

# 实验 8 实现基于注解的 AOP 实验程序

## 一、 实现基于注解的 AOP 需求描述

**编程 1：**代码演示基于注解的 AOP 的实现过程。

1. 创建切面类 AnnotationAdvice，并加注解@Aspect 声明为 AOP 切面类。配置相关的切点、连接点、通知或增强、织入等等，完善整个 AOP 切面编程

2. 创建业务代码，实现订单保存 save() 功能。包含 service 层，dao 层，并使用注解注册 bean。调用链：service 层方法调用 dao 层方法。

3. 创建测试类，测试上面 save() 方法，跟踪切面监控。

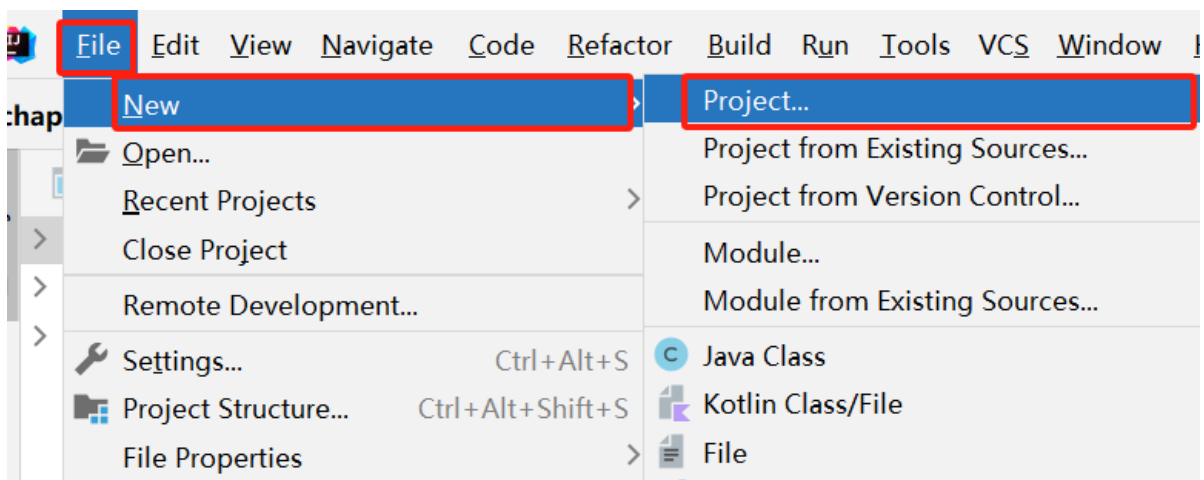
结论：通过调用切面类 AnnotationAdvice 的增强方法，验证使用 AOP 方式对目标对象（业务方法）的功能增强。

## 二、 开发步骤

### 第 1 步：创建 Maven 工程

方式一：直接创建新的 Maven 工程

按如下示例的步骤，进行工程创建（基于 IDEA 2022）



New Project

×

?

Generators

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Kotlin Multiplatform

Compose Multiplatform

HTML

React

Express

Angular CLI

IDE Plugin

Android

Vue.js

Vite

Name:

training

项目名称

Location:

D:\Workspace\Trainging\07.javaee\training

项目存放的目录

Project will be created in: D:\Workspace\Trainging\07.javaee\chapter01\demo

☐ Create Git repository

Template:

Library

Dependencies only

Application server:

Tomcat 9.0.44

New...

Language:

Java

Kotlin

Groovy

Build system:

Maven

Gradle

Group:

com.xhxc

项目源代码包命名路径

Artifact:

training

与项目名称保持一致即可

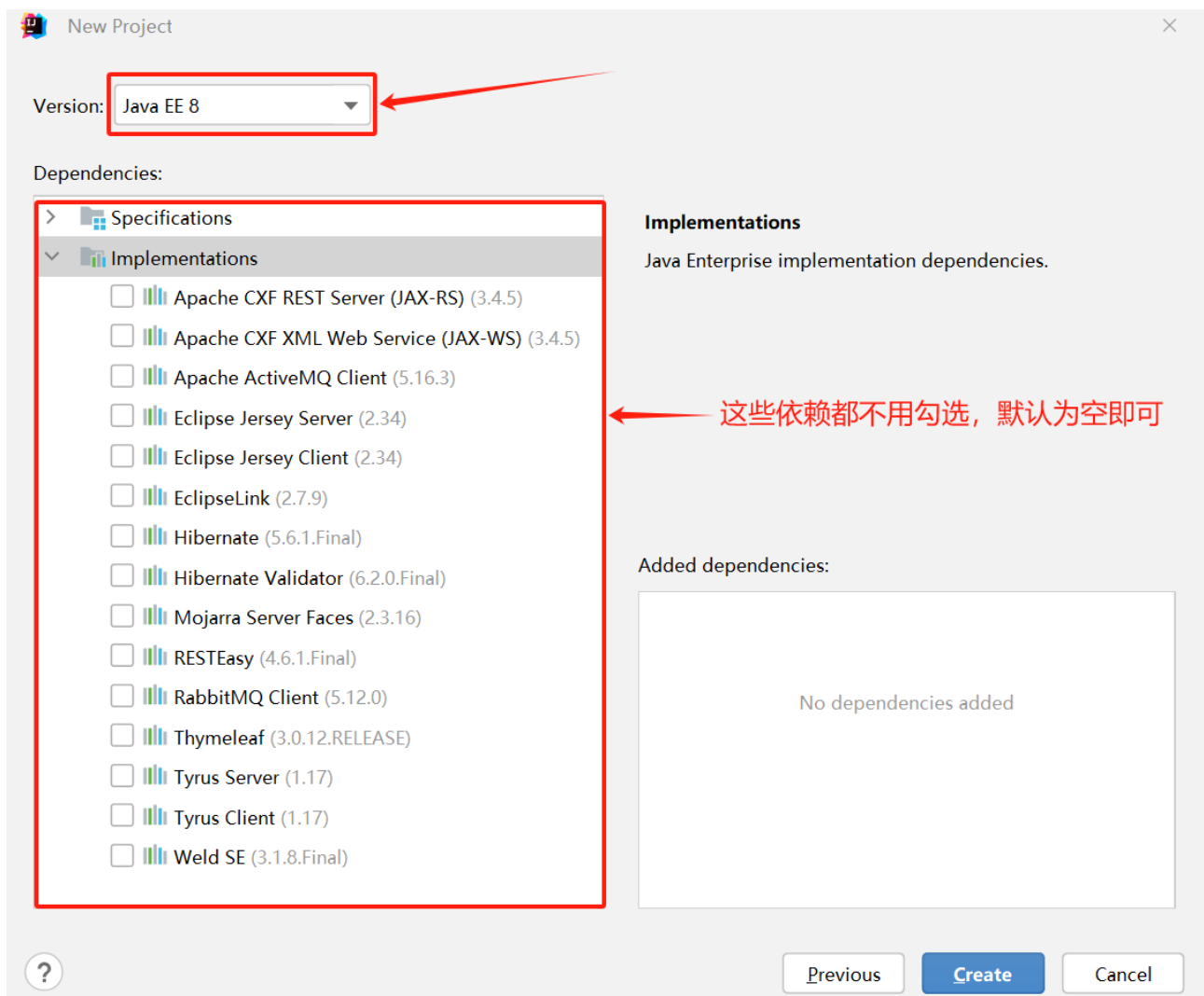
JDK:

1.8 Oracle OpenJDK version 1.8.0\_191

JDK版本

Next

Cancel




最后一步，点击【**Create**】即可完成工程创建。

### 方式二：导入已存在的 Maven 工程（推荐）

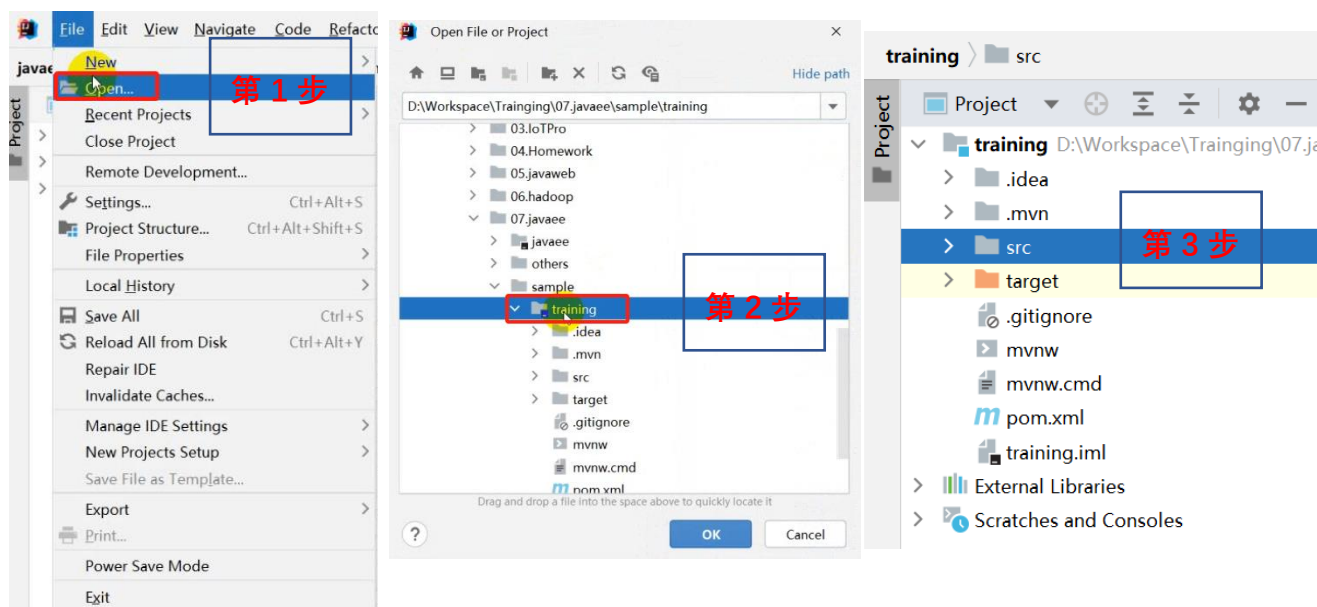
**Step1**，下载并解压 training 工程。从【学习通】→【章节】→【第八章…】→【上机实验 8…】，找到 **training.zip** 压缩包，下载到本地，并解压成 training 目录。

**Step2**，导入 training 工程。打开 IDEA 工具，点击【File】→【Open…】→选中上一步解压的 training 工程，如下图所示。。确定即可导入。

**注意**，导入工程后，对于 IDEA 是 2023 的版本，要点击左上角边的  文件夹小图标，即可查看到工程中的文件。

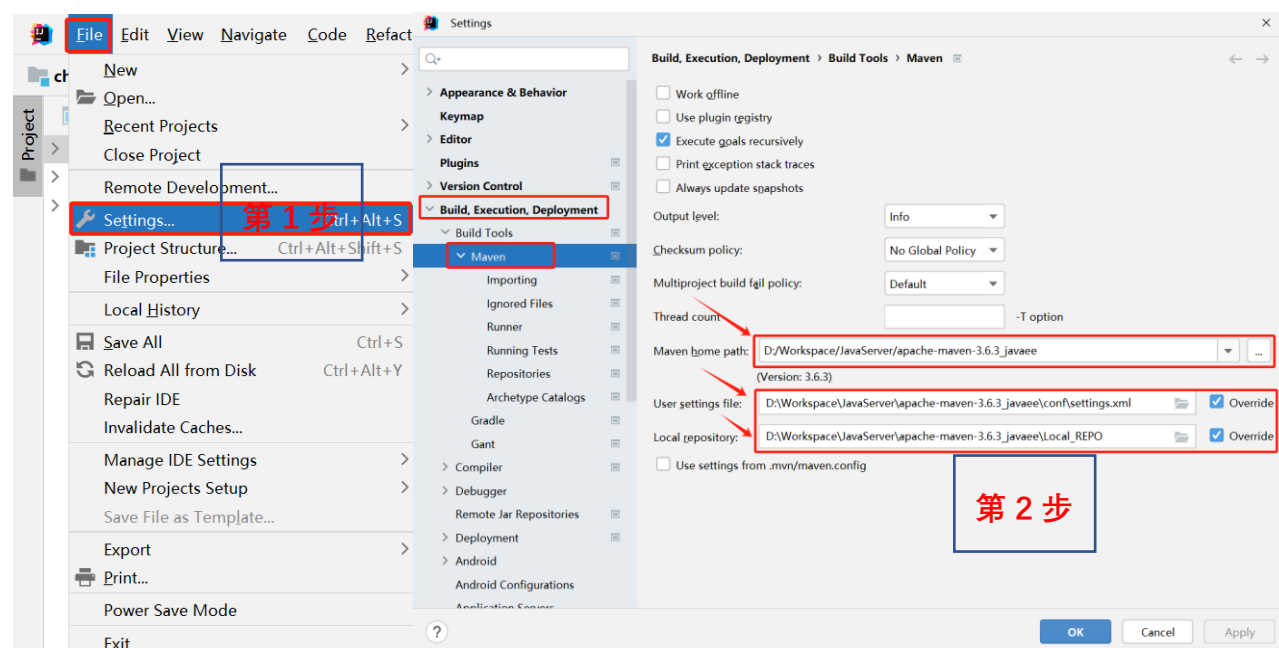
**Tips**：如果前几个章节已经下载并导入了此 **training** 工程，那么下载本章节最新工程后，

可以将新工程 **training** 文件夹内的所有文件全部复制，然后覆盖更新替换掉原有已经存在的 **training** 工程文件夹中的文件。这样，保证工程的内容是最新的，跟课程同步的。这样处理后，也就不需要再通过 IDEA 导入新工程了，直接使用旧 **training** 工程即可。



## 第 2 步：配置本地 Maven 路径

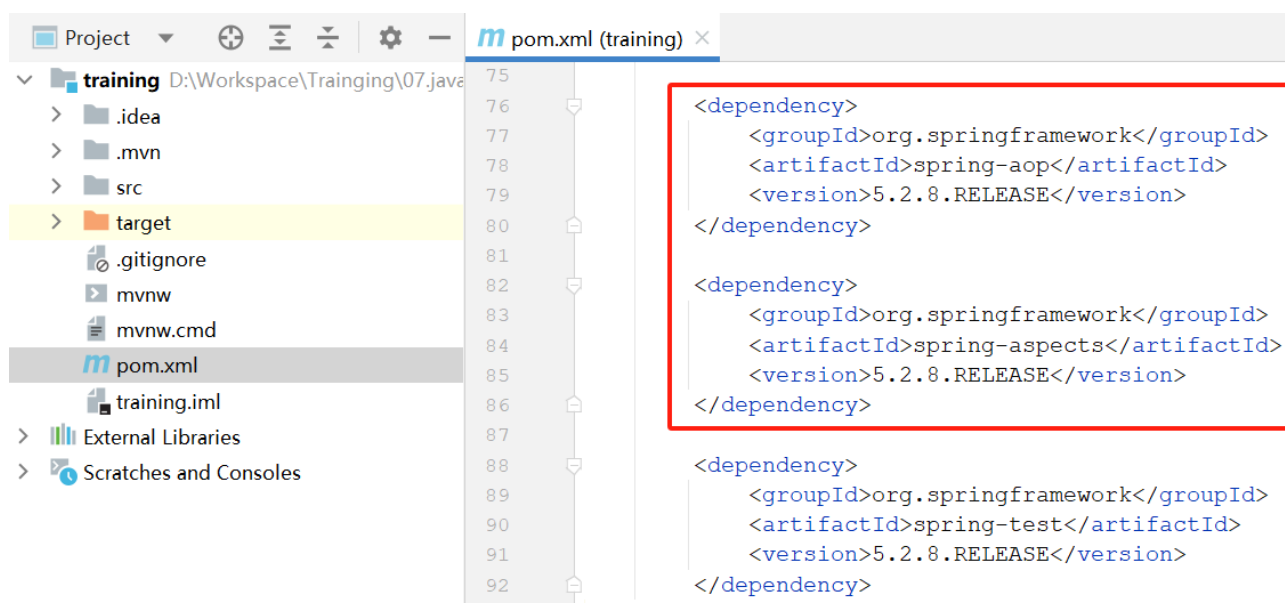
选择【File】→【Setting...】然后找到菜单【Build, Execution, Deployment】→【Build Tools】→【Maven】，按如下图所示，将你的本地 Maven 路径和配置文件设置好即可。



## 第 3 步：程序代码开发

### 1、配置 pom.xml

由于本章是在前 6、7 章的基础上学习 Spring 框架，因此，需要补充增加如下依赖，才能正常运行实验代码。pom.xml 文件需要新增依赖如下截图：



需要添加的依赖原配置为：

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>5.2.8.RELEASE</version>
</dependency>
```

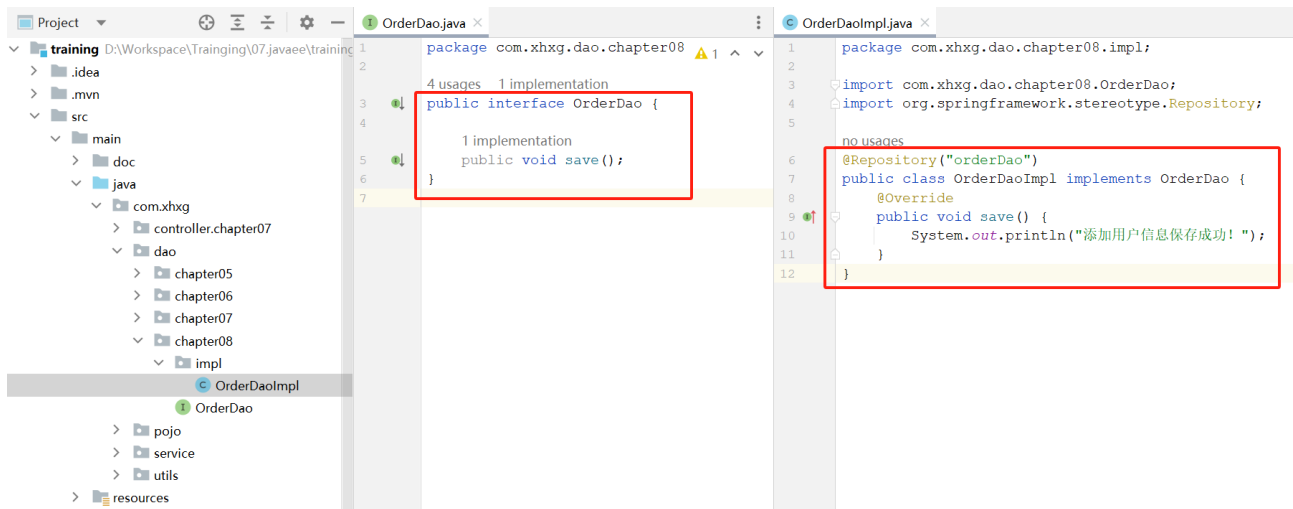
```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>5.2.8.RELEASE</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>5.2.8.RELEASE</version>
</dependency>
```

最后一个是增加 spring 注解测试的依赖，添加后就可以使用注解的方式读取配置文件并且使用自动注解的方式调用 bean。

## 2、创建 Dao 接口及其实现类

创建代码包为：**com.xhgx.dao.chapter08**，创建 OrderDao 接口类及其实现类 OrderDaoImpl，定义 save() 接口方法，代码如下截图所示：



**注意：**数据接口使用了注解 `@Repository("orderDao")`，其中括号中的 `orderDao` 名称为自定义 Bean 名称，如不自定义，默认以类的名称（首字母小写）作为 Spring Bean 名称

## 3、创建 Service 接口及其实现类

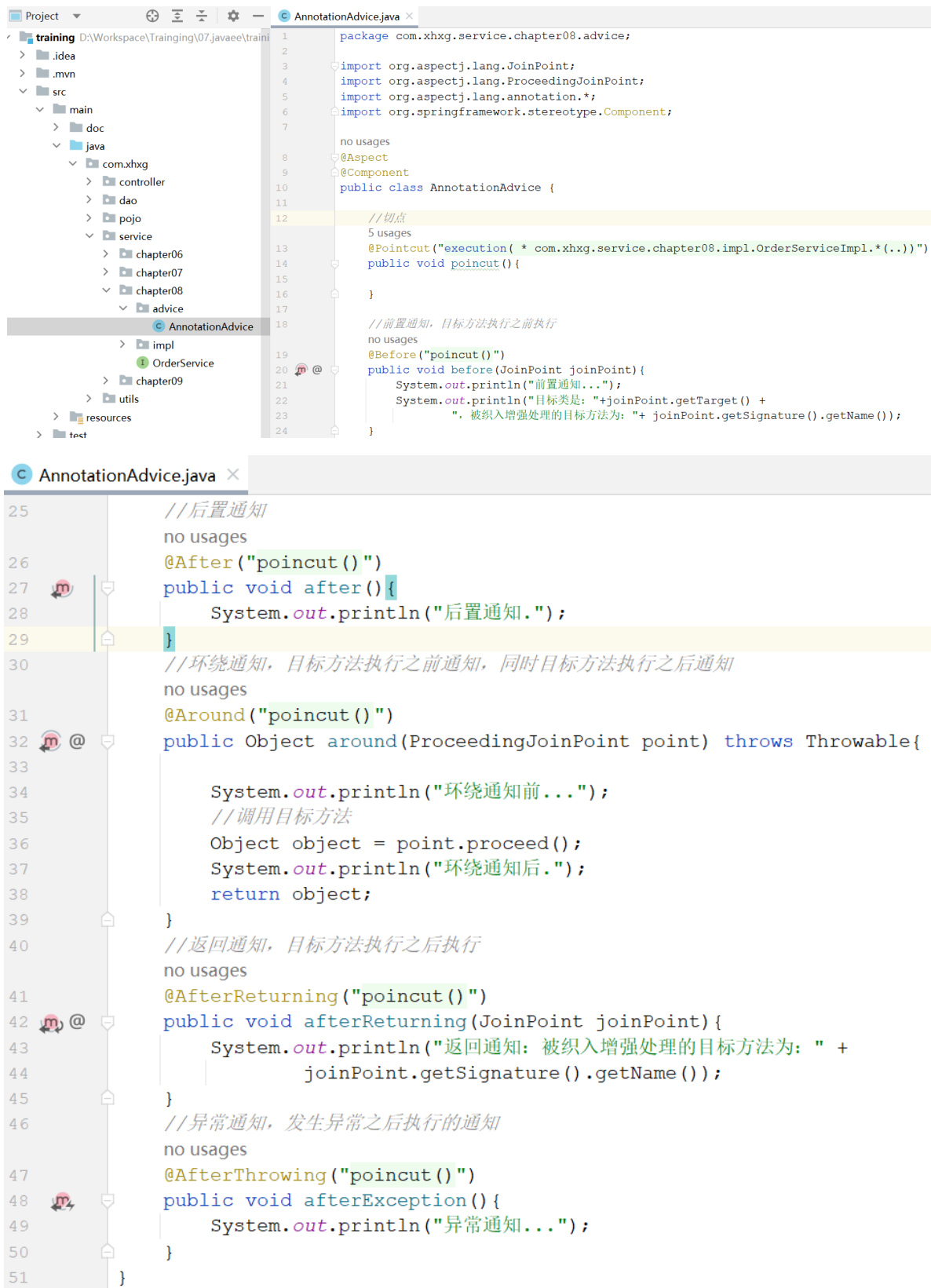
创建代码包为：**com.xhgx.service.chapter08**，创建 OrderService 接口类及其实现类 OrderServiceImpl，定义 save() 接口方法，代码如下截图所示：



**注意：**业务层接口使用了注解 `@Service("orderService")`，其中括号中的 `orderService` 名称为自定义 Bean 名称，如不自定义，默认以类的名称（首字母小写）作为 Spring Bean 名称。另外，在注入数据层接口时使用了自动注解 `@Autowired`，指向 Dao 数据层的接口 Bean。

## 4、创建切面类

创建代码包为：**com.xhxc.service.chapter08.advice**，创建 **AnnotationAdvice** 类，实现 AOP 切面监控的方法，并使用相应的注解，代码如下截图所示：

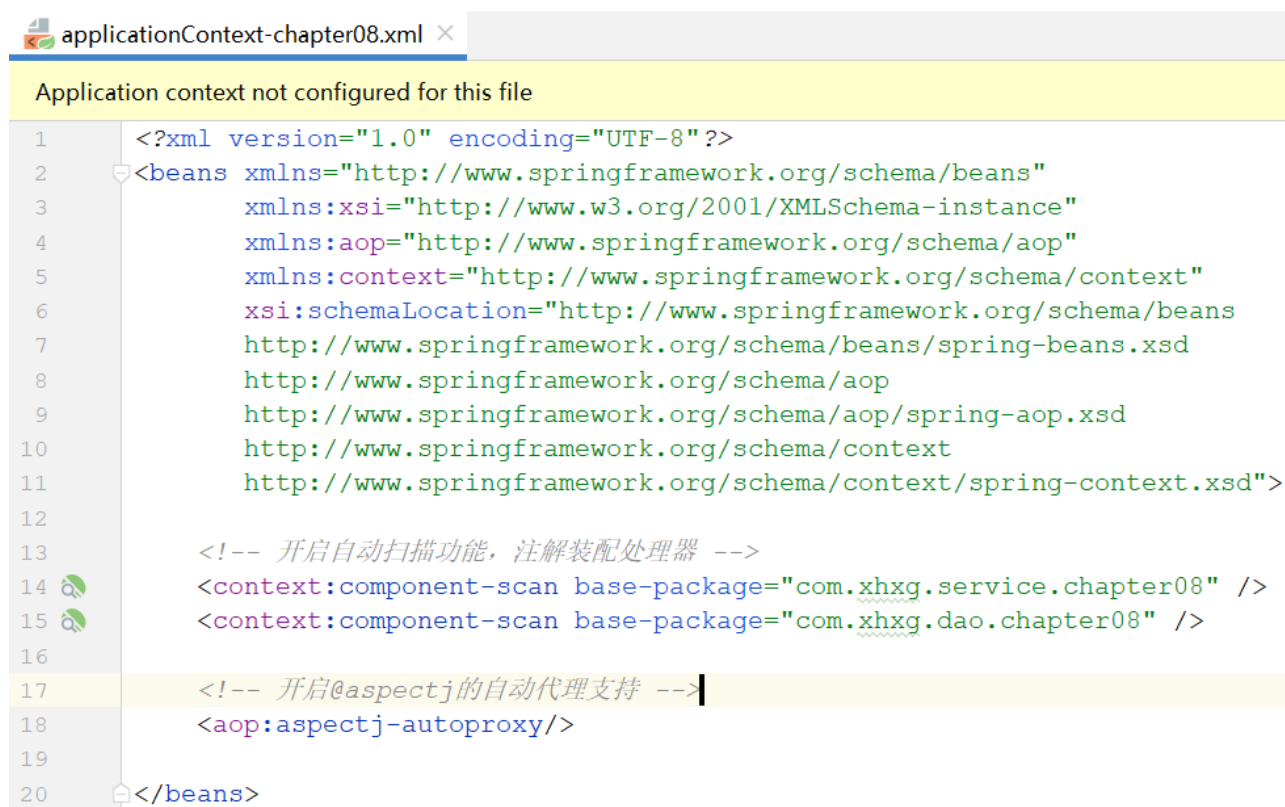


```
1 package com.xhxc.service.chapter08.advice;
2
3 import org.aspectj.lang.JoinPoint;
4 import org.aspectj.lang.ProceedingJoinPoint;
5 import org.aspectj.lang.annotation.*;
6 import org.springframework.stereotype.Component;
7
8 no usages
9 @Aspect
10 @Component
11 public class AnnotationAdvice {
12     //切点
13     @Pointcut("execution(* com.xhxc.service.chapter08.impl.OrderServiceImpl.*(..))")
14     public void pointcut() {
15     }
16
17     //前置通知，目标方法执行之前执行
18     no usages
19     @Before("pointcut()")
20     public void before(JoinPoint joinPoint) {
21         System.out.println("前置通知...");
22         System.out.println("目标类是: " + joinPoint.getTarget().getName() +
23             ", 被织入增强处理的目标方法为: " + joinPoint.getSignature().getName());
24     }
25
26     //后置通知
27     no usages
28     @After("pointcut()")
29     public void after() {
30         System.out.println("后置通知.");
31     }
32
33     //环绕通知，目标方法执行之前通知，同时目标方法执行之后通知
34     no usages
35     @Around("pointcut()")
36     public Object around(ProceedingJoinPoint point) throws Throwable {
37         System.out.println("环绕通知前...");
38         //调用目标方法
39         Object object = point.proceed();
40         System.out.println("环绕通知后.");
41         return object;
42     }
43
44     //返回通知，目标方法执行之后执行
45     no usages
46     @AfterReturning("pointcut()")
47     public void afterReturning(JoinPoint joinPoint) {
48         System.out.println("返回通知: 被织入增强处理的目标方法为: " +
49             joinPoint.getSignature().getName());
50     }
51
52     //异常通知，发生异常之后执行的通知
53     no usages
54     @AfterThrowing("pointcut()")
55     public void afterException() {
56         System.out.println("异常通知...");
57     }
58 }
```

**注意：**切面类使用了注解**@Aspect** 声明为 AOP 的切面类，并使用**@Component** 注册为 bean。分别定义切点方法，使用注解**@Pointcut** 声明；前置通知方法，使用注解**@Before** 声明；后置通知方法，使用注解**@After** 声明；环绕通知方法，使用注解**@Around** 声明；返回通知方法，使用注解**@AfterReturning** 声明；异常通知方法，使用注解**@AfterThrowing** 声明。

## 5、创建 Spring 配置文件 applicationContext.xml

在 src/main/resources 目录下创建一个 **applicationContext-chapter08.xml** 文件，内容如下：



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4        xmlns:aop="http://www.springframework.org/schema/aop"
5        xmlns:context="http://www.springframework.org/schema/context"
6        xsi:schemaLocation="http://www.springframework.org/schema/beans
7                             http://www.springframework.org/schema/beans/spring-beans.xsd
8                             http://www.springframework.org/schema/aop
9                             http://www.springframework.org/schema/aop/spring-aop.xsd
10                            http://www.springframework.org/schema/context
11                            http://www.springframework.org/schema/context/spring-context.xsd">
12
13      <!-- 开启自动扫描功能，注解装配处理器 -->
14      <context:component-scan base-package="com.xhxc.service.chapter08" />
15      <context:component-scan base-package="com.xhxc.dao.chapter08" />
16
17      <!-- 开启@aspectj的自动代理支持 -->
18      <aop:aspectj-autoproxy/>
19
20  </beans>
```

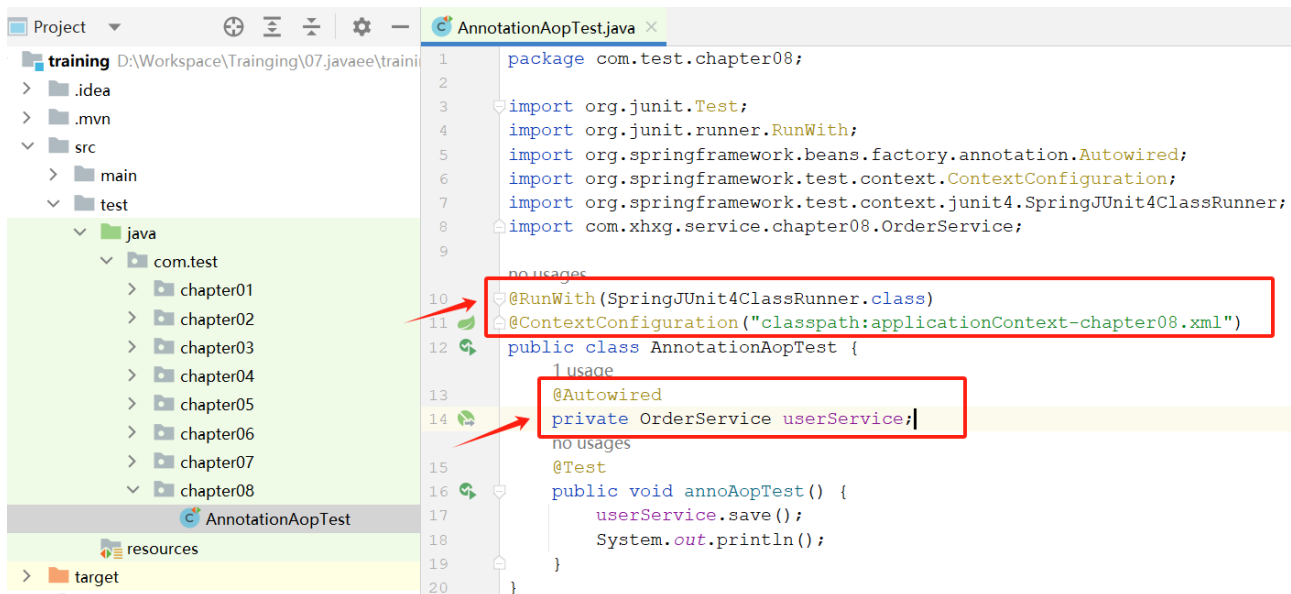
在该 Spring 配置文件中，需要开启自动扫描功能 **<context:component-scan base-package="xxxx" />**，和自动代理模式 **<aop:aspectj-autoproxy/>** 实现注解。

## 7、编写测试类

以上代码和配置创建好后，就可以编写测试类进行验证测试。

我们在 src/test/java 目录下，建议测试包，路径为：com.test.chapter08，然后创建测试类 **AnnotationAopTest**，定义测试方法 **annoAopTest()**，代码如下：





在本单元测试中，通过注解读取 Spring 配置文件 **applicationContext-chapter08.xml**，并且使用自动注解的方式读取 Bean 实例，代码如下所示。

注意，使用注解方式进行测试，需要下载 spring-test 依赖，因此 pom.xml 需要加入：

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>5.2.8.RELEASE</version>
</dependency>
```

运行结果：

```
环绕通知前...
前置通知...
目标类是: com.xhcg.service.chapter08.impl.OrderServiceImpl@3f6f6701, 被织入增强处理的目标方法为: save
添加用户信息保存成功!
返回通知: 被织入增强处理的目标方法为: save
后置通知.
环绕通知后.
```

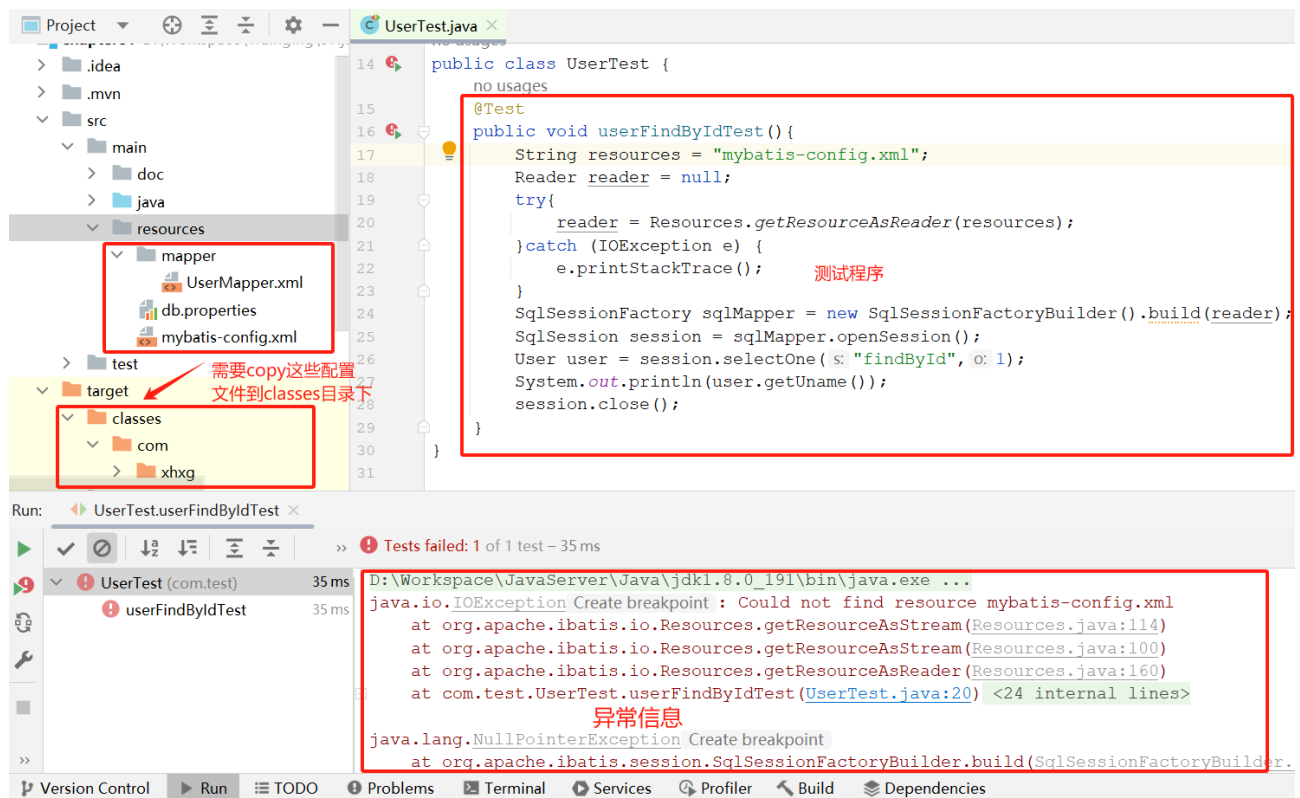
至此，本实验圆满成功。

### 三、常见问题

有时候 Maven 不能正确的复制配置文件到 target 目录中，此时运行测试程序就会报如下异常：Could not find resource mybatis-config.xml

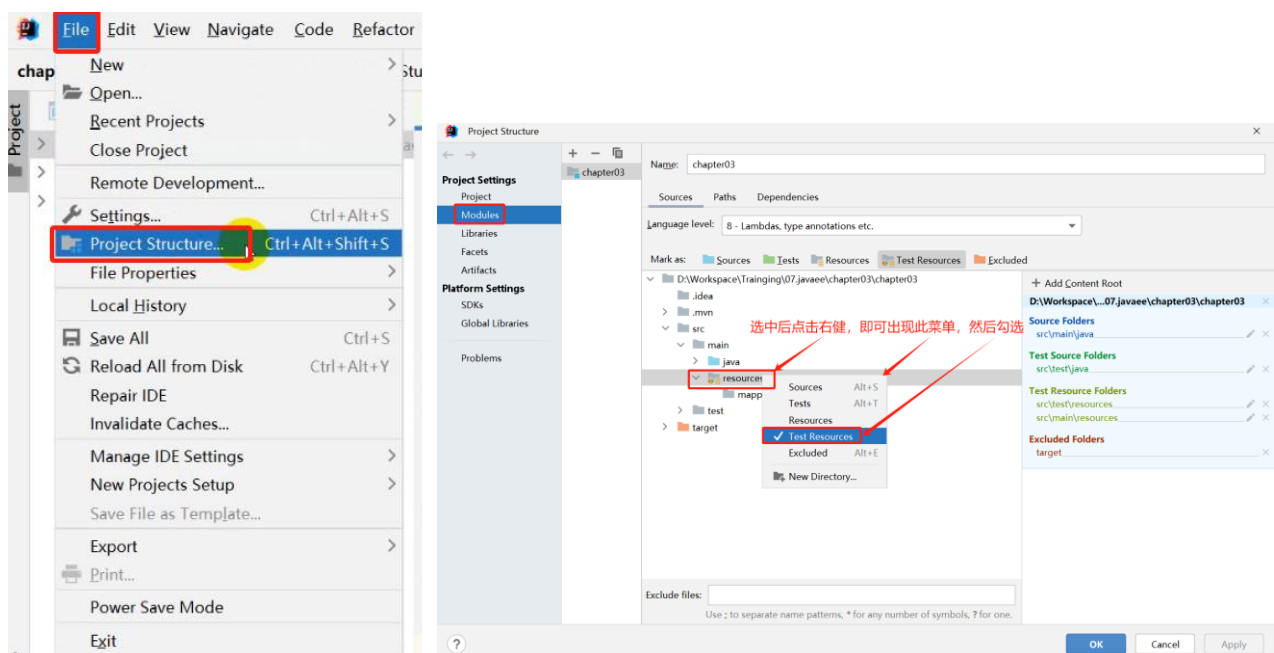
**解决办法 1**，就是手动将 src/main/resources/ 目录下的所有配置文件（包含目录）复制到

target/classes/目录下即可解决。(初学者建议采用此方法, 增加你对项目工程的熟悉度)



解决办法 2, 让 IDEA 自动编译复制, 步骤如下截图。配置好后, IDEA 编译时会自动复制

resource 目录中配置文件到编译后的目录, 就不需要再手工复制了。



注: 若 Test Resources 不生效, 可以换成勾选 Resources