实验五 ： AspectJ 实现 AOP 的开发

一、 **目的：**Spring 的 AOP 模块，是 Spring 框架体系结构中十分重要的内容，该模块中提供了面向切面编程实现。

二、 **预习要求：**

1、 了解 AOP 的概念和作用

2、 理解 AOP 中的相关术语

3、 掌握基于 XML 和注解的 AspectJ 开发

三、 **作业内容：**

1、 基于 XML 的声明式 AspectJ

2、 基于注解的声明式 AspectJ

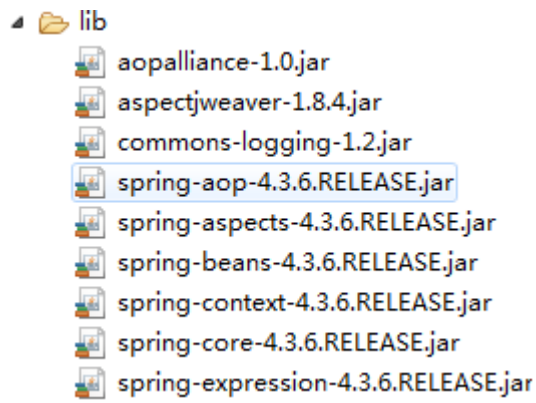
四、 **方法和步骤：**

按照教材 8.3 和 8.4 小节的内容和案例实现步骤，完成基于 XML 的声明式和基于注解的声明式 AspectJ 开发案例代码的编写。

Spring 使用 AspectJ 进行 AOP 的开发:XML 的方式

引入相应的 jar 包

```
* spring 的传统 AOP 的开发的包
spring-aop-4.3.6.RELEASE.jar
aopalliance-1.0.0.jar
* aspectJ 的开发包:
aspectjweaver-1.8.4.jar
spring-aspects-4.3.6.RELEASE.jar
```



引入 Spring 的配置文件

```
    引入 AOP 约束：
    <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">

    </beans>
```

编写目标类

```
    创建接口和类：
    public interface OrderDao {
        public void save();
        public void update();
        public void delete();
        public void find();
    }

    public class OrderDaoImpl implements OrderDao {

        @Override
        public void save() {
            System.out.println("保存订单...");
        }

        @Override
        public void update() {
```

```
            System.out.println("修改订单...");
        }

        @Override
        public void delete() {
            System.out.println("删除订单...");
        }

        @Override
        public void find() {
            System.out.println("查询订单...");
        }

    }
```

目标类的配置

```
        <!-- 目标类================ -->
        <bean id="orderDao" class="com.haust.dao.OrderDaoImpl">

        </bean>
```

整合 Junit 单元测试

```
    引入 spring-test.jar

    @RunWith(SpringJUnit4ClassRunner.class)
    @ContextConfiguration("classpath:applicationContext.xml")
    public class SpringDemo3 {
        @Resource(name="orderDao")
        private OrderDao orderDao;

        @Test
        public void demo1(){
            orderDao.save();
            orderDao.update();
            orderDao.delete();
            orderDao.find();
        }
    }
```

通知类型

```
    前置通知 ：在目标方法执行之前执行.
    后置通知 ：在目标方法执行之后执行
```

```
环绕通知 ：在目标方法执行前和执行后执行
异常抛出通知：在目标方法执行出现 异常的时候 执行
最终通知 ：无论目标方法是否出现异常 最终通知都会 执行.
```

切入点表达式

```
execution(表达式)
表达式:
[方法访问修饰符] 方法返回值 包名.类名.方法名(方法的参数)
public * com.haust.dao.*.*(..)
* com.haust.dao.*.*(..)
* com.haust.dao.UserDaoImpl.*(..)
* com.haust.dao..*.*(..)
```

编写一个切面类

```
public class MyAspectXml {
    // 前置增强
    public void before(){
        System.out.println("前置增强===========");
    }
}
```

配置完成增强

```
    <!-- 配置切面类 -->
    <bean id="myAspectXml" class="com.haust.aspect.MyAspectXml"></bean>

    <!-- 进行 aop 的配置 -->
    <aop:config>
        <!-- 配置切入点表达式:哪些类的哪些方法需要进行增强 -->
        <aop:pointcut                                expression="execution(*
com.haust.dao.OrderDaoImpl.save(..))" id="pointcut1"/>
        <!-- 配置切面 -->
        <aop:aspect ref="myAspectXml">
            <aop:before method="before" pointcut-ref="pointcut1"/>
        </aop:aspect>
    </aop:config>
```

其他的增强的配置：

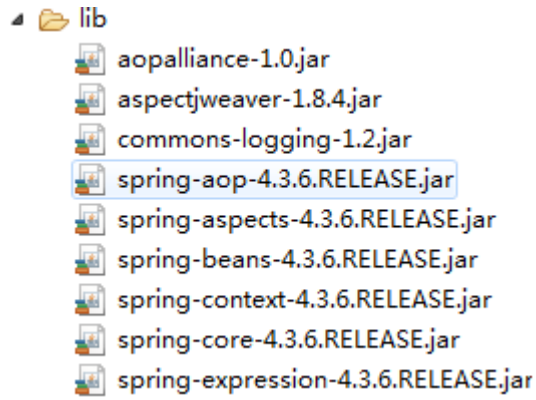
```
    <!-- 配置切面类 -->
    <bean id="myAspectXml" class="com.haust.myaspect.MyAspectXml"></bean>
```

```
        <!-- 进行 aop 的配置 -->
        <aop:config>
            <!-- 配置切入点表达式:哪些类的哪些方法需要进行增强 -->
            <aop:pointcut                                    expression="execution(*
com.haust.dao.*DaoImpl.save(..))" id="pointcut1"/>
            <aop:pointcut                                    expression="execution(*
com.haust.dao.*DaoImpl.delete(..))" id="pointcut2"/>
            <aop:pointcut                                    expression="execution(*
com.haust.dao.*DaoImpl.update(..))" id="pointcut3"/>
            <aop:pointcut                                    expression="execution(*
com.haust.dao.*DaoImpl.find(..))" id="pointcut4"/>
            <!-- 配置切面 -->
            <aop:aspect ref="myAspectXml">
                <aop:before method="before" pointcut-ref="pointcut1"/>
                <aop:after-returning     method="afterReturing"     pointcut-
ref="pointcut2"/>
                <aop:around method="around" pointcut-ref="pointcut3"/>
                <aop:after-throwing      method="afterThrowing"     pointcut-
ref="pointcut4"/>
                <aop:after method="after" pointcut-ref="pointcut4"/>
            </aop:aspect>
        </aop:config>
```

Spring 使用 AspectJ 进行 AOP 的开发:注解的方式

引入相关的 jar 包:

```
    * spring 的传统 AOP 的开发的包
    spring-aop-4.3.6.RELEASE.jar
    aopalliance-1.0.0.jar
    * aspectJ 的开发包:
    aspectjweaver-1.8.4.jar
    spring-aspects-4.3.6.RELEASE.jar
```



引入 Spring 的配置文件

```
    引入 AOP 约束：
    <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">

    </beans>
```

编写目标类:

```
    public class ProductDao {
        public void save();
        public void update();
        public void delete();
        public void find();
    }
    public class ProductDaoImpl implements ProductDao {
        public void save(){
            System.out.println("保存商品...");
        }
        public void update(){
            System.out.println("修改商品...");
        }
        public void delete(){
            System.out.println("删除商品...");
        }
        public void find(){
```

```
            System.out.println("查询商品...");
        }
    }
```

配置目标类：

```
        <!-- 目标类============ -->
        <bean                                                    id="productDao"
class="com.haust.spring.dao.ProductDaoImpl"></bean>
```

开启 aop 注解的自动代理:

```
    <aop:aspectj-autoproxy/>
```

AspectJ 的 AOP 的注解:

```
    @Aspect:定义切面类的注解

    通知类型:
        * @Before            :前置通知
        * @AfterReturing     :后置通知
        * @Around            :环绕通知
        * @After             :最终通知
        * @AfterThrowing     :异常抛出通知.

    @Pointcut:定义切入点的注解
```

编写切面类:

```
    @Aspect
    public class MyAspectAnno {

        @Before("MyAspectAnno.pointcut1()")
        public void before(){
            System.out.println("前置通知===========");
        }

        @Pointcut("execution(* com.haust.spring.demo4.ProductDao.save(..))")
        private void pointcut1(){}
    }
```

配置切面:

```
        <!-- 配置切面类 -->
        <bean                                                    id="myAspectAnno"
```

```
class="com.haust.spring.aspect.MyAspectAnno"></bean>
```

其他通知的注解:

```
@Aspect
public class MyAspectAnno {

    @Before("MyAspectAnno.pointcut1()")
    public void before(){
        System.out.println("前置通知==========");
    }

    @AfterReturning("MyAspectAnno.pointcut2()")
    public void afterReturning(){
        System.out.println("后置通知==========");
    }

    @Around("MyAspectAnno.pointcut3()")
    public Object around(ProceedingJoinPoint joinPoint) throws Throwable{
        System.out.println("环绕前通知==========");
        Object obj = joinPoint.proceed();
        System.out.println("环绕后通知==========");
        return obj;
    }

    @AfterThrowing("MyAspectAnno.pointcut4()")
    public void afterThrowing(){
        System.out.println("异常抛出通知========");
    }

    @After("MyAspectAnno.pointcut4()")
    public void after(){
        System.out.println("最终通知==========");
    }

    @Pointcut("execution(* com.haust.dao.ProductDaoImpl.save(..))")
    private void pointcut1(){}
    @Pointcut("execution(* com.haust.dao.ProductDaoImpl.update(..))")
    private void pointcut2(){}
    @Pointcut("execution(* com.haust.dao.ProductDaoImpl.delete(..))")
    private void pointcut3(){}
    @Pointcut("execution(* com.haust.dao.ProductDaoImpl.find(..))")
    private void pointcut4(){}
}
```

五、 思考题：

AspectJ 方式较之代理方式的优点是什么？

六、 作业要求：

（一） 作业目的：

（二） 作业内容：

（三） 作业结果：可以是运行结果截图或其他形式的结果展示

（四） 问题及解决：作业中遇到的问题及解决方法。

（五） 回答思考题提出的问题