

实验 9 实现用户登录实验程序

一、 实现用户登录需求描述

需求：通过所学的 Spring 数据库编程知识，实现学生管理系统的登录验证。同时验证注解实现数据库事务管理的功能。

功能点 1：通过使用 Spring 数据库编程，访问数据库验证用户名和密码是否输入正确，如果正确则显示用户**所属班级**，如果验证失败则显示**登录失败**。

功能点 2：使用**注解模式，配置数据库事务管理**。通过**修改学生所在班级名称**这个功能验证事务管理，包括**成功事务和失败事务回滚**。

思路分析：

功能 1：为验证用户名和密码是否正确，需要创建**学生表 t_student9** 存储学生信息，所需字段有：id, username, password, course。需要编写查询方法访问数据库，以获得验证信息。

功能 2：创建修改学生信息方法，根据学生 ID 修改学生的所在班级。

注意 1：程序开发遵循分层设计，包括 Service 层、Dao 层，调用链从 Service 层访问 Dao，再访问数据库，获取数据。

注意 2：需要使用注解模式进行 Bean 的配置与调用，需要配置注解模式的事务管理，注解事务写到 Service 层，具体的业务方法上加事务控制。

二、 开发准备

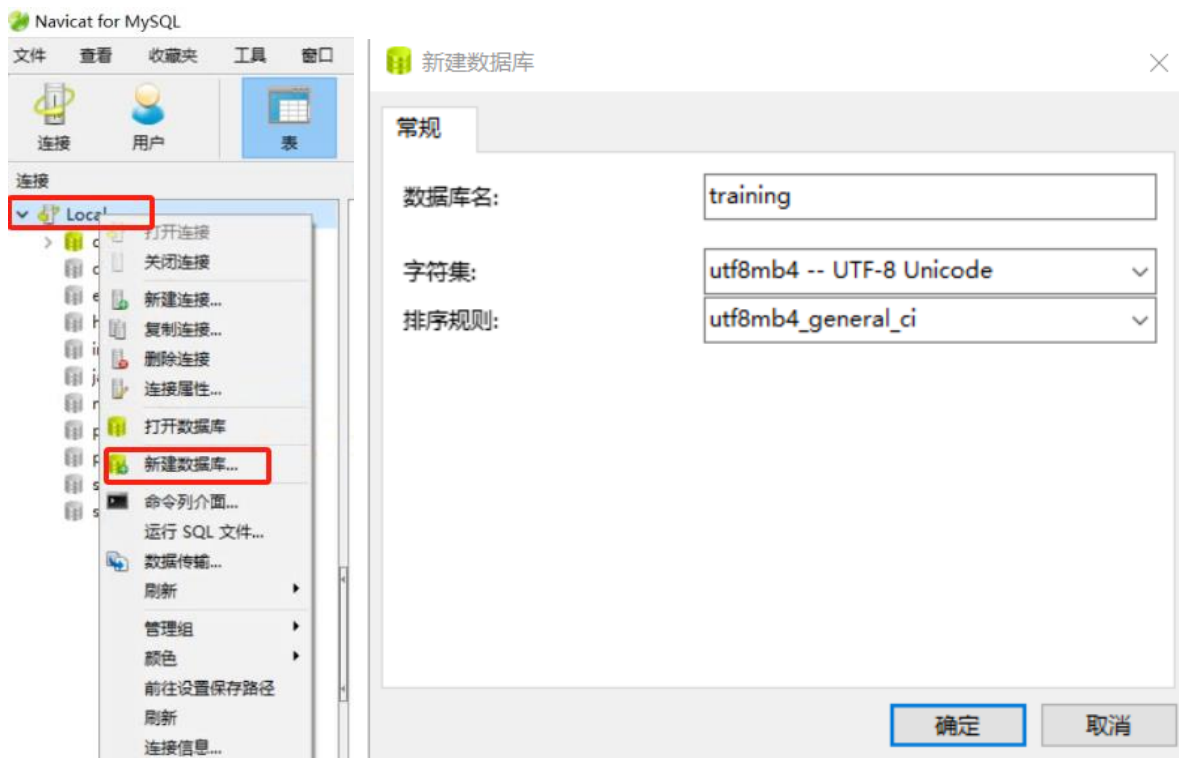
第 1 步：数据库开发

先使用 MySQL 数据库创建一个 **training** 数据库，然后创建一个学生表 **t_student9**，方法如下：

通过使用 Navicat 工具，**连接上你安装的 MySQL 数据库**，输入用户名和密码进行**登录**。

假设建立的 MySQL 连接名为 **Local**，数据库登录用户名和密码为：**root/123456**。

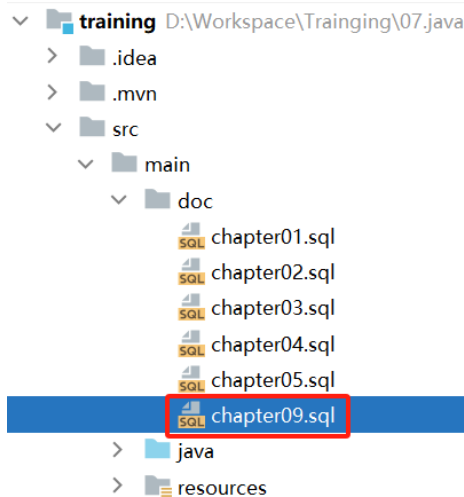
登录后，手动创建数据库，需要创建的数据库名为：**training**，如图：选中 **Local** 连接名，右键选择【新建数据库】



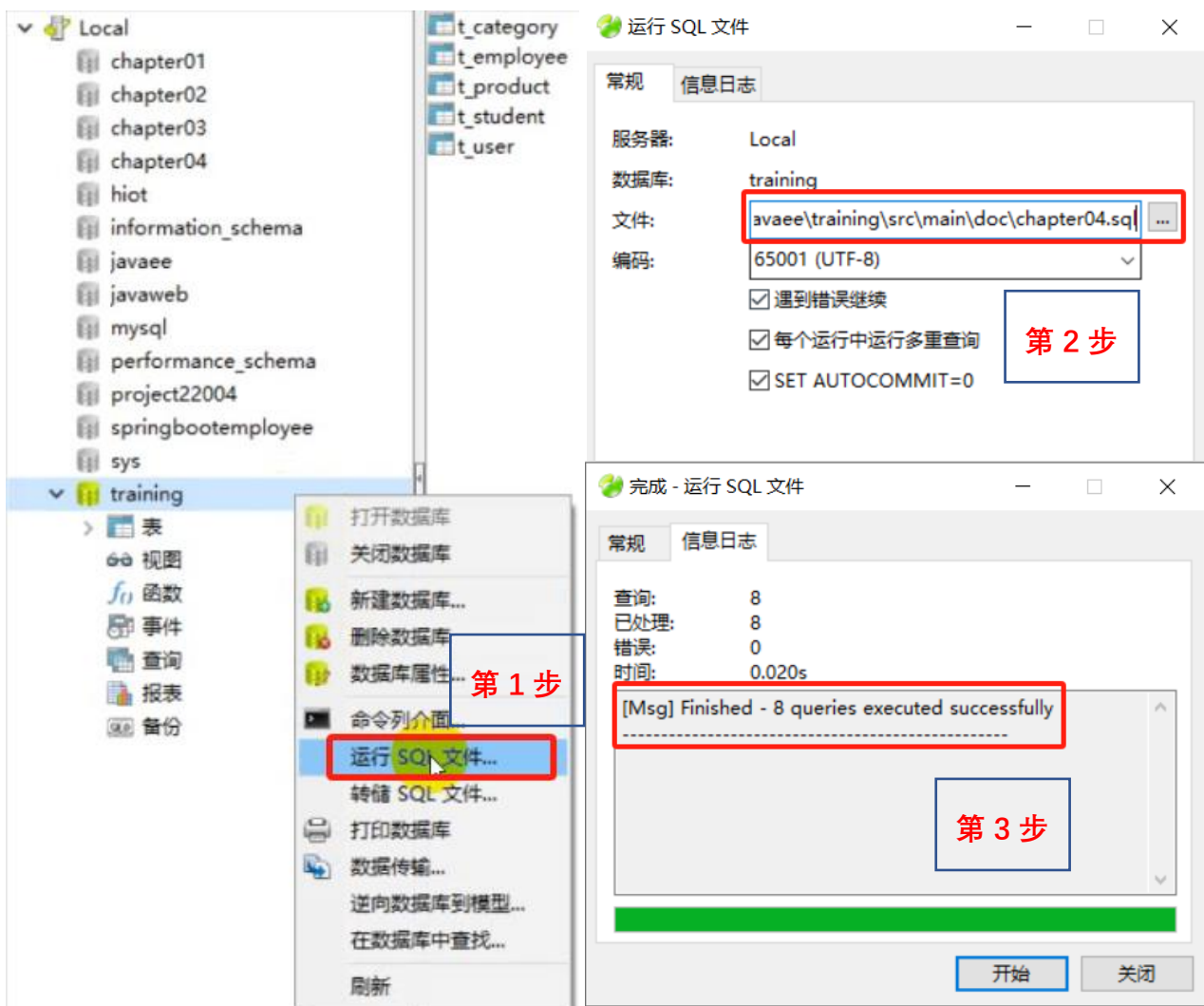
数据库创建好后，创建学生表 **t_student9**，表结构如下。并按要求加入测试数据。

```
1  USE training;
2  DROP TABLE IF EXISTS t_student9;
3  create table t_student9(
4      id int(11) primary key auto_increment,
5      username varchar(255),
6      password varchar(255),
7      course varchar(255)
8  );
9
10 insert into t_student9(id,username,password,course)
11 values(1,'zhangsan','123456','计算机2班'),
12        (3,'lisi','123456','软件工程3班'),
13        (4,'wangwu','123456','电子信息1班');
```

注：以上建表数据库脚本已放置工程的 **doc** 目录，文件路径如下图所示：



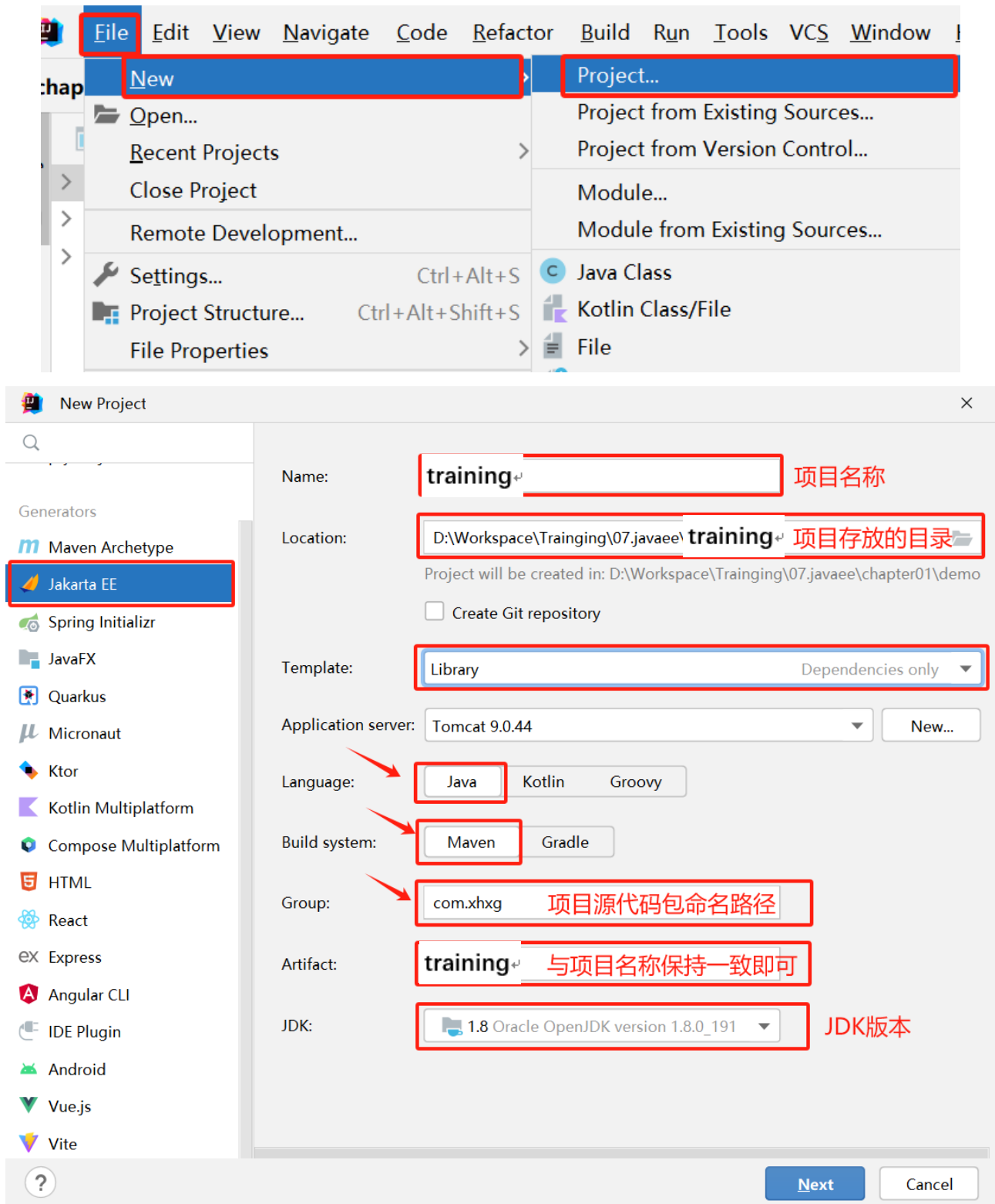
可以通过 Navicat 工具直接读取 chapter09.sql 脚本文件，以完成数据表的创建。

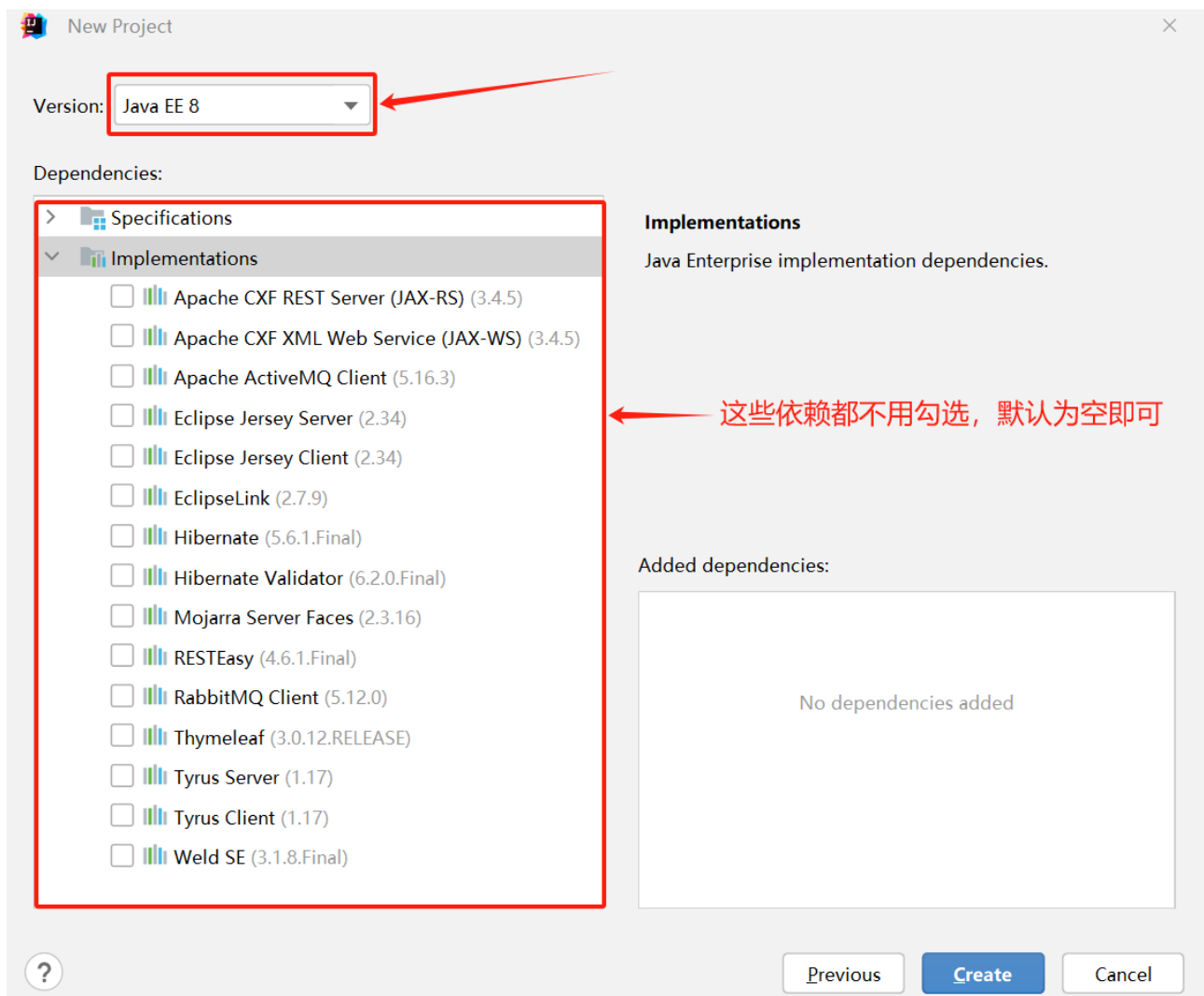


第 2 步：创建 Maven 工程

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）






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）

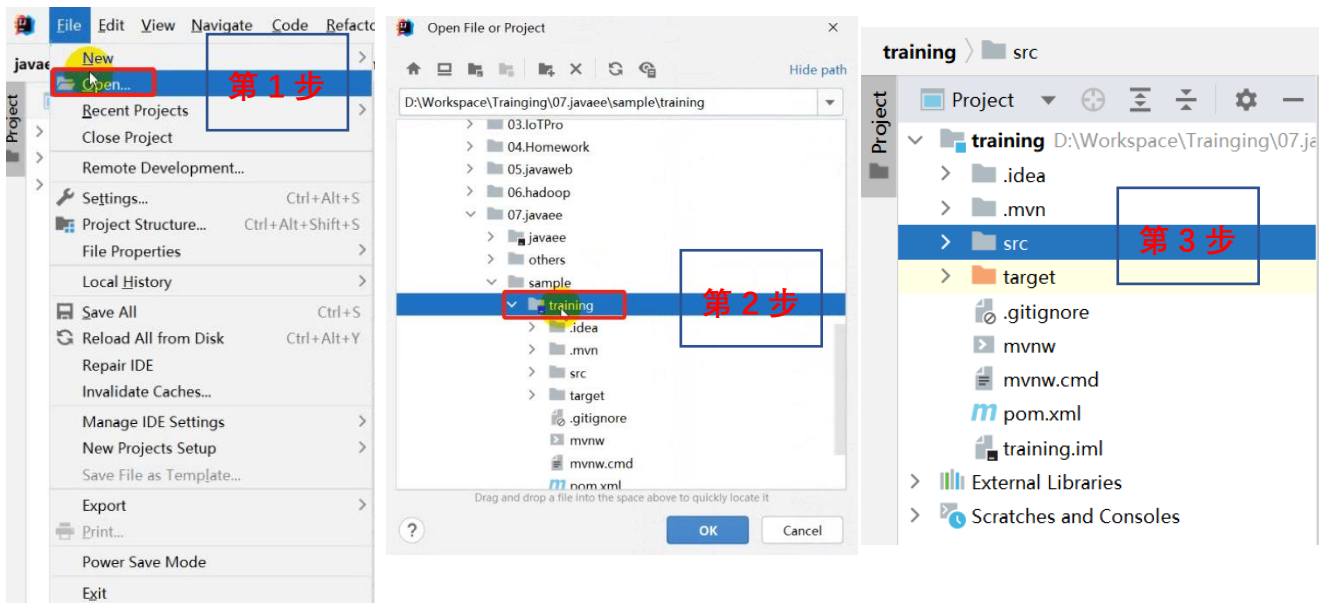
Step1，下载并解压 training 工程。从【学习通】→【章节】→【第九章…】→【上机实验 9…】，找到 **training.zip** 压缩包，下载到本地，并解压成 training 目录。

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。

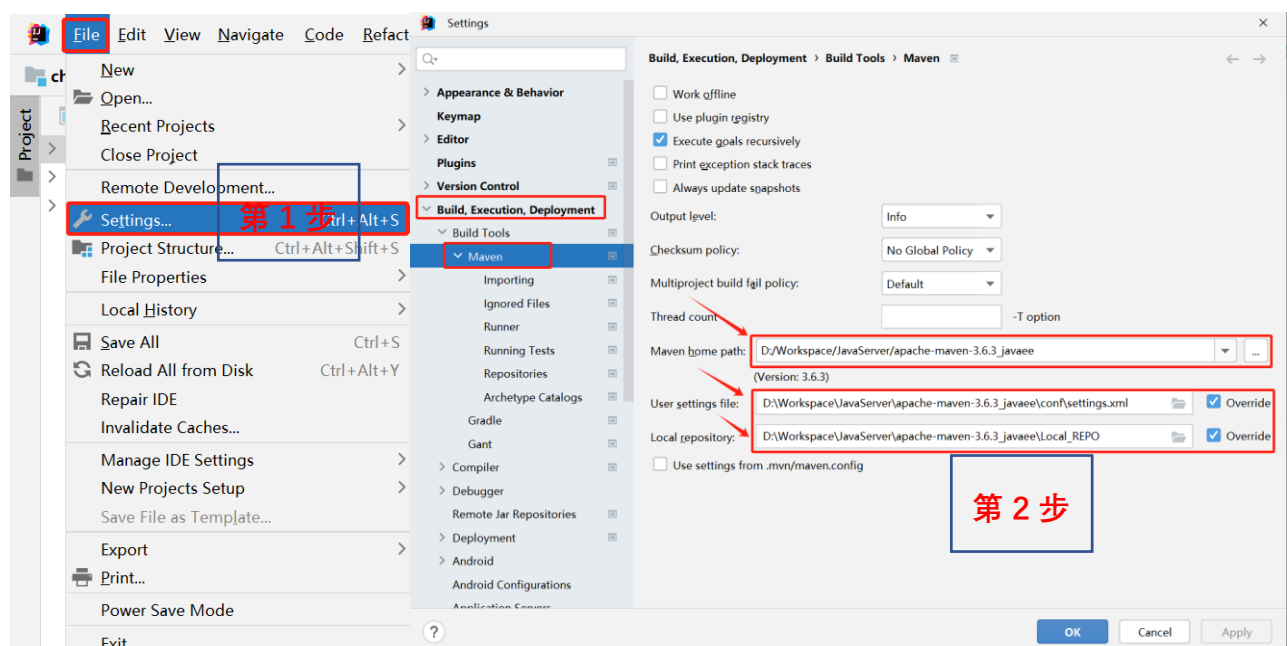
Tips：如果前几个章节已经下载并导入了此 **training** 工程，那么下载本章节最新工程后，

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 3 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。



三、 程序开发

第 1 步：配置 pom.xml

由于本章是在前 6、7、8 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：



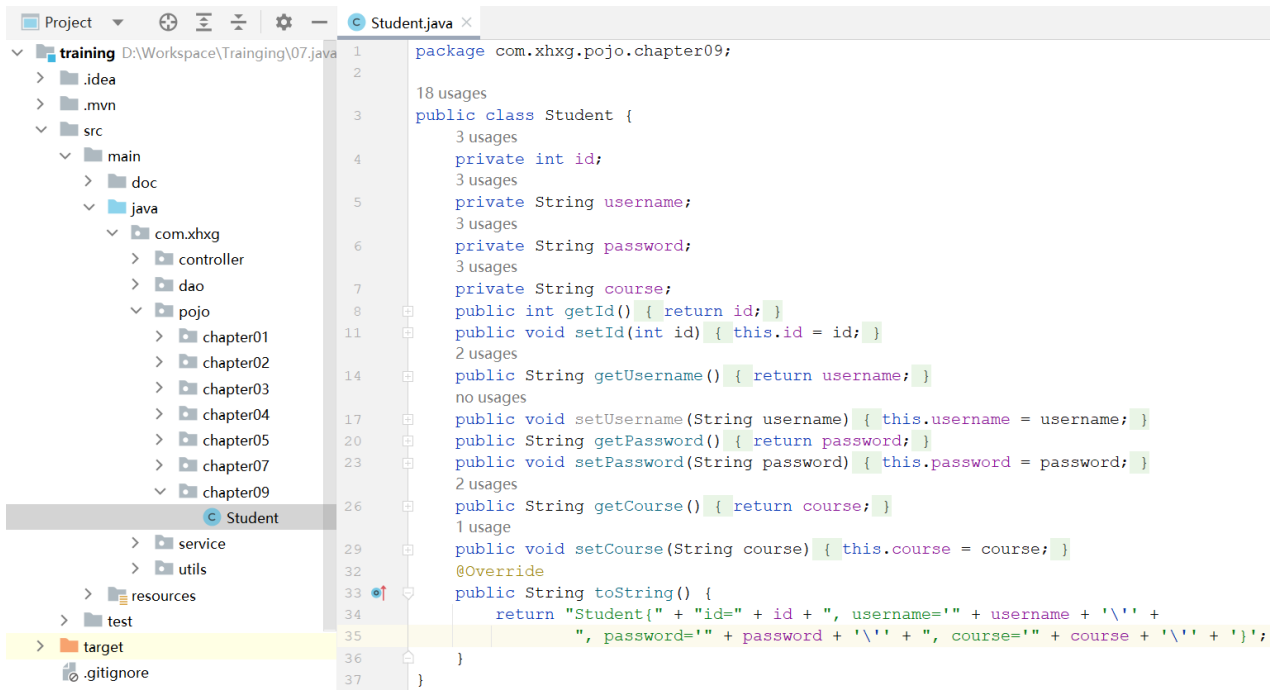
需要添加的依赖原配置为：

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

<dependency>
    <groupId>aopalliance</groupId>
    <artifactId>aopalliance</artifactId>
    <version>1.0</version>
</dependency>
```

第 2 步：创建 POJO 实体

打开 training 项目工程。创建代码包为：**com.xhxc.pojo.chapter09**，并且创建 **Student** 类，定义属性，使用 IDEA 快捷方式，创建 get/set 方法，toString 方法

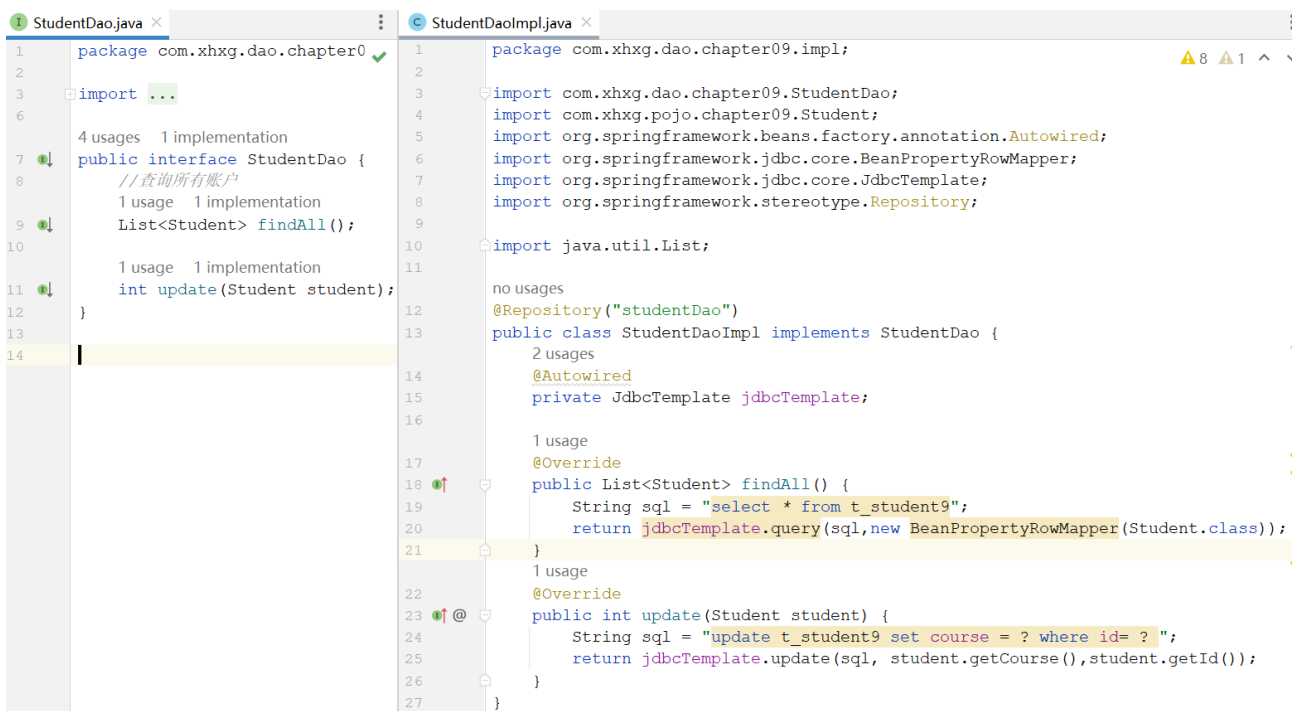


The screenshot shows the IntelliJ IDEA interface. On the left, the Project Explorer displays the directory structure: training > src > main > java > com.xhxc > pojo > chapter09. The 'Student' class is selected under chapter09. The main editor shows the code for Student.java:

```
1 package com.xhxc.pojo.chapter09;
2
3 public class Student {
4     private int id;
5     private String username;
6     private String password;
7     private String course;
8     public int getId() { return id; }
9     public void setId(int id) { this.id = id; }
10    public String getUsername() { return username; }
11    public void setUsername(String username) { this.username = username; }
12    public String getPassword() { return password; }
13    public void setPassword(String password) { this.password = password; }
14    public String getCourse() { return course; }
15    public void setCourse(String course) { this.course = course; }
16    @Override
17    public String toString() {
18        return "Student{" + "id=" + id + ", username='" + username + '\'' +
19            ", password='" + password + '\'' + ", course='" + course + '\'' + '}';
20    }
21 }
```

第 3 步：创建 Dao 接口及其实现类

创建代码包为：**com.xhxc.dao.chapter09**，创建 **StudentDao** 接口类及其实现类 **StudentDaoImpl**，定义 **findAll()** 和 **update()** 接口方法，代码如下截图所示：



The screenshot shows two files open in IntelliJ IDEA. The left file is StudentDao.java:

```
1 package com.xhxc.dao.chapter09;
2
3 import ...
4
5 public interface StudentDao {
6     // 查询所有账户
7     List<Student> findAll();
8
9     // 更新课程
10    int update(Student student);
11 }
12
```

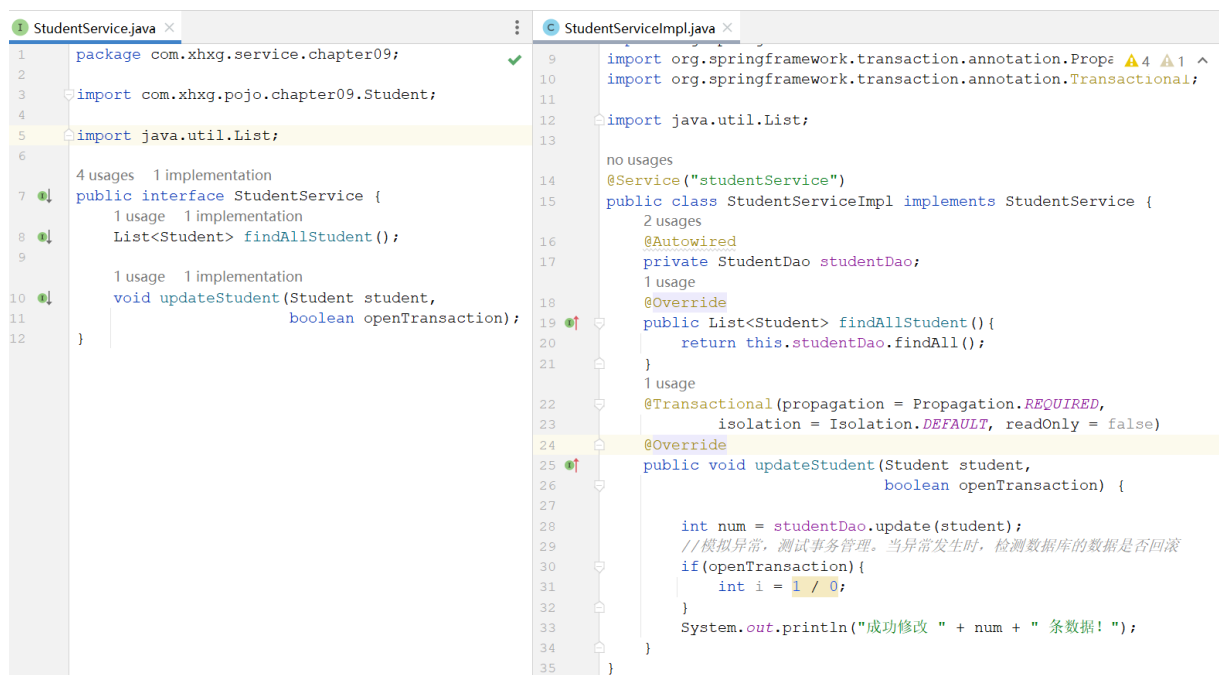
The right file is StudentDaoImpl.java:

```
1 package com.xhxc.dao.chapter09.impl;
2
3 import com.xhxc.dao.chapter09.StudentDao;
4 import com.xhxc.pojo.chapter09.Student;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.jdbc.core.BeanPropertyRowMapper;
7 import org.springframework.jdbc.core.JdbcTemplate;
8 import org.springframework.stereotype.Repository;
9
10 import java.util.List;
11
12 @Repository("studentDao")
13 public class StudentDaoImpl implements StudentDao {
14     @Autowired
15     private JdbcTemplate jdbcTemplate;
16
17     @Override
18     public List<Student> findAll() {
19         String sql = "select * from t_student9";
20         return jdbcTemplate.query(sql, new BeanPropertyRowMapper(Student.class));
21     }
22
23     @Override
24     public int update(Student student) {
25         String sql = "update t_student9 set course = ? where id = ?";
26         return jdbcTemplate.update(sql, student.getCourse(), student.getId());
27     }
28 }
```


注意：数据接口使用了注解@Repository("studentDao")，其中括号中的 studentDao 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

第 4 步：创建 Service 接口及其实现类

创建代码包为：com.xhxc.service.chapter09，创建 StudentService 接口类及其实现类 StudentServiceImpl，定义 findAllStudent() 和 updateStudent() 接口方法，代码如下截图所示：



```
1 StudentService.java
2 package com.xhxc.service.chapter09;
3 import com.xhxc.pojo.chapter09.Student;
4 import java.util.List;
5
6 4 usages 1 implementation
7 public interface StudentService {
8     1 usage 1 implementation
9     List<Student> findAllStudent();
10
11     1 usage 1 implementation
12     void updateStudent(Student student,
13                         boolean openTransaction);
14 }
15
16 StudentServiceImpl.java
17 import org.springframework.transaction.annotation.Propagation;
18 import org.springframework.transaction.annotation.Transactional;
19 import java.util.List;
20
21 no usages
22 @Service("studentService")
23 public class StudentServiceImpl implements StudentService {
24     2 usages
25     @Autowired
26     private StudentDao studentDao;
27
28     1 usage
29     @Override
30     public List<Student> findAllStudent() {
31         return this.studentDao.findAll();
32     }
33
34     1 usage
35     @Transactional(propagation = Propagation.REQUIRED,
36                     isolation = Isolation.DEFAULT, readOnly = false)
37     @Override
38     public void updateStudent(Student student,
39                             boolean openTransaction) {
40
41         int num = studentDao.update(student);
42         //模拟异常，测试事务管理。当异常发生时，检测数据库的数据是否回滚
43         if(openTransaction){
44             int i = 1 / 0;
45         }
46         System.out.println("成功修改 " + num + " 条数据!");
47     }
48 }
```

注意：业务层接口使用了注解@Service("studentService")，其中括号中的 studentService 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。

第 5 步：创建 Spring 配置文件 applicationContext.xml

在 src/main/resources 目录下创建一个 applicationContext-chapter09.xml 文件，内容如下：



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 `<tx:annotation-driven transaction-manager="transactionManager"/>` 事务配置，以开启注解事务管理。

第 6 步：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.chapter09`，然后创建测试类 `StudentTest`，定义测试方法 `findAllStudentTest()` 和 `updateStudentTest()`，代码如下：

```

StudentTest.java x
4  import com.xhxc.service.chapter09.StudentService;
5  import org.junit.Test;
6  import org.junit.runner.RunWith;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.test.context.ContextConfiguration;
9  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10 import java.util.List;
    no usages
11 @RunWith(SpringJUnit4ClassRunner.class)
12 @ContextConfiguration("classpath:applicationContext-chapter09.xml")
13 public class StudentTest {
    2 usages
14     @Autowired
15     StudentService studentService;
    no usages
16     @Test
17     public void findAllStudentTest() {
18         System.out.println("欢迎来到学生管理系统");
19         String name = "zhangsan";
20         String password = "123456";
21         List<Student> student = studentService.findAllStudent();
22         for (Student std : student) {
23             if (name.equals(std.getUsername()) && password.equals(std.getPassword())) {
24                 System.out.println("用户登录成功!");
25                 System.out.println(std.getUsername() + "是" + std.getCourse() + "班的");
26                 break;
27             } else {
28                 System.out.println("用户登录失败, 账号密码错误!");
29                 break;
30             }
31         }
32     }

34     //事务测试
    no usages
35     @Test
36     public void updateStudentTest() {
37
38         //是否开启事务测试
39         boolean isOpenTransaction = false;
40
41         Student student = new Student();
42         student.setId(1);
43         //student.setCourse("计算机1班");
44         student.setCourse("大数据2班");
45         studentService.updateStudent(student, isOpenTransaction);
46     }
47 }

```

在本单元测试中，通过注解读取 Spring 配置文件 **applicationContext-chapter09.xml**，并且使用自动注解的方式读取 Bean 实例，代码如下所示。

需要注意注解事务的配置方法和验证方式。

配置方法：需要在配置使用事务的方法上加入注解@Transactional()，以表明此方法使用事务管理，另外，还需要在 Spring 配置文件中配置事务驱动器<tx:annotation-driven transaction-manager="transactionManager"/>，以开启注解事务管理

```
//<tx:annotation-driven transaction-manager="transactionManager"/>需要开启此事务驱动器才生效
1 usage
@Transactional(propagation = Propagation.REQUIRED,isolation = Isolation.DEFAULT, readOnly = false)
@Override
public void updateStudent(Student student, boolean openTransaction) {
    int num = studentDao.update(student);
    //模拟异常，测试事务管理。当异常发生时，检测数据库的数据是否回滚
    if(openTransaction){
        int i = 1 / 0;
    }
    System.out.println("成功修改 " + num + " 条数据!");
}
```

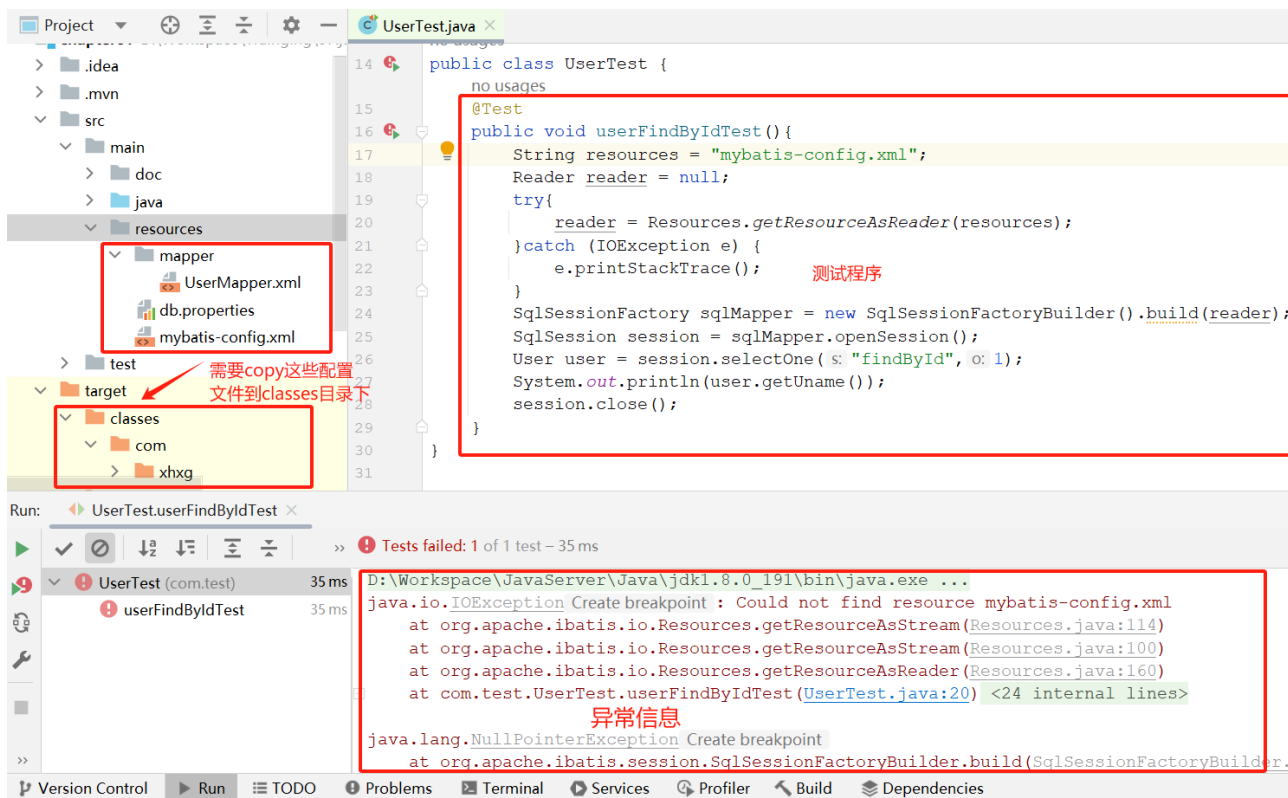
验证方法：主要是模拟程序调用出错。注意看 updateStudentTest() 测试方法，调用的 updateStudent() 方法中有一个参数 boolean openTransaction，这个是开启模拟出错的开关，**当为 true 时**，程序调用会出错，此时运行测试程序，观察数据库中数据的变化，是否程序出错后，数据是否修改成功？

还有就是，将事务管理关闭掉（比如把注解的@Transactional()注释掉），然后再启用模拟程序出错，此时再运行测试程序，观察数据库中的数据变化，是否程序出错后，数据是否修改成功。比较这两个测试的区别在哪里。这样就能理解事务管理控制的作用。

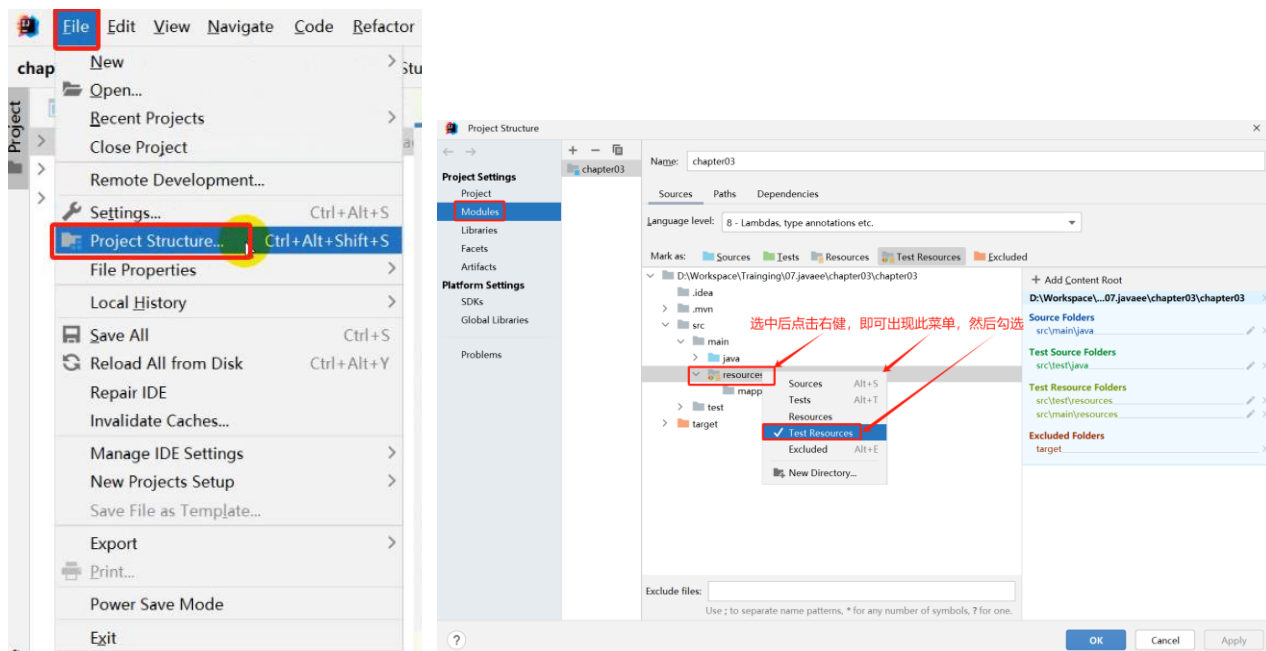
三、常见问题

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