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：使用Mybatis访问数据库,Rest访问，安全控制（一） | | | |
| 实验学时：2学时 | | 实验日期： 2023-11-28 | |
| 实验目的：  1.能在Spring Boot中，使用Mybatis持久层框架访问数据库  2.了解REST服务的特色，能使用Spring Boot支持REST 服务。会使用客户端访问Restful服务  3.了解Spring Boot 安全控制框架 | | | |
| 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 Boot中使用Mybatis持久层框架访问数据库  在上周课本项目ch6\_6的基础上，  修改ch6\_6为使用SQL注解方式：  1）打开src/main/resources下的application.properties位置文件，修改其中的连接数据库地址、参数。  删除以下两条：  mybatis.type-aliases-package=com.ch.ch6\_6.entity  mybatis.mapperLocations=classpath:mappers/\*.xml  Q:这样做的目的是什么？  这两条配置语句是关于 MyBatis 框架的配置。它们的目的是告诉 MyBatis 框架实体类的别名位置和映射文件（XML 文件）的位置。  mybatis.type-aliases-package=com.ch.ch6\_6.entity：  这个配置指定了 MyBatis 在扫描时要扫描的包路径，以查找实体类的别名。  MyBatis 使用别名来代替完整的类名，以简化 SQL 映射文件中的配置。这样在映射文件中可以直接使用简短的别名，而不必使用完整的类路径。  mybatis.mapperLocations=classpath:mappers/\*.xml：  删除这两条配置的话，MyBatis 将不知道在哪里寻找实体类的别名和 SQL 映射文件，可能导致框架无法正确地解析和映射数据库操作。这两项配置提供了框架正确运行所需的元信息。  2）删除资源中的Mapper文件  打开src/main/resources，删除mappers包。  Q:Why？  删除 mappers 包中的 Mapper 文件会导致 MyBatis 框架无法找到相应的映射文件，从而影响对数据库的操作。在典型的 MyBatis 项目中，mappers 包通常包含了与数据库交互的 XML 映射文件。具体影响如下：  无法执行 SQL 语句： MyBatis 的 Mapper 文件包含了 SQL 语句以及与数据库表的映射关系，如果删除了这些文件，框架将无法识别和执行相应的 SQL 语句。  无法进行对象映射： MyBatis 使用 Mapper 文件定义了数据库表和实体类之间的映射关系。如果删除了这些文件，MyBatis 将不知道如何将数据库中的数据映射到实体类，或者如何将实体类的数据保存到数据库。  可能导致运行时异常： 在运行时，MyBatis 将尝试加载这些映射文件。如果文件不存在，将抛出运行时异常，导致应用程序无法正常执行数据库操作。  3）修改数据访问层的接口  打开src/main/java下的com.ch.ch6\_6.repository包中的MyUserRepository，修改如下：  (a)将import org.apache.ibatis.annotations.Mapper; 改为  import org.apache.ibatis.annotations.Select;  (b)删除注解@Mapper  (c)在findAll方法上一行加注解@Select("select \* from user")  Q:如此，SQL语句是否与java代码混在了一起？  在上述代码中，SQL语句与Java代码并没有完全混在一起。虽然在这个MyBatis的Mapper映射接口中使用了注解 @Select("select \* from user")，但这是一种特殊的注解，用于标识该方法执行的 SQL 语句。这种注解的目的是简化 MyBatis 的配置，将 SQL 语句直接嵌入到接口方法上，而不是像传统的 XML 映射文件一样将 SQL 语句单独维护。  4）重新运行  关闭之前运行的项目。重新运行src/main/java下com.ch.ch6\_6包中Ch66Application.java(run as Java Application)，  在浏览器地址栏输入：http://localhost:8080/ch6\_6/findAll  Q: 是否和上周实验显示一样的所有用户信息？    Q:这种方式与前面使用XML mapper文件的方式对比，有什么不同？哪种形式与业务代码的解耦性好？  上述代码使用了注解方式直接将 SQL 语句嵌入到接口方法上，这是一种 MyBatis 提供的基于注解的映射方式。与之相对比的是，MyBatis 还支持使用 XML 文件进行映射，即使用 XML mapper 文件的方式。以下是两种方式的主要对比：  1.可读性和维护性：  注解方式：SQL 语句直接嵌入到 Java 代码中，可读性较差，尤其是对于复杂的 SQL 语句。  XML mapper 文件方式：SQL 语句独立于 Java 代码，存放在 XML 文件中，提高了可读性和维护性，尤其适用于大型项目。  2.解耦性：  注解方式：SQL 语句与 Java 代码紧密结合，解耦性较差，一旦 SQL 发生变化，可能需要修改 Java 代码。  XML mapper 文件方式：SQL 与 Java 代码分离，提高了解耦性。对于同一个 SQL 语句的变化，只需要修改 XML 文件，不影响 Java 代码。  2.练习Spring Boot支持RESTful服务。  了解以下REST服务的风格，  REST：http://localhost:8080/api/v1.0/user  REST：http://localhost:8080/api/findArticleByAuthor\_id/2  URL ：http://localhost:8080/api/findArticleByAuthor\_id?id=2  Q:对比之前的服务请求有什么不同？  在所提供的 REST 服务 URL 中，可以观察到以下不同之处：  1.URL结构：  之前的服务请求示例：http://localhost:8080/findAll  新的服务请求示例1：http://localhost:8080/api/v1.0/user  新的服务请求示例2：http://localhost:8080/api/findArticleByAuthor\_id/2  新的服务请求示例3：http://localhost:8080/api/findArticleByAuthor\_id?id=2  2.RESTful路径风格：  之前的服务请求采用了类似RPC调用的风格，如 findAll 表示获取所有数据。  新的服务请求示例1采用了 RESTful 风格，使用了/api/v1.0/user形式的路径，强调了资源的层级结构。  新的服务请求示例2也采用了 RESTful 风格，通过/api/findArticleByAuthor\_id/2路径表达了获取某作者文章的语义。  新的服务请求示例3采用了类似 RESTful 风格，但通过 URL 参数的形式传递参数。  3.版本控制：  新的服务请求示例1中的/api/v1.0/user可能表明了对于 API 版本的管理，采用了版本号。  之前的服务请求未提及版本号。  4.路径参数 vs. 查询参数：  之前的服务请求示例未使用路径参数或查询参数，而是通过固定路径表示不同的服务。  新的服务请求示例2使用了路径参数，而新的服务请求示例3使用了查询参数。  5.语义化路径：  新的服务请求示例2中的/api/findArticleByAuthor\_id/2更加语义化，直接表达了获取某作者文章的目的。  新的服务请求示例3中的/api/findArticleByAuthor\_id?id=2也在一定程度上表达了相同的语义，但通过查询参数传递。  用STS4打开课本项目ch6\_7。  （1）分析pom.xml文件  Q:其中支持REST的依赖是哪一条？    （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:其中的上下文路径与之前的项目有什么不同？  通过观察多个项目的application.properties文件，发现其余项目的上下文路径都是项目名称一致，只有项目ch6\_7的上下文路径是/api  （3）分析持久化实体类  打开src/main/java下的com.ch.ch6\_7.entity包中的Student.java持久化实体类，分析其中的代码，问：  Q:其对应的数据名？有哪几个属性？  Student 具有四个属性：id、sno、sname、ssex。这些属性分别映射到数据库表 "student\_table" 中的相应列。  （4）分析数据访问层  打开src/main/java下的com.ch.ch6\_7.repository包中的StudentRepository.java，分析其中代码。  Q：@RestResource注解中，REST风格的资源路径是什么？  findBySnameStartsWith方法的形参注解@Param("sname")的含义是什么？  上述两条联合起来，最终请求的URL形式是什么？  在 @RestResource 注解中，path 属性指定了REST风格的资源路径。在这个例子中，path = "snameStartsWith" 意味着通过该自定义查询方法暴露的REST资源的路径为 "snameStartsWith"。所以，当使用这个自定义查询方法进行REST请求时，路径应该是类似于 "/snameStartsWith" 的形式。  @Param("sname") 注解的含义是将方法参数 sname 映射到查询语句中的具体参数。在这个例子中，findBySnameStartsWith 方法有一个参数 sname，通过 @Param("sname") 注解，Spring Data JPA 将该参数映射到查询语句中的 :sname。  因此，最终请求的URL形式是根据 @RestResource 注解中的 path 属性和方法名来构建的。在这个例子中，假设你的应用程序部署在 localhost:8080，那么通过该自定义查询方法进行REST请求的URL形式应该是：  http://localhost:8080/snameStartsWith  这个URL将触发 findBySnameStartsWith 方法的执行，其中的 sname 参数将通过请求进行传递，用于执行相应的查询。  Q：该项目为什么即没有service层，也没有Imply？还没有控制器？  在这个项目中，采用了一种简化的架构或者特殊的设计理念。以下是一些可能的解释：  Spring Data JPA的简化用法： 项目可能充分利用了Spring Data JPA的简化用法，其中Repository接口本身已经包含了一些常见的CRUD（创建、读取、更新、删除）操作。在这种情况下，不需要为每个实体创建Service层或Impl实现类，直接在Repository中定义查询方法，Spring Data JPA会在运行时自动生成相应的实现。  （5）运行项目  运行src/main/java下com.ch.ch6\_7包中Ch67Application.java(run as Java Application)，  观察控制台窗口中显示的的信息，看项目是否正常运行。  用外部可视化工具(如：Navicat for MySQL）打开MySQL中springdatajpa数据库,向其中student\_table表中添加几条记录。  如下图所示，可以看到项目正常运行了。    （6）通过https://github.com/Wisdom-Projects/rest-client或课程网站下载REST API测试客户端rest-client-master.zip，  解压后双击tools文件夹下的restclient-1.2.jar包，运行。  （a）REST Client中，使用GET方式访问“http://localhost:8080/api/students”请求路径获得所有学生信息。  查看返回的Response信息  Q:之前的请求有什么不同？  这次的请求是通过 REST API 测试客户端（rest-client）进行的，而不是通过浏览器。通过 REST 客户端，你可以更直观地查看和管理 API 请求和响应，提供了更多的测试和调试功能。  用浏览器访问“http://localhost:8080/api/students”  Q:可以返回同样的信息吗？  可以  （b）使用GET方式访问“http://localhost:8080/api/students/1”请求路径获得id为1的学生信息，  Q:访问风格与之前有什么不同。  这次的请求是获取特定 id（id 为 1）的学生信息，相对于之前获取所有学生信息的请求，这 是一种更具体和定向的访问风格。  （c）使用GET访问“http://localhost:8080/api/students/search/snameStartsWith? sname=陈”，Q:它调用了什么方法？  这次的请求很可能调用了一个名为 search/snameStartsWith 的自定义方法，该方法通过学生 名字（sname）以“陈”开头进行搜索。  （d）使用GET方式访问“http://localhost:8080/api/students/?page=0&size=2”，  Q:请求路径的含义是什么？  这个请求涉及到分页。它在请求中指定了页数为 0（通常是从 0 开始），每页包含 2 个学生 信息。这是为了有效处理大量数据集的一种常见做法。  （e）使用GET方式访问“http://localhost:8080/api/students/?sort=sno,desc”，  Q:请求路径的含义是什么？  这个请求涉及到对结果进行排序。它要求按照学生编号（sno）降序（desc）排列返回的学 生信息。  （f）用REST Client，发起POST方式请求实现新增一条记录，将要保存的数据放置在请求体中，数据类型为JSON。如第六章PPT P55  （g）用REST Client，使用PUT方式访问“http://localhost:8080/api/students/5”，修改提交的数据。如第六章PPT P56  （h）使用DELETE方式访问“http://localhost:8080/api/students/5”，Q:此请求命令的目的？  删除特定 ID的学生信息。DELETE 请求将指示服务器删除特定资源，即学生信 息，从数据库或 API 中永久移除该条目  （i）用REST Client，使用GET方式访问“http://localhost:8080/api/students”Q:此请求命令的目的？  获取所有学生信息列表。通过 GET 请求，服务器将返回学生信息的列表或集合，允许客户 端查看所有学生的数据。  3.练习使用Spring Boot 安全控制  打开课本项目ch7\_1。  （1）分析pom.xml文件，  Q:其中与安全控制有关的依赖是什么？    这个依赖引入了Spring Boot的Security Starter，它提供了Spring Security的基本配置和依赖，用于实现应用程序的安全性。    这个依赖是Spring Security的测试支持库，用于编写和执行与Spring Security相关的测试。它提供了一些工具和类，使测试Spring Security配置变得更加容易。  这两个依赖一起构成了Spring Boot应用程序中的基本安全控制。spring-boot-starter-security引入了Spring Security的核心功能，而spring-security-test则提供了测试所需的支持。  （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:其中与安全控制有关的配置是什么？  在配置文件中，与安全控制有关的配置是以下一行：    这行配置设置了Spring Security的日志级别为trace，表示启用最详细的日志记录级别，以便更详细地了解Spring Security在应用程序中的行为。这对于调试和排查安全控制相关的问题非常有用。  该配置项指定了Spring Security的包路径 org.springframework.security 下的日志级别为 trace，以便输出详细的安全控制日志。这样，你可以在应用程序运行时查看有关Spring Security内部操作的详细信息，包括安全过滤器链的执行、认证过程等。  （3）分析src/main/resources下templates中的页面文件  Q:页面中使用了哪些变量、超链和方法函数？  **对于register.html页面：**  涉及到以下变量、超链接和方法函数：  1.变量：  ${userDomain}: 该变量对应表单的数据模型，可能是一个包含用户信息的Java对象。  \*{username}: 通过Thymeleaf的表达式语法表示从userDomain对象中获取用户名字段。  \*{password}: 通过Thymeleaf的表达式语法表示从userDomain对象中获取密码字段。  \*{repassword}: 通过Thymeleaf的表达式语法表示从userDomain对象中获取确认密码字段。  2.超链接：  @{/css/bootstrap.min.css}: 表示引用项目中的CSS文件，用于样式设置。  @{/js/jquery.min.js}: 表示引用项目中的JavaScript文件，用于页面交互。  @{/register}: 表示注册表单提交的目标URL。  3.方法函数：  checkBpwd(): 这是一个JavaScript函数，用于在提交表单之前验证输入的用户名和密码。在点击注册按钮时，会触发这个函数，进行一系列的输入验证。  **对于login.html文件：**  涉及到以下变量、超链接和方法函数：  1.变量：  ${param.error}: 表示登录时，如果有错误信息传递过来，则显示用户名或密码错误的提示。  ${param.logout}: 表示用户注销后，显示用户已注销成功的提示。  2.超链接：  @{css/bootstrap.min.css}: 引用项目中的CSS文件，用于样式设置。  @{js/jquery.min.js}: 引用项目中的JavaScript文件，用于页面交互。  @{/login}: 表示登录表单提交的目标URL。  3.方法函数：  $(function() {...}): 这是一个jQuery的文档就绪函数，表示在页面加载完成后执行的JavaScript代码。  $("#loginBtn").click(function() {...}): 当登录按钮被点击时，执行的JavaScript函数，用于验证用户名和密码是否为空，如果不为空则提交表单。  （4）分析持久化实体类  打开src/main/java下的com.ch.ch7\_1.entity包中的Myuser.java和权限Authority.java持久化实体类,分析其中的代码。  注意：@ManyToMany注解、@JoinTable、@Transient，及其中的属性。  对应的表：user，authority，user\_authority。  ★用工具(如Navicat)先将这三张表中原先的记录都删除，然后在权限authority表中添加如下三条记录：  id name  1 ROLE\_ADMIN  2 ROLE\_DBA  3 ROLE\_USER  Q:为什么要先在数据库权限authority表中输入角色权限记录？  在持久化实体类中，MyUser 与 Authority 之间建立了多对多的关联关系，需要通过中间表 user\_authority 存储它们的关联关系。为了测试这一关联关系，需要在 authority 表中添加初始的角色权限记录，以便在测试时可以正确地关联用户和权限。这样可以确保在数据库中有足够的数据来进行多对多关联的测试。  （5）数据访问层  打开src/main/java下的com.ch.ch7\_1.repository包中的MyUserRepository.java，分析其代码。  Q:它与ch6\_5中的数据访问层有何不同？  对于本项目的数据访问层，这是一个使用 Spring Data JPA 的接口，继承了 JpaRepository。Spring Data JPA 会根据方法名自动生成查询语句，使得开发者无需手动实现 CRUD 操作。在该接口中，通过继承 JpaRepository，已经包含了一系列的通用数据操作方法，例如 save、deleteById、findAll 等。  而对于项目ch6\_5的数据访问层，这是一个传统的 Repository 实现方式，使用了 Spring 的 JdbcTemplate 进行手动的 SQL 操作。在该实现类中，开发者需要手动编写 SQL 语句，并使用 JdbcTemplate 执行操作。这样的方式相对较为繁琐，需要开发者关注 SQL 语句的编写和数据映射。  不同之处  （6）业务层  打开src/main/java下的com.ch.ch7\_1.service包中的 UserService.java和UserServiceImpl.java，分析其代码。  注意其中几个与安全控制相关的方法，如：  BCryptPasswordEncoder().encode()  SecurityContextHolder.getContext().getAuthentication()  SecurityContextHolder.getContext().getAuthentication().getName()  SecurityContextLogoutHandler().logout  getAuthorities()  Q:其中给admin和非admin两类用户的角色权限是什么？  在 UserServiceImpl 类的 register 方法中，根据用户名判断用户是管理员（admin）还是普通用户。对于不同的用户类别，分别分配了不同的角色权限。  **对于管理员用户（admin）：**  if ("admin".equals(username)) {  Authority a1 = new Authority();  Authority a2 = new Authority();  a1.setId(1);  a1.setName("ROLE\_ADMIN");  a2.setId(2);  a2.setName("ROLE\_DBA");  authorityList.add(a1);  authorityList.add(a2);  }  对于管理员用户，分配了两个角色权限：  ROLE\_ADMIN  ROLE\_DBA  **对于非管理员用户：**  else {  Authority a1 = new Authority();  a1.setId(3);  a1.setName("ROLE\_USER");  authorityList.add(a1);  }  对于非管理员用户，分配了一个角色权限：  ROLE\_USER  这样，通过注册时判断用户名，给不同类别的用户分配了相应的角色权限。这也是一个常见的权限管理策略，不同角色对应不同的权限，以实现细粒度的访问控制。  （7）安全控制包：  打开src/main/java下的com.ch.ch7\_1.security包中3个与安全控制相关的类和方法，分析其代码：  辅助类：MyUserSecurityService.java，实现了UserDetailsService接口。  loadUserByUsername()  getAuthorityList()  SimpleGrantedAuthority()  辅助类：MyAuthenticationSuccessHandler.java,继承了SimpleUrlAuthentication SuccessHandler。  handle()  getTargetURL()  ......  核心类：MySecurityConfigurerAdapter.java，继承了WebSecurityConfigurerAdapter  authenticationProvider()，DaoAuthenticationProvider()，  setUserDetailsService()，setPasswordEncoder()  configure(AuthenticationManagerBuilder auth)，authenticationProvider()  configure(HttpSecurity http).antMatchers(),  .antMatchers("/user/\*\*").hasRole("USER").antMatchers("/admin/\*\*").hasAnyRole("ADMIN", "DBA").  //其他所有请求登录后才能访问  .anyRequest().authenticated()  ......  Q:对不同L请求，什么用户能访问什么页面？  在 MySecurityConfigurerAdapter 中，配置了不同的请求路径对应的用户访问权限，主要通过 .antMatchers() 方法进行配置。  首页、登录、注册页面、登录注册功能、以及静态资源过滤掉，即可任意访问：  .antMatchers("/toLogin", "/toRegister", "/", "/login", "/register", "/css/\*\*", "/fonts/\*\*", "/js/\*\*").permitAll()  对于 /user/\*\* 路径，只有拥有角色为 "ROLE\_USER" 的用户可以访问：  .antMatchers("/user/\*\*").hasRole("USER")  对于 /admin/\*\* 路径，只有拥有角色为 "ROLE\_ADMIN" 或 "ROLE\_DBA" 的用户可以访问：  .antMatchers("/admin/\*\*").hasAnyRole("ADMIN", "DBA")  其他所有请求登录后才能访问：  .anyRequest().authenticated()  这样，根据不同的请求路径配置了相应的访问权限，对应不同的用户角色。具体来说：  /toLogin, /toRegister, /, /login, /register, /css/\*\*, /fonts/\*\*, /js/\*\* 这些路径允许任意用户访问。  /user/\*\* 路径需要用户拥有 "ROLE\_USER" 角色。  /admin/\*\* 路径需要用户拥有 "ROLE\_ADMIN" 或 "ROLE\_DBA" 角色。  （8）控制器类  打开src/main/java下的com.ch.ch7\_1.controller包中的TestSecurityController，分析其中的代码。  注意@RequestMapping对应的方法及参数。  Q:客户端发出的请求在被控制器处理之前，不同的请求会被谁拦截？  在Spring Security中，请求在被控制器处理之前会经过一系列的拦截器（interceptor）和过滤器（filter）处理。主要的拦截器是由Spring Security提供的，用于处理认证（Authentication）和授权（Authorization）。  通过 MySecurityConfigurerAdapter 中的配置，定义了请求路径的访问规则。这些规则由Spring Security的拦截器进行拦截和处理。以下是其中的一些关键配置：    /toLogin, /toRegister, /, /login, /register, /css/\*\*, /fonts/\*\*, /js/\*\* 这些路径可以被任何用户访问，不受拦截。  /user/\*\* 路径需要用户拥有 "ROLE\_USER" 角色，否则将被拦截。  /admin/\*\* 路径需要用户拥有 "ROLE\_ADMIN" 或 "ROLE\_DBA" 角色，否则将被拦截。  其他所有路径需要用户登录后才能访问。  拦截的处理由Spring Security的过滤器链负责。在 MySecurityConfigurerAdapter 中，通过 configure(HttpSecurity http) 方法配置了一系列的过滤器和拦截规则，其中包括认证过滤器、授权过滤器等。这样，在请求进入控制器之前，Spring Security的拦截器和过滤器会根据配置进行相应的认证和授权处理。  （9）运行  关闭之前运行的项目。运行src/main/java下com.ch.ch7\_1包中Ch71Application.java(run as Java Application)，  在浏览器地址栏输入：测试运行。  http://localhost:8080/ch7\_1    http://localhost:8080/ch7\_1/toLogin  Q:如果登录时，输错密码导致认证失败，会转什么页面？ 为什么？提示什么信息？    http://localhost:8080/ch7\_1/toRegister  Q:如果注册时，用户名或密码为空，会出现什么？为什么？      http://localhost:8080/ch7\_1/login  Q:与上面的toLogin请求有什么区别？    http://localhost:8080/ch7\_1/register  Q:与上面的toRegister请求有什么区别？    在“注册”页面，注册两类不同的用户，用户名为admin（管理员账号）和非admin账号。  Q:查看user，authority，user\_authority三张表中的记录，密码是什么形式存储的？        在“登录”页面，试用普通用户登录后，去访问管理员页面http://localhost:8080/ch7\_1/admin/main Q:能否成功？  不能成功，因为没有权限  登录管理员账号admin，访问普通用户user的页面 http://localhost:8080/ch7\_1/user/loginSuccess Q:能成功吗？  不能成功，也是因为没有权限： | | | |
| 思考题解答：  无 | | | |