表结构参照“实验八 SSH 整合”中提高实验的步骤一）；
2. 在工程 spring-prj3 中添加 MySQL 驱动程序库文件和 Hibernate 核心包，并添加 Spring 框架中与数据库操作相关的三个 JAR 包： spring-jdbc-4.0.0.

RELEASE.jar、spring-orm-4.0.0.RELEASE.jar、spring-tx-4.0.0.RELEASE.jar；

1. 在spring-prj3的cn.edu.zjut.po 包中创建Hibernate映射文件Customer.hbm.xml （代码略）；
2. 在工程的 cn.edu.zjut.dao 包中创建数据库操作基础类 BaseHibernateDAO.java，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.dao; import org.hibernate.Session; import org.hibernate.SessionFactory;  public class BaseHibernateDAO{ private SessionFactory sessionFactory;  public Session getSession(){ **return sessionFactory.getCurrentSession();**  }  public void setSessionFactory(SessionFactory sessionFactory) { this.sessionFactory = sessionFactory; }  } |

1. 修改 cn.edu.zjut.dao 包中的 CustomerDAO 实现类，使之继承数据库操作基础类 BaseHibernateDAO，并实现数据库操作，代码片段如下：

|  |
| --- |
| package cn.edu.zjut.dao; import org.hibernate.Query; import org.hibernate.Session; import cn.edu.zjut.po.Customer;  public class CustomerDAO **extends BaseHibernateDAO** implements ICustomerDAO{  public void save(Customer transientInstance) { System.out.println("execute --save()-- method."); try {  **getSession().save(transientInstance);**  } catch (RuntimeException re) { throw re;  }  }  ……  } |

1. 修改 Spring 配置文件 applicationContext.xml，在其中增加数据源的配置、增加 SessionFactory 实例的配置、增加 BaseHibernateDAO 实例的配置并在其中注入 sessionFactory，同时修改 CustomerDAO 实例的配置属性，代码略；
2. 运行测试类 SpringEnvTest，观察数据库中是否新增加了 SPRING 用户，思考导致该结果的原因并记录下来；

未增加

1. 修改 Spring 配置文件 applicationContext.xml，在头文件中添加“xmlns:tx”的命名申明，在“xsi:schemaLocation”中指定 tx 配置的 schema 的地址，代码

片段如下：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" **xmlns:tx="http://www.springframework.org/schema/tx"** xmlns:aop="http://www.springframework.org/schema/aop" xsi:schemaLocation=" http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xs d http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd **http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd** ">  ……  </beans> |

1. 修改 Spring 配置文件，配置事务管理器，代码片段如下：

|  |
| --- |
| <!-- 配置事务管理器 -->  <bean **id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManag er"**>  <property name="sessionFactory">  <ref bean="sessionFactory" />  </property>  </bean> |

1. 修改 Spring 配置文件，通过 AOP 的方式为工程 spring-prj3 添加事务管理，代码片段如下：

|  |
| --- |
| <!-- 定义增强处理拦截方法 -->  <tx:advice id="txAdvice" **transaction-manager="transactionManager"**>  <tx:attributes>  <tx:method name="add\*" propagation="REQUIRED" />  <tx:method name="update\*" propagation="REQUIRED" />  <tx:method name="delete\*" propagation="REQUIRED" />  <tx:method name="del\*" propagation="REQUIRED" />  <tx:method name="\*" read-only="true" />  </tx:attributes>  </tx:advice> |

1. 修改 Spring 配置文件，定义事务拦截切面，通过 AOP 的方式添加事务管理，代码片段如下（为避免 **[Spring](http://blog.csdn.net/mydeman/article/details/6947683)** [中混用](http://blog.csdn.net/mydeman/article/details/6947683)**[@Aspectj](http://blog.csdn.net/mydeman/article/details/6947683)** [和](http://blog.csdn.net/mydeman/article/details/6947683) **[Schema-based](http://blog.csdn.net/mydeman/article/details/6947683) [AOP](http://blog.csdn.net/mydeman/article/details/6947683)** 引起错误，注意删除基础实验中的 **AOP** 配置）：

|  |
| --- |
| <!-- 定义事务拦截切面 -->  <aop:config>  <aop:pointcut id="allServiceMethod" expression="execution(\* cn.edu.zjut.service.\*.\*(..))" />  <aop:advisor pointcut-ref="allServiceMethod" advice-ref="txAdvice" />  </aop:config> |

1. 运行测试类 SpringEnvTest，观察数据库中是否新增加了 SPRING 用户，思考导致该结果的原因并记录下来。



# （四）实验要求

1、填写并上交实验报告，报告中应包括：

1. 运行结果截图；
2. 根据实验步骤 4，查找相关资料，总结 sessionFactory 对象的 openSession() 方法与 getCurrentSession()方法的区别，思考在本次实验中为什么使用后者，并记录下来；

opensession每次打开的都是新的session，且需要close。而getCurrentSession则会先从上下文环境中找，找到了则会直接拿过来，如果没有，则创建新的session，且事务提交后会自动close。

**在本次实验中使用后者：**在一个应用程序中，如果DAO 层使用Spring来控制session 的生命周期，则首选getCurrentSession()。

在 SessionFactory 启动的时候，Hibernate 会根据配置创建相应的 CurrentSessionContext，在 getCurrentSession()被调用的时候，实际被执行的方法是 CurrentSessionContext.currentSession()。在 currentSession() 执行时，如果当前Session 为空，currentSession 会调用 SessionFactory的openSession。

1. 根据实验步骤 7，观察运行结果（数据库中是否新增加了 SPRING 用户），思考导致该结果的原因，并记录下来；
2. 根据实验步骤 9-11，查找相关资料，总结 Spring 配置文件中与实现声明式事务管理相关的主要元素及其属性的作用，并记录下来；

1） advice（建议）的命名：由于每个模块都会有自己的Advice，所以在命名上需要作出规范，初步的构想就是模块名+Advice（只是一种命名规范）。

（2） **tx:attribute标签所配置的是作为事务的方法的命名类型**。

         如<tx:method **name**="save\*" **propagation**="REQUIRED"/>

        其中\*为通配符，即代表以save为开头的所有方法，即表示符合此命名规则的方法作为一个事务。

**propagation**="REQUIRED"代表支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

（3） aop:pointcut标签配置参与事务的类，由于是在Service中进行数据库业务操作，配的应该是包含那些作为事务的方法的Service类。

       首先应该特别注意的是id的命名，同样由于每个模块都有自己事务切面，所以我觉得初步的命名规则因为 all+模块名+ServiceMethod。而且每个模块之间不同之处还在于以下一句：

**expression**="execution(\* com.test.testAda.test.model.service.\*.\*(..))"

       其中第一个\*代表返回值，第二\*代表service下子包，第三个\*代表方法名，“（..）”代表方法参数。

（4） aop:advisor标签就是把上面我们所配置的事务管理两部分属性整合起来作为整个事务管理。

1. 根据实验步骤 10，查找相关资料，总结 Spring 事务管理中的七种事务传播行为（propagation）所起到的不同作用，并记录下来；



1. 碰到的问题及解决方案或思考；
2. 实验收获及总结。

2、上交程序源代码，代码中应有相关注释。

三、扩展实验——**SpringAOP** 的核心工作原理：代理和代理工厂

# （一）实验目的

1. 进一步理解 AOP 的基本概念与作用；
2. 理解 Spring AOP 基于代理和代理工厂的核心工作原理，理解 Spring AOP 的主要实现过程；
3. 掌握使用 Spring 提供的代理工厂 ProxyFactoryBean 来实现 AOP 的方法，理解

Spring AOP 与 IoC 容器的结合方式，以及各自起到的作用；

1. 掌握 Spring AOP 中四种类型增强处理的作用和实现方法。

# （二）基本知识与原理

1. Spring AOP 的实现是基于代理和代理工厂的，由代理工厂创建代理对象，再由代理对象向目标对象的业务逻辑进行功能增强；
2. Spring 框架中提供了 ProxyFactory 和 ProxyFactoryBean 等多种代理工厂；使用

ProxyFactory，能够独立于 Spring 的 IoC 容器之外来使用 Spring 的 AOP 支持；而使用 ProxyFactoryBean，能使得 Spring AOP 与 IoC 容器有机地结合在一起，

Spring 的 IoC 容器可以管理切点（Pointcut）和增强处理（Advice）等 AOP 组件；

1. Spring AOP 可以创建两种类型的代理对象：Java 动态代理和 CGLIB 代理；Java 动态代理只能针对接口生成代理对象（JdkDynamicAopProxy），因此使用 Java 动态代理的对象必须实现至少一个接口；而 CGLIB 代理可以生成类级别的代理对象（Cglib2AopProxy）；
2. Spring AOP 中有四种增强处理的类型，如表 10-1 所示：表 10-1 Spring AOP 中增强处理的类型

|  |  |  |
| --- | --- | --- |
| 增强处理类型 | 接口 | 描述 |
| Around | org.aopalliance.intercept.MethodInterceptor | 在目标方法调用前后切入 |
| Before | Org.springframework.aop.  MethodBeforeAdvice | 在目标方法调用之前切入 |
| After | Org.springframework.aop.  AfterReturningAdvice | 在目标方法调用之后切入 |
| Throws | org.springframework.aop.ThrowsAdvice | 在目标方法抛出异常时切入 |

# （三）实验内容及步骤

1. 在 spring-prj3 中新建 cn.edu.zjut.advice 包，并创建 Before 类型的增强处理 LoggingAdvice.java，用于实现日志管理，其中的 before 方法定义了在目标方法调用之前需要切入的增强功能，具体代码如下：

package cn.edu.zjut.advice;

|  |
| --- |
| import org.springframework.aop.MethodBeforeAdvice; import java.lang.reflect.Method; import java.util.Date; public class LoggingAdvice **implements MethodBeforeAdvice** {  public void **before(Method m,Object[] args,Object target) throws Throwable** {  String date1 = (new Date()).toLocaleString();  System.out.println("信息：[logging before ]["+date1+"]用户 "  +args[0]+" 尝试修改个人信息...");  }  } |

1. 修改 Spring 配置文件 applicationContext.xml，在其中增加对 LoggingAdvice 实例的配置，代码片段如下：

<!-- 增强处理定义 -->

<bean id="logAdvice" class="cn.edu.zjut.advice.LoggingAdvice" />

1. 修改 Spring 配置文件 applicationContext.xml，在其中通过 ProxyFactoryBean 对目标对象和增强处理等进行装配，代码片段如下：

|  |
| --- |
| <!--设定代理-->  <bean id="logProxy" **class="org.springframework.aop.framework.ProxyFactoryBean"**>  <!--设定代理的是接口-->  <property **name="proxyInterfaces"**>  <value>cn.edu.zjut.service.IUserService</value>  </property>  <!--设定代理的目标类-->  <property **name="target"**>  <ref bean="userService"/>  </property>  <!--设定切入的Advice-->  <property **name="interceptorNames"**>  <list>  <value>logAdvice</value>  </list>  </property>  </bean> |

1. 修改 cn.edu.zjut.app 包中的测试类 SpringEnvTest ，通过代理工厂返回

UserService 实例，并访问其 addUser()方法，具体代码如下：

|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  IUserService userService = (IUserService) ctx.getBean(**"logProxy"**);  ……  userService.addUser(cust); }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；
2. 修改 cn.edu.zjut.advice 包中的 LoggingAdvice.java ，使其同时实现 AfterReturningAdvice 接口，并添加 afterReturning 方法用于定义在目标方法调

用之后需要切入的增强功能，代码片段如下：

|  |
| --- |
| package cn.edu.zjut.advice;  import org.springframework.aop.AfterReturningAdvice ;  ……  public class LoggingAdvice implements MethodBeforeAdvice, **AfterReturningAdvice** {  public void **afterReturning(Object returnValue, Method method, Object[] args, Object target) throws Throwable** {  String date2 = (new Date()).toLocaleString();  System.out.println("信息：[logging after ]["+date2+"]用户 "  +args[0]+" 成功修改个人信息...");  }  ……  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；
2. 在 cn.edu.zjut.advice 包 中 创 建 Around 类 型 的 增 强 处 理

LoggingInterceptor.java，并在其 invoke 方法中定义日志管理功能，其中的代码行“Object returnObject = invo.proceed();”不能缺少，需要通过它来调用目标对象的方法，具体代码如下：

package cn.edu.zjut.advice;

import java.util.Date; import org.aopalliance.intercept.MethodInterceptor; import org.aopalliance.intercept.MethodInvocation; public class LoggingInterceptor **implements MethodInterceptor** {

|  |
| --- |
| public Object **invoke(MethodInvocation invo) throws Throwable** {  Object[] args = invo.getArguments();  String date1 = (new Date()).toLocaleString();  System.out.println("Interceptor信息：[loggingbefore]["+date1  + "]用户 " + args[0] + " 尝试修改个人信息...");  **Object returnObject = invo.proceed();**  String date2 = (new Date()).toLocaleString();  System.out.println("Interceptor信息：[logging after ]["+date2  +"]用户 "+args[0]+" 成功修改个人信息...");  return args;  }  } |

1. 修改 Spring 配置文件 applicationContext.xml，在其中增加对 LoggingInterceptor 实例的配置，代码片段如下：

<!-- 增强处理定义 -->

<bean id="logInterceptor" class="cn.edu.zjut.advice.LoggingInterceptor" />

1. 修改 Spring 配置文件 applicationContext.xml ，在代理设定中添加对

LoggingInterceptor 的装配，代码片段如下：

|  |
| --- |
| <bean id="logProxy" class="org.springframework.aop.framework.ProxyFactoryBean">  ……  <property name="interceptorNames">  <list>  <value>logAdvice</value>  <value>logInterceptor</value>  </list>  </property>  </bean> |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果。

# （四）实验要求

1、填写并上交实验报告，报告中应包括：

1. 运行结果截图；
2. 根据实验过程，查找相关资料，总结使用 ProxyFactoryBean 实现 AOP 的基本步骤和主要配置方法，并记录下来；
3. 根据实验过程，查找相关资料，总结 Spring AOP 中四种类型增强处理的作用及实现方法，并记录下来；
4. 查找相关资料，思考如果需要代理的目标对象很多，可以借助于 Spring API 中的哪些接口或类来完成配置，并记录下来；
5. 碰到的问题及解决方案或思考；
6. 实验收获及总结。

2、上交程序源代码，代码中应有相关注释。