山东大学 计算机科学与技术 学院

现代软件开发技术 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：2020001300 | 姓名： | | 班级： |
| 实验题目：Spring Boot Data JPA 多对多，排序和分页数据，jdbc template, Mybatis 访问数据库 | | | |
| 实验学时：2学时 | | 实验日期： 2023-11-21 | |
| 实验目的：  1.掌握使用Spring Data JPA 多对多的数据库关联查询  2.会使用@Query和@Modifying注解  3.会使用Spring Data JPA中提供的排序和分页数据支持前端页面分页输出  4.在Spring Boot中，会使用Jdbc Template访问数据库  5.能在Spring Boot中，使用Mybatis持久层框架访问数据库 | | | |
| 硬件环境： | | | |
| 软件环境：  操作系统：windows 10  开发平台与工具：IDEA开发平台、jdk-11.0.1、apache-tomcat-9.0.37 | | | |
| 实验内容与问题回答：  1.练习在Spring Data JPA中使用@ManyToMany注解多对多的映射关系（用户与权限的多对多关系映射）  （0）项目ch6\_2。打开src/main/resources下的application.properties位置文件，修改其中的数据库连接参数。  （1）分析持久化实体类  打开src/main/java下的com.ch.ch6\_2.entity包中的User.java和Authority.java实体类，分析其中的代码，  Q: 注解@JoinTable在哪个实体类中？其各个属性代表的含义是什么？与前例相比还存在@JoinColumn注解吗？    在上述代码中，@JoinTable 注解用于配置多对多关系的关联表信息。以下是各个属性的含义：  name：指定关联表的名称，即存储多对多关系的中间表的表名。在这里，它指定了关联表的名称为 "user\_authority"。  joinColumns：配置关联表中关联到当前实体（User）的外键列信息。joinColumns属性用于指定关联表中的列，它是一个数组，每个元素表示一个外键。在这里，它指定了一个外键列 "user\_id"，该外键关联到 User 表的主键列。  inverseJoinColumns：配置关联表中关联到目标实体（Authority）的外键列信息。inverseJoinColumns属性用于指定关联表中的列，它也是一个数组，每个元素表示一个外键。在这里，它指定了一个外键列 "authority\_id"，该外键关联到 Authority 表的主键列。  Q: mappedBy属性在哪个实体类中？它的值对应的是什么？  通过观察代码发现mappedBy属性在Authority这个实体类中；  它的值对应的是："authorityList"。如下图所示：    Q: 创建的关联表名是？主表、从表分别是？  在这两个实体类的代码中，通过 @JoinTable 注解定义了关联表，它的名字是 "user\_authority"。这个注解出现在 User 实体类的 authorityList 属性上  关联表名：user\_authority  主表（关系维护端）：user  从表（关系被维护端）：authority  在这里，User 是关系的维护端，因此被认为是主表。而 Authority 是关系的被维护端，被认为是从表。  Q: 实体类中表示关联的属性分别是？  User 实体类中，表示关联的属性是 private List<Authority> authorityList;，它通过 @ManyToMany 注解与 Authority 实体类建立了多对多关系。  Authority 实体类中，表示关联的属性是 private List<User> userList;，它通过 @ManyToMany(mappedBy = "authorityList") 注解指定了关联关系的维护端，这里是 User 实体类。  （2）数据访问层  打开src/main/java下的com.ch.ch6\_2.repository包中的UserRepository.java和AuthorityRepository.java，分析接口中的代码，  细读每个方法所对应的注释中的JPQL命令。  Q：AuthorityRepository.java中为什么使用@Query注解？注解下的抽象方法名哪里不符合命名规范？  在 AuthorityRepository.java 中使用 @Query 注解的原因是，它提供了一种使用自定义的JPQL（Java Persistence Query Language）语句来查询数据库的方式。有时候，通过自定义JPQL语句可以更灵活地执行一些复杂的查询，而不仅限于Spring Data JPA默认的方法命名规范。  （3）业务层  打开src/main/java下的com.ch.ch6\_2.service包中的UserAndAuthorityService.java和UserAndAuthorityServiceImpl.java，分析其代码。  Q：其中为什么有如下两个功能一样的方法？  public List<Authority> findByUserList\_Username(String username);  public List<Authority> findByUserListUsername(String username);  在Spring Data JPA中，方法的命名是有规范的，它们会根据方法名来解析生成查询。在这两个方法中：  findByUserList\_Username(String username)：这个方法的命名符合Spring Data JPA默认的命名规范。它表示通过User实体类中的userList属性（即@ManyToMany关联关系）中的Username属性进行查询。Spring Data JPA会自动解析这个方法名，并生成相应的查询语句。  findByUserListUsername(String username)：这个方法使用了@Query注解，表明使用自定义的JPQL语句进行查询。这种方式允许更灵活的查询，并且不受默认命名规范的限制。在这里，它可能是为了演示或者实现一些特定的查询需求。  Q:在UserAndAuthorityServiceImpl.java的saveAll()方法中，先保存的是哪个表？User的实例对象与Authority的实例对象是怎样建立多对多关联的？  在UserAndAuthorityServiceImpl.java的saveAll()方法中，先保存的是Authority表。  多对多关联关系的建立如下：  1.创建Authority实例对象：  Authority at1 = new Authority();  at1.setName("增加");  authorityRepository.save(at1);  2.创建User实例对象：  User u1 = new User();  u1.setUsername("陈恒1");  u1.setPassword("123");  3.创建关联关系：  ArrayList<Authority> authorityList1 = new ArrayList<Authority>();  authorityList1.add(at1);  authorityList1.add(at2);  authorityList1.add(at3);  u1.setAuthorityList(authorityList1);  4.保存User实例对象，这时会级联保存关联的Authority实例对象：  userRepository.save(u1);  这样，通过设置User实例对象的authorityList属性，同时保存了User和与其关联的Authority之间的多对多关系。这是通过@ManyToMany注解和关联表实现的。  （4）控制器类  打开src/main/java下的com.ch.ch6\_2.controller包中的TestManyToManyController.java，分析其中的代码。  分析@RequestMapping注解对应的方法名及其参数,  Q:方法中调用的哪个层的实例对象方法？  方法调用的是业务层或者说Service层的实例对象方法。  （5）运行  先关闭之前运行的项目。然后运行src/main/java下com.ch.ch6\_2包中Ch62Application.java (run as Java Application)，  观察控制台窗口中显示的关联表user\_authority的foreign key的信息有哪几条    （a）在浏览器地址栏输入：http://localhost:8080/ch6\_2/saveManyToMany/，保存数据（创建表）。    用外部可视化工具(如：Navicat for MySQL）打开MySQL中springdatajpa数据库,看其中是否创建了以下三张表：  user、authority和user\_authority，  Q：其中的属性都有哪些？记录有几条？外键的值是否都已填入？  可以看到已经创建了上述的三张表    其中的属性和记录如下图所示，并且可以看到外键的值都已填入。        （b）输入：http://localhost:8080/ch6\_2/findByAuthorityList\_id?id=1，关联查询权限id为1（id的值需要依据实际数据库中的值）的用户信息。    （c）输入：http://localhost:8080/ch6\_2/findByAuthorityList\_name?name=修改，关联查询权限名为修改的用户信息。    （d）在浏览器地址栏输入：http://localhost:8080/ch6\_2/findByUserList\_id?id=2，关联查询用户id为2（id的值需要依据实际数据库中的值）的权限列表。    （e）输入：http://localhost:8080/ch6\_2/findByUserList\_Username?username=陈恒2，关联查询用户名为陈恒2的权限列表。    （f）输入：http://localhost:8080/ch6\_2/findByUserListUsername?username=陈恒2，Q:查询结果是否与步骤(e)一样？    一致  2.练习使用@Query和@Modifying注解,加入JPQL语句。  （0）项目ch6\_3。打开src/main/resources下的application.properties位置文件，修改其中的数据库连接参数。  （1）分析持久化实体类（作者和文章，一对多）  打开src/main/java下的com.ch.ch6\_3.entity包中的Author.java和Article.java实体类，分析其中的代码，同前。  （2）数据访问层  打开src/main/java下的com.ch.ch6\_3.repository包中的AuthorRepository.java和ArticleRepository.java，分析其代码。  分析@Query注解的JPQL命令，  Q:其中的方法符合命名规范吗？  **对于AuthorResposity.java来说：**  分析@Query注解的JPQL命令：  1.findAuthorByArticleListtitleContaining 方法：  @Query("select a from Author a inner join a.articleList t where t.title like %?1%")  根据文章标题包含的内容，查询作者。使用了关联查询，通过文章列表关联到作者表，条件是文章标题包含指定内容。  2．deleteAuthorByAuthorId 方法：  @Modifying：标识该查询是修改操作，需要使用@Modifying注解。  @Query("delete from Author a where a.id = ?1")  **对于ArticleReposity.java来说：**  分析 @Query 注解的 JPQL 命令：  1.findTitleAndContentByAuthorId 方法：  @Query("select new Map(a.title as title, a.content as content) from Article a where a.author.id = ?1 ")  根据作者id查询文章信息（标题和内容）。使用了 new Map 投影，将查询结果映射为 Map 对象。  2.findArticleByAuthorAnameAndId 方法：  @Query("select a from Article a where a.author.aname = :aname1 and a.author.id = :id1 ")  Q:@Modifying注解附加在哪个抽象方法之上？对应的JPQL指令是？  如下图所示，@Modifying注解附加在了deleteAuthorByAuthorId方法上；对应的JPQL指令是：    （3）业务层  打开src/main/java下的com.ch.ch6\_3.service包中的AuthorAndArticleService.java和AuthorAndArticleServiceImpl.java，分析其代码。  Q:业务层中方法名符合规范吗？为什么不需要@Query注解？  在 Spring Data JPA 中，如果方法名符合特定的命名规范，就不需要使用 @Query 注解，因为 Spring Data JPA 会根据方法名自动生成查询。以下是对业务层方法的分析：  findTitleAndContentByAuthorId: 方法名符合 Spring Data JPA 的命名规范，该方法用于根据作者id查询文章的标题和内容。  findArticleByAuthorAnameAndId: 方法名同样符合规范，用于根据作者名和作者id查询文章信息。  findAuthorByArticleListtitleContaining: 方法名符合规范，用于根据文章标题的内容查询作者。  deleteAuthorByAuthorId: 方法名符合规范，用于根据作者id删除作者。  （4）控制器类  打开src/main/java下的com.ch.ch6\_3.controller包中的TestOneToManyController.java，分析其中的代码。  注意几个注解@RequestMapping对应的方法名及其参数，  Q:代码中注入的实例是？  如下图所示，代码中注入的实例是：AuthorAndArticleService类的一个实例；    （5）运行  关闭之前运行的项目。运行src/main/java下com.ch.ch6\_3包中Ch63Application.java(run as Java Application)，  （a）输入：http://localhost:8080/ch6\_3/findTitleAndContentByAuthorId?id=1，关联查询作者id为1（id的值需要依据实际数据库中的值）的文章列表。    （b）输入：http://localhost:8080/ch6\_3/findArticleByAuthorAnameAndId?aname=陈恒2&id=2，关联查询作者名为陈恒2且作者id为2（id的值需要依据实际数据库中的值）的文章列表。    （c）输入：http://localhost:8080/ch6\_3/AuthorBArticleListtitleContain?title=对多1，关联查询文章标题含“对多1”的作者。    （d）输入：http://localhost:8080/ch6\_3/deleteAuthorByAuthorId?id=1，删除文章id为1（id的值需要依据实际数据库中的值）的作者    3.练习Spring Data JPA中使用排序和分页查询  （0）项目ch6\_4。打开src/main/resources下的application.properties 项目属性文件，修改其中的数据库连接参数。  （1）分析持久化实体类（作者和文章，一对多联系）  打开src/main/java下的com.ch.ch6\_4.entity包中的Author.java和Article.java实体类，分析其中的代码，同前。  用工具（如MySQL Workbench或Navicat）打开springbootjpa数据库中author\_table和article\_table,查看数据是否正确。如无数据，请运行ch6\_2项目，发请求 http://localhost:8080/ch6\_2/saveOneToMany/，建立、存储数据。      （2）数据访问层  打开src/main/java下的com.ch.ch6\_4.repository包中的AuthorRepository.java，分析其代码。  Q:其方法中如何使用Sort类形参  在AuthorReposity.java文件的代码中，方法 findByAnameContaining 中通过 Sort 类型的参数 sort 来指定排序规则。该参数允许在查询结果中应用排序。  （3）业务层  打开src/main/java下的com.ch.ch6\_4.service包中的ArticleAndAuthorService.java和ArticleAndAuthorServiceImpl.java，分析其代码。  Q: 方法中如何使用形参 sortColum;  在这两个文件中，形参 sortColum 被用于指定排序的字段。具体来说：  1.在 findByAnameContaining 方法中，通过创建 Sort 对象，指定了按照 sortColum 字段的降序排序。这个 Sort 对象被传递给了 authorRepository.findByAnameContaining 方法，从而在查询时应用了指定的排序规则。  return authorRepository.findByAnameContaining(aname, new Sort(Direction.DESC, sortColum));  这样，通过在调用该方法时传递不同的 sortColum 参数，可以实现根据不同字段进行降序排序。  2.在 findAllAuthorByPage 方法中，通过创建 Sort 对象，指定了按照 "id" 字段的降序排序。这个 Sort 对象被传递给了 PageRequest.of 方法，从而在分页查询时应用了指定的排序规则。  Page<Author> pageData = authorRepository.findAll(PageRequest.of(page-1, size, new Sort(Direction.DESC, "id")));  Q: 使用的pageData对象方法都有哪些？  在上述代码中，使用的 pageData 对象的方法包括：  1.getContent()：获取当前页面的内容作为列表。  List<Author> allAuthor = pageData.getContent();  2.getTotalElements()：获取数据集中的总数。  model.addAttribute("totalCount", pageData.getTotalElements());  3.getTotalPages()：获取总页数。  model.addAttribute("totalPage", pageData.getTotalPages());  Q: 加入model中的属性变量有哪些？  在上述代码中，向 Model 中添加的属性变量包括：  1."allAuthor"：当前页面的作者列表。  model.addAttribute("allAuthor", allAuthor);  2."totalPage"：总页数。  model.addAttribute("totalPage", pageData.getTotalPages());  3."page"：当前页的页码。  model.addAttribute("page", page);  （4）控制器类  打开src/main/java下的com.ch.ch6\_4.controller包中的TestSortAndPage.java，分析其中的代码。  注意几个@RequestMapping注解 对应的方法名及其参数。  （5）视图层  这些代码的目的是在页面上展示查询结果的表格，并提供上一页和下一页的链接，使用户能够浏览不同页的查询结果。 Thymeleaf表达式 th:text 用于将后端数据显示在页面上，而 th:if 用于根据条件控制元素的显示。  Q:index.html中的allAuthor是查询到的全部作者记录吗？  是  （6）运行  先关闭之前运行的项目。然后运行src/main/java下com.ch.ch6\_4包中Ch64Application.java(run as Java Application)，  （a）输入：http://localhost:8080/ch6\_4/findByAnameContaining?aname=陈&sortColum=id , 查看结果。    （b）在浏览器地址栏输入：http://localhost:8080/ch6\_4/findAllAuthorByPage ，分页查询。    4.练习在Spring Boot中使用Jdbc Template访问数据库。  打开课本项目ch6\_5。  （1）分析pom.xml文件，  Q:与之前的项目如ch6\_4相比，其中的依赖有什么不同？  加入了jdbc依赖。s  （2）打开src/main/resources下的application.properties配置文件，修改其中的连接数据库地址、参数。  Q:与之前的项目相比，配置有什么不同？  ch6\_5项目没有了JPA持久化相关的一些配置。  （3）分析持久化实体类  打开src/main/java下的com.ch.ch6\_5.entity包中的MyUser.java实体类，分析其中的代码，  Q:与之前的JPA项目的实体类相比，有什么不同？能看出其对应的表吗？  1.注解差异：  在第一个JPA项目的实体类（Author）中，使用了来自javax.persistence.Entity的@Entity注解。  在第二个JPA项目的实体类（MyUser）中，没有使用@Entity注解。看起来这个实体缺少JPA注解。  2.表名差异：  在第一个JPA项目的实体类（Author）中，@Table注解指定了表名为 "author\_table"。  在第二个JPA项目的实体类（MyUser）中，没有@Table注解，因此将使用默认表名。  3.字段差异：  Author实体类通过articleList字段与另一个实体类（Article）建立了关系（@OneToMany）。  MyUser实体类没有定义任何关系。  4.缺失的JPA注解：  MyUser实体缺少关键的JPA注解，如@Entity和@Table，这些对于JPA识别和映射实体到数据库表是必要的。  根据这些差异，很明显Author实体对应于 "author\_table"，但由于缺少JPA注解，无法确定MyUser实体对应的表。  （4）数据访问层  打开src/main/java下的com.ch.ch6\_5.repository包中的MyUserRepository.java和MyUserRepositoryImpl.java，分析其中代码。  Q：在接口实现文件中的几个方法实现中，是用什么方式利用JdbcTemplate对象的什么方法实现数据库交互的？其中的查询方法分别调用了JdbcTemplate的什么方法？  在上述接口实现文件中，使用了JdbcTemplate对象进行数据库交互。以下是几个方法的实现方式：  saveUser 方法：  使用 SQL 语句：insert into user (username, password) values (?,?)。  通过 Object 数组传递参数。  使用 jdbcTemplate.update(sql, args) 方法执行 SQL 更新操作。  deleteUser 方法：  使用 SQL 语句：delete from user where id = ?。  通过 Object 数组传递参数。  使用 jdbcTemplate.update(sql, args) 方法执行 SQL 删除操作。  updateUser 方法：  使用 SQL 语句：update user set username = ?, password = ? where id = ?。  通过 Object 数组传递参数。  使用 jdbcTemplate.update(sql, args) 方法执行 SQL 更新操作。  与MyUser对象对应的表的名称？  与 MyUser 对象对应的表的名称是 "user"。这是通过 @Table(name = "user") 注解指定的。  Q：对比ch6\_1至ch6\_4的数据访问层，为何它们不需要接口的实现程序\*Impl.java？对比两种方式的特点？  在前几个项目中，使用的是Spring Data JPA框架，该框架会自动根据接口的方法名生成查询语句，无需手动编写实现类。Spring Data JPA框架会在运行时动态生成实现类，称为动态代理。这样可以减少开发者的工作量，提高代码的简洁性。  在本项目（ch6\_5）中，使用的是JdbcTemplate，这是Spring提供的对JDBC的简化封装。JdbcTemplate不具备Spring Data JPA那样的动态代理功能，需要手动编写实现类来完成数据库的交互操作。  Spring Data JPA框架提供了强大的自动查询功能，无需手动实现大部分的数据访问层方法。这样可以提高开发效率，减少重复代码。  JdbcTemplate需要手动编写SQL语句和实现数据访问层方法，相对繁琐，但更灵活，可以完全自定义SQL语句。  Spring Data JPA框架更适合简单的CRUD操作，而JdbcTemplate适合复杂的SQL操作或者需要更灵活控制的场景。  因此，选择使用Spring Data JPA还是JdbcTemplate取决于项目的需求和开发者的习惯。  （5）业务层  打开src/main/java下的com.ch.ch6\_5.service包中的MyUserService.java和MyUserServiceImpl.java，分析其代码。  Q：为什么要分业务层和数据访问层？  分离业务层和数据访问层是一种常见的软件设计模式，被称为分层架构。这种设计模式有多个优势，包括：  模块化和可维护性： 通过分离业务逻辑和数据访问逻辑，代码分为不同的层次，使得每个层次都专注于特定的任务。这提高了代码的模块化程度，降低了代码的复杂性，使得代码更易于理解和维护。  可测试性： 分层架构使得业务逻辑和数据访问逻辑能够独立地进行单元测试。业务层的逻辑可以通过模拟数据访问层的行为进行测试，而数据访问层可以通过模拟数据库或其他数据源进行测试。这提高了代码的可测试性，有助于确保代码的质量。  可扩展性： 当应用需要变更或扩展时，可以更容易地对其进行修改。例如，可以更容易地更改数据  （6）控制器类  打开src/main/java下的com.ch.ch6\_5.controller包中的MyUserController.java，分析其中的代码。  Q:几个@RequestMapping 对应的方法名，其参数从哪里传入？返回的数据形式是？  RequestMapping("/saveUser") 对应的方法：  方法名：saveUser  参数来源：通过POST请求，请求体中传入MyUser对象，Spring MVC将请求体中的数据绑定到MyUser对象的相应属性上。  返回的数据形式：返回一个整数，可能表示插入记录的状态或其他信息。  RequestMapping("/deleteUser") 对应的方法：  方法名：deleteUser  参数来源：通过GET或POST请求，请求参数中传入Integer类型的id，Spring MVC将请求参数绑定到方法的id参数上。  返回的数据形式：返回一个整数，可能表示删除记录的状态或其他信息。  RequestMapping("/updateUser") 对应的方法：  方法名：updateUser  参数来源：通过POST请求，请求体中传入MyUser对象，Spring MVC将请求体中的数据绑定到MyUser对象的相应属性上。  返回的数据形式：返回一个整数，可能表示更新记录的状态或其他信息。  （7）运行  关闭之前运行的项目。然后运行src/main/java下com.ch.ch6\_5包中Ch65Application.java(run as Java Application)，  （a）在浏览器地址栏输入：http://localhost:8080/ch6\_5/saveUser?username=陈恒3&&password=123456，保存数据（创建新用户陈恒3）。  注意网页中显示的返回信息：1    用外部可视化工具(如：Navicat for MySQL）打开MySQL中springdatajpa数据库,看其中user表中是否增加了新用户？    （b）在浏览器地址栏输入：http://localhost:8080/ch6\_5/deleteUser?id=1，删除用户id=1的用户信息。（id应对应数据表中的值）    （c）输入：http://localhost:8080/ch6\_5/updateUser?id=2&&username=陈恒222&&password=888888，修改用户id=2的用户信息。    （d）在浏览器地址栏输入：http://localhost:8080/ch6\_5/findAll，查询所有用户列表。  查看相应的用户是否已删除、修改、添加？    Q:此例与前面使用JPA的项目相比，业务代码与SQL是否纠缠在了一起？  在这个例子中，业务代码中的数据访问部分与具体的 SQL 操作确实有所纠缠，因为使用了Spring JDBC（JdbcTemplate）来直接执行 SQL 语句。相比前面使用JPA的项目，JPA 是一个对象关系映射（ORM）框架，能够将实体对象与数据库表进行映射，通过对象的方式来进行数据库操作，更加抽象化和面向对象。  5. 练习在Spring Boot中使用Mybatis持久层框架访问数据库  打开课本项目ch6\_6。  （1）分析pom.xml文件，  Q:与之前的项目如ch6\_4、ch6\_5相比，其中的依赖有什么不同？  增加了mybatis依赖。  （2）打开src/main/resources下的application.properties配置文件，修改其中的连接数据库地址、参数。  Q:以下两条的含义？  mybatis.type-aliases-package=com.ch.ch6\_6.entity  mybatis.mapperLocations=classpath:mappers/\*.xml  这两条语句是配置 MyBatis 框架的配置文件中的内容，用于告诉 MyBatis 框架关于类型别名和 XML 映射文件位置的信息。  mybatis.type-aliases-package=com.ch.ch6\_6.entity：  这个配置用于指定 MyBatis 在扫描时要注册的类型别名的包路径。MyBatis 中的类型别名是为 Java 类型指定一个短名字，以便在 XML 映射文件中更方便地引用。  在这个例子中，指定了包路径 com.ch.ch6\_6.entity，MyBatis 将扫描这个包下的所有类，并将它们注册为类型别名，使得在 XML 映射文件中可以直接使用这些类名。  mybatis.mapperLocations=classpath:mappers/\*.xml：  这个配置用于指定 MyBatis 在类路径下查找 XML 映射文件的位置。XML 映射文件包含了 SQL 映射信息，定义了 SQL 语句与 Java 方法的映射关系。  （3）分析持久化实体类  打开src/main/java下的com.ch.ch6\_6.entity包中的MyUser.java实体类，分析其中的代码，同ch6\_5。  Q:实体类为何没有@Entity和@Table注解？Q:为何这里的接口要写实现？  在 JPA (Java Persistence API) 中，使用 @Entity 和 @Table 注解是为了将一个普通的 Java 类映射到数据库表。这两个注解分别用于声明一个类是实体类，并指定实体在数据库中对应的表的信息。  在上面提供的 MyUser 类中，缺少 @Entity 和 @Table 注解，可能是因为该类不是使用 JPA 进行持久化操作的实体类，或者它是一个 MyBatis 等其他持久化框架的实体类。在不同的持久化框架中，实体类的注解可能会有所不同。  关于接口写实现的问题，这是因为在这个项目中可能使用了 MyBatis 框架，而 MyBatis 通常需要为接口提供对应的实现类。MyBatis 的工作方式是通过 XML 映射文件将 SQL 语句与 Java 接口方法关联起来，而为了执行这些 SQL 语句，需要有对应的实现类来实际执行数据库操作。  （4）数据访问层  打开src/main/java下的com.ch.ch6\_6.repository包中的MyUserRepository，分析其代码。  Q:它与ch6\_5中的数据访问层有何不同？  其中只给出了哪个方法？其对应@Mapper的作用?  上述代码使用了 MyBatis 框架的注解方式来定义数据访问接口。相对于之前使用 Spring Data JPA 或者 Spring JDBC 的方式，MyBatis 是另一种持久化框架，它采用 XML 映射文件或注解方式定义 SQL 映射，不需要实体类继承特定的接口。  具体差异如下：  注解方式定义SQL： 上述代码使用了 @Select 注解，直接在接口中编写 SQL 语句，而不需要像 Spring Data JPA 那样定义方法名按照规范进行查询。  @Mapper 注解： 使用 @Mapper 注解标记这个接口，表示这是一个 MyBatis 的 Mapper 映射接口。在 MyBatis 中，@Mapper 注解通常用于标识该接口是一个 Mapper 接口，Spring Boot 在扫描时会自动注册这个接口。  （5）资源中的mapper文件（基于XML方式的SQL映射文件）  打开src/main/resources下的mappers\MyUserMapper.xml，分析其代码。  Q:namespace属性值是？  namespace 属性值是 MyBatis 的 Mapper 接口的全限定名，这里是 com.ch.ch6\_6.repository.MyUserRepository。  Q:select标签中的id和resultType属性代表什么？ 标签中的命令Sql命令是什么？  id 属性是 SQL 语句的唯一标识符，在 MyBatis 的 Mapper 接口中，这个值通常对应接口中的方法名，这里是 findAll。  resultType 属性指定了查询结果的类型，这里是 MyUser。  标签中的命令SQL 命令是：select \* from user  Q:这种用xml参数文件的方式与ch6\_5中将SQL写在java文件中的方式相比，优点是？  XML 配置文件的方式与将 SQL 写在 Java 文件中的方式相比，具有以下一些优点：  解耦性： 将 SQL 语句独立于代码之外，实现了数据访问层和 SQL 语句的解耦，使得代码更加清晰和可维护。  可读性： XML 文件提供了一种结构化的方式来组织 SQL 语句，使得 SQL 语句更易于阅读和理解。  可维护性： 修改 SQL 语句不需要修改 Java 代码，降低了维护的成本。 SQL 语句的修改只需要修改 XML 文件，无需重新编译 Java 代码。  （6）业务层  打开src/main/java下的com.ch.ch6\_6.service包中的MyUserService.java和MyUserServiceImpl.java，分析其代码。  Q: findAll方法与ch6\_5中的findAll方法的区别？  在两个项目中，findAll 方法的区别主要体现在数据访问层的实现方式不同：  ch6\_5项目中的findAll方法是使用 Spring Data JPA 提供的 findAll 方法，它是由 Spring Data JPA 自动实现的，无需手动编写 SQL 语句，简化了数据访问层的编码。  ch6\_6项目中的findAll方法是使用 MyBatis 提供的 @Select 注解，通过 SQL 语句直接查询数据库。在 MyBatis 中，需要手动编写 SQL 语句，灵活性更高，但相应地需要更多的编码工作。  （7）控制器类  打开src/main/java下的com.ch.ch6\_6.controller包中的MyUserController.java，分析其中的代码。  注意@RequestMapping对应的方法调用。  Q:其中调用的是业务层还是数据访问层对象的方法？  调用的是业务层对象的方法。在Spring框架中，控制器（Controller）通常与业务逻辑（Service）层进行交互，而不直接与数据访问层（Repository、Mapper等）交互。这是因为业务逻辑层负责处理业务规则和流程，而数据访问层则负责具体的数据操作。通过这种分层的方式，实现了业务逻辑的解耦，提高了代码的可维护性和灵活性。  （8）启动类  分析启动类src/main/java下com.ch.ch6\_6包中Ch66Application.java的代码，  Q:与之前项目相比，多了什么注解？  与之前的项目相比多的注解是@MapperScan注解，如下图所示：    （9）运行  运行启动类Ch66Application.java(run as Java Application)，  在浏览器地址栏输入：http://localhost:8080/ch6\_6/findAll，显示user表中的所有用户信息。  Q:如果将MyUserMapper.xml改名为RemMyUserMapper.xml,程序还能正常运行吗？为什么？  将文件名字修改后程序不能正常运行。  因为在MyBatis中，Mapper.xml 文件的名称默认需要与对应的 Mapper接口 名称一致，除了文件后缀 .xml 之外。因此，如果将 MyUserMapper.xml 改名为 RemMyUserMapper.xml，程序在运行时可能无法找到对应的 XML 文件，导致 MyBatis 无法正确解析映射关系，从而出现错误。  正确的做法是保持 MyUserMapper.xml 与 MyUserRepository 接口的名称一致（除了文件后缀 .xml），以确保 MyBatis 能够正确加载和解析映射关系。 | | | |
| 思考题解答：  1.“一对一”、“一对多”和“多对多”是概念层的概念，它们在SQL实现层有差别吗（在SQL层面上有一对一、一对多、多对多的描述指令吗）？  在 SQL 实现层面，一对一、一对多和多对多的概念通常不是直接用于描述指令，而是通过表的关系和约束来体现。这些关系通常通过外键和联结来实现。尽管 SQL 并没有直接的一对一、一对多、多对多的特定指令，但通过外键和联结等机制，可以在数据库层面实现这些关系。这些关系的具体表现方式取决于数据库设计和约束。 | | | |