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：单元测试 | | | |
| 实验学时：2学时 | | 实验日期： 2023-12-12 | |
| 实验目的：  1.会使用Spring Boot中的@WebMvcTest进行单元测试  2.会使用Spring Boot中的@SpringBootTest进行集成测试 | | | |
| 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. 项目代码分析  用STS4打开课本项目ch9\_2。  （1）分析ch9\_2项目的 pom.xml文件，  Q:其中与测试有关的依赖是什么？  Q:根据其中的其它依赖可以判断这是一个基于什么数据访问的应用？    这个依赖是Spring Boot提供的测试起步依赖，用于支持在Spring Boot应用程序中进行单元测试和集成测试。  关于数据访问的判断：  项目中包含了以下依赖项：    这是Spring Boot提供的JPA（Java Persistence API）数据访问的起步依赖。它为应用程序提供了集成JPA的支持，允许你通过简单的注解配置来实现持久性。  此外，还包含了MySQL数据库连接器的依赖：    这表明该应用程序是基于MySQL数据库的。因此，可以判断这是一个使用Spring Data JPA 进行数据访问的应用程序，并且与 MySQL 数据库进行交互。  （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，与步骤0对应。  （3）分析持久化实体类  打开src/main/java下的com.ch.ch9\_2.entity包中的Student.java持久化实体类,分析其中的代码。  注意：其中的属性和@Table注解中对应的外部数据表。  Q：数据库中表名是？  根据注解和属性的对应关系，可以得知表名为 "student\_table"。这是通过@Table(name = "student\_table")注解指定的。  用外部工具（如Navicat）打开springbootjpa数据库，观察该表的内容,可清空已有的记录.  （4）数据访问层  打开src/main/java下的com.ch.ch9\_2.repository包中的StudentRepository.java，分析其代码。  Q:其代码为空的含义？  StudentRepository 继承自 JpaRepository<Student, Integer>，这意味着它是一个 Spring Data JPA 的仓库接口。这个接口的命名规范是按照实体类（这里是 Student）和主键类型（这里是 Integer）来命名的。  它的代码为空是因为它继承了 JpaRepository，而这个接口已经提供了一组基本的数据访问方法，包括保存、删除、查询等。Spring Data JPA 会根据这个接口的定义，自动实现这些方法，无需开发者手动编写。  （5）分析控制器类  打开src/main/java下com.ch.ch9\_2.controller包中的StudentController.java，分析其中的代码。  Q:以下各注解的含义？  @RestController  @RequestMapping("/student")  @PostMapping("/save")  public String save(@RequestBody Student student)  @GetMapping("/getOne/{id}")  public Student getOne(@PathVariable("id") int id)  @RestController: 告诉Spring，这个类是一个控制器（Controller），并且其中的方法返回的数据直接写入 HTTP 响应体中，通常用于构建 RESTful 风格的 Web 服务。  @RequestMapping("/student"): 映射请求路径，指定控制器处理的基本 URI。在这里，所有的请求都会以 /student 作为根路径。  @PostMapping("/save"): 映射处理 HTTP POST 请求的方法，处理 /student/save 路径的请求。该方法用于保存学生信息。  @PostMapping 表示这是一个处理 POST 请求的方法。  /save 表示处理的路径是 /student/save。  public String save(@RequestBody Student student): 处理保存学生信息的请求的方法。  @RequestBody: 用于接收 HTTP 请求的请求体中的数据，并将其绑定到方法的参数上。在这里，它表示将请求体中的 JSON 数据（包含学生信息）转换为 Student 对象。  Student student: 方法参数，用于接收从请求体中转换得到的学生信息。  public String: 返回类型为字符串，表示保存成功后返回的消息。  @GetMapping("/getOne/{id}"): 映射处理 HTTP GET 请求的方法，处理 /student/getOne/{id} 路径的请求。该方法用于根据学生的ID查询学生信息。  @GetMapping 表示这是一个处理 GET 请求的方法。  /getOne/{id} 表示处理的路径是 /student/getOne/{id}，其中 {id} 是一个占位符，用于接收路径中的ID参数。  public Student getOne(@PathVariable("id") int id): 处理查询学生信息的请求的方法。  @PathVariable("id"): 用于将 URI 模板变量（在这里是 {id}）绑定到方法的参数上。  int id: 方法参数，用于接收从路径中提取的学生ID。  public Student: 返回类型为 Student，表示查询成功后返回的学生信息。  （6）分析项目src/main/resources/templates文件夹，  Q:其中有前端页面文件吗？    无任何文件  2. 使用@WebMvcTest的单元测试（在缺失前端模块的场景下,测试controller模块）  （1）单元测试\_@WebMvcTest测试用例  打开src/test/java下的com.ch.ch9\_2包中的WebMvcTestStudentController.java，分析其代码。  Q:以下几个与测试相关的注解的含义：  @RunWith(SpringRunner.class)  @RunWith(SpringRunner.class) 注解表示使用 Spring Runner 来运行测试。SpringRunner 是 Spring TestContext Framework 的一部分，它提供了整合测试的支持。它会创建 Spring 应用程序上下文，并在测试执行过程中对其进行管理。  在这个特定的测试中，@RunWith(SpringRunner.class) 的目的是启用 Spring 集成，确保测试能够使用 Spring 的特性，如注入依赖、使用 @Autowired 等。  测试类 WebMvcTestStudentController 使用了 @WebMvcTest(StudentController.class) 注解，它表明这是一个专门测试 StudentController 的 Spring MVC 测试。@WebMvcTest 注解会限制 Spring 上下文的加载，仅加载与测试类有关的一部分 Bean（在这里是 StudentController）。  @WebMvcTest(StudentController.class)  Q:此处的StudentController.class是什么模块？  @WebMvcTest(StudentController.class) 注解表示这是一个专注于测试 StudentController 的 Spring MVC 测试。该注解用于测试 Spring MVC 控制器的功能，提供了对 MVC 层的支持，包括注入 MockMvc 实例。  具体含义如下：  @RunWith(SpringRunner.class): 使用 Spring Runner 来运行测试。  @WebMvcTest(StudentController.class): 限制 Spring 上下文的加载，只加载与 StudentController 相关的一部分 Bean，用于测试 MVC 层。  @Autowired private MockMvc mvc: 注入一个 MockMvc 实例，用于模拟和测试 Spring MVC 控制器。  @MockBean private StudentRepository studentRepository;: 使用 @MockBean 注解模拟 StudentRepository，因为 StudentController 依赖于 StudentRepository。这样，在测试中可以定义和配置模拟对象的行为。  这个注解的目的是让测试更加专注和快速，只加载与 StudentController 相关的组件，而不是整个 Spring 上下文。这有助于提高测试的执行速度，同时保留了测试的焦点。  在 @WebMvcTest(StudentController.class) 注解中，StudentController.class 是指要测试的 Spring MVC 控制器的类。在这个注解中，你传入的是要测试的具体控制器类，也就是 StudentController。  @Test  @Test 注解是 JUnit 框架中用于标识测试方法的注解。  （2）Q: 测试代码中，@MockBean 模拟的是？  @MockBean 注解用于模拟一个 Spring Bean 对象，它在测试环境中替代了原有的 Bean 对象。在这个测试中，@MockBean 用于模拟 StudentRepository，即在测试过程中，将原始的 StudentRepository 替换为一个模拟对象。  具体来说，@MockBean 注解的作用是将模拟对象注入到 Spring 容器中，以替代应用中相同类型的真实 Bean 对象。在这个测试中，studentRepository 是 StudentController 的一个依赖项，通过使用 @MockBean，我们能够在测试中控制 studentRepository 的行为，例如定义在调用 getOne(1) 时返回一个特定的学生对象。  在测试过程中，通过使用 BDDMockito.given(studentRepository.getOne(1)).willReturn(stu) 设置了对 getOne(1) 方法的模拟行为，使其返回预定义的 stu 学生对象。这样，在执行 StudentController 中的相应测试方法时，就能够控制和验证与 studentRepository 交互的行为  （3）分析saveTest()中的代码：  Q:它模拟是对controller中的哪个请求？  它模拟的是对 StudentController 中的 /student/save 请求。  Q:save请求传递的Student实例数据是？它传递给了谁？  save 请求传递的 Student 实例数据是：    这个 Student 对象包含了学生的信息，其中包括学号 (sno)、姓名 (sname)、性别 (ssex)。  Q:解释 perform()的参数  解释 perform() 的参数：  post("/student/save"): 模拟发送一个 POST 请求，请求的路径是 /student/save。  contentType(MediaType.APPLICATION\_JSON\_UTF8): 设置请求的内容类型为 JSON 格式，字符集为 UTF-8。  accept(MediaType.APPLICATION\_JSON\_UTF8): 设置期望响应的内容类型为 JSON 格式，字符集为 UTF-8。  content(mapper.writeValueAsString(stu)): 将 stu 对象转换成 JSON 字符串，并作为请求的内容发送。  （4）分析getStudent()中的代码：  Q:它模拟是对controller中的哪个请求？  它模拟的是对 StudentController 中的 /student/getOne/{id} 请求。  Q:BDDMockito.given()模拟的输入和输出数据是什么？  BDDMockito.given(studentRepository.getOne(1)).willReturn(stu); 模拟的输入和输出数据是：  输入：调用 studentRepository 的 getOne 方法，参数为 1。  输出：返回 stu，即一个预先定义好的 Student 对象  （5）测试运行WebMvcTestStudentController.java (Run as/JUnit Test),  Q:观察左侧JUnit窗口的统计信息和控制台，解释其中的输出信息：  MockHttpServletRequest:  Handler:  ......  MockHttpServletResponse:  对 /student/save 接口的模拟请求和响应：  模拟的请求信息：  HTTP Method: POST  Request URI: /student/save  Parameters: 无额外参数  Headers: Content-Type: application/json;charset=UTF-8, Accept: application/json;charset=UTF-8  Body: {"id":1,"sno":"5555","sname":"陈恒","ssex":"男"}  处理的 Handler（Controller 方法）：  Type: com.ch.ch9\_2.controller.StudentController  Method:publicjava.lang.String  com.ch.ch9\_2.controller.StudentController.save(com.ch.ch9\_2.entity.Student)  响应信息：  Status: 200  Headers: Content-Type: application/json;charset=UTF-8, Content-Length: 7  Content type: application/json;charset=UTF-8  Body: success  对 /student/getOne/1 接口的模拟请求和响应：  模拟的请求信息：  HTTP Method: GET  Request URI: /student/getOne/1  Parameters: 无额外参数  Headers: Content-Type: application/json;charset=UTF-8, Accept: application/json;charset=UTF-8  Body: 无请求体  处理的 Handler（Controller 方法）：  Type: com.ch.ch9\_2.controller.StudentController  Method: public com.ch.ch9\_2.entity.Student com.ch.ch9\_2.controller.StudentController.getOne(int)  响应信息：  Status: 200  Headers: Content-Type: application/json;charset=UTF-8  Content type: application/json;charset=UTF-8  Body:  {  "id": 1,  "sno": "5555",  "sname": "陈恒",  "ssex": "男"  }  （6）此测试运行中，  Q:Ch92Application.java 和 StudentRepository.java模块运行了吗？读写外部数据库了吗？  在这个测试过程中，Ch92Application.java和StudentRepository.java模块均没有运行，并且这个过程也没有读写外部数据库。  Q:用外部工具(如Navicat）打开springbootjpa数据库，查察其中student\_table的内容有变化吗？  数据库的student\_table表的内容没有发生改变。  3. 基于@SpringBootTest的集成测试（在缺失前端模块的场景下,后端模块集成测试）  （1）打开src/test/java下的com.ch.ch9\_2包中的SpringBootTestStudent Controller.java，分析其代码。  注意其中几个与测试相关的注解：  @RunWith(SpringRunner.class)  @SpringBootTest(classes = Ch92Application.class) Q:该注解与上面的测试用例有什么不同？  1.Scope（范围）：  @SpringBootTest：通常用于集成测试，会加载完整的 Spring 应用程序上下文，包括所有的 bean。它主要用于测试整个应用程序的集成，而不仅仅是单个组件。  @WebMvcTest：专门用于测试 Spring MVC 控制器（Controller），会限制应用程序上下文的范围，仅加载与 Web 层相关的组件。这样可以更专注地测试 MVC 层的行为。  2.加载的组件：  @SpringBootTest：加载整个应用程序上下文，包括所有的 bean，适用于集成测试，测试范围更广。  @WebMvcTest：仅加载与 Web 层相关的组件，如控制器、拦截器等。不加载整个应用程序上下文，更专注于测试 Web 层。  3.用途：  @SpringBootTest：适用于集成测试，测试应用程序的多个组件之间的交互，更适用于端到端的测试场景。  @WebMvcTest：适用于测试单个控制器（Controller）的行为，专注于 MVC 层的测试，可以提供更快的测试反馈。  （2）Q:@Before及其注释的方法initMockMvc() 的作用？  在每个测试方法执行之前初始化 MockMvc 对象。主要任务是创建 MockMvc 对象并配置它以模拟对 Spring MVC 控制器的请求。  （3）Q:分析saveTest()和getStudent()中的代码，与上面的测试有什么不同之处？  SpringBootTestStudentController 的 saveTest() 和 getStudent():  初始化方式：  saveTest() 和 getStudent() 方法使用 @Before 注解的 initMockMvc() 方法初始化 MockMvc 对象。这是因为它们使用 @SpringBootTest 注解，需要初始化整个 Spring 上下文。  @Transactional 注解表明 saveTest() 方法将在事务内运行，并在测试结束时回滚事务。这是因为 saveTest() 涉及到对数据库的更改，而希望在测试结束时撤销这些更改。  WebMvcTestStudentController 的 saveTest() 和 getStudent():  初始化方式：  saveTest() 和 getStudent() 方法不需要显式初始化 MockMvc 对象。在 @WebMvcTest 注解下，Spring 会自动为您创建和配置 MockMvc。  （4）测试运行SpringBootTestStudentController.java(Run as/JUnit Test),观察左侧JUnit窗口的统计信息,  Q: 请解释控制台输出的信息：  MockHttpServletRequest:  Handler:  ......  MockHttpServletResponse:  saveTest()方法的输出信息：  MockHttpServletRequest：描述模拟的HTTP请求。  HTTP Method = POST：请求方法为POST。  Request URI = /student/save：请求的URI路径。  Parameters = {}：请求的参数为空。  Headers = [Content-Type:"application/json;charset=UTF-8", Accept:"application/json;charset=UTF-8"]：请求头信息，包括接受和发送JSON数据的内容类型。  Body = {"id":1,"sno":"5555","sname":"陈恒","ssex":"男"}：请求体中包含的JSON数据。  Session Attrs = {}：会话属性为空。  Handler：描述处理请求的控制器和方法。  Type = com.ch.ch9\_2.controller.StudentController：控制器类型。  Method=publicjava.lang.String  com.ch.ch9\_2.controller.StudentController.save(com.ch.ch9\_2.entity.Student)：处理请求的方法。  Async：异步处理相关信息。  Async started = false：未启动异步处理。  Async result = null：异步处理结果为空。  Resolved Exception：已解决的异常类型为空。  ModelAndView：关于模型和视图的信息。  View name = null：视图名称为空。  View = null：视图为空。  Model = null：模型为空。  FlashMap：FlashMap属性为空。  MockHttpServletResponse：模拟的HTTP响应。  Status = 200：HTTP响应状态码为200（OK）。  Content type = application/json;charset=UTF-8：响应内容类型为JSON。  Body = success：响应体为 "success"。  其他响应信息，如头信息和Cookie等为空。  getStudent()方法的输出信息：  MockHttpServletRequest：描述模拟的HTTP请求。  HTTP Method = GET：请求方法为GET。  Request URI = /student/getOne/1：请求的URI路径。  Parameters = {}：请求的参数为空。  Headers = [Content-Type:"application/json;charset=UTF-8", Accept:"application/json;charset=UTF-8"]：请求头信息，包括接受和发送JSON数据的内容类型。  Body = null：请求体为空。  Session Attrs = {}：会话属性为空。  Handler：描述处理请求的控制器和方法。  Type = com.ch.ch9\_2.controller.StudentController：控制器类型。  Method = public com.ch.ch9\_2.entity.Student com.ch.ch9\_2.controller.StudentController.getOne(int)：处理请求的方法。  Async：异步处理相关信息。  Async started = false：未启动异步处理。  Async result = null：异步处理结果为空。  Resolved Exception：已解决的异常类型为空。  ModelAndView：关于模型和视图的信息。  View name = null：视图名称为空。  View = null：视图为空。  Model：模型信息包含了响应的Student对象的JSON表示。  FlashMap：FlashMap属性为空。  MockHttpServletResponse：模拟的HTTP响应。  Status = 200：HTTP响应状态码为200（OK）。  Content type = application/json;charset=UTF-8：响应内容类型为JSON。  其他响应信息，如头信息和Cookie等为空。  后续的日志信息包括事务的回滚信息，以及测试环境的初始化和关闭过程。  （5）此测试运行中，  Q:Ch92Application.java 和 StudentRepository.java模块运行了吗？读写外部数据库了吗？  在这个测试中，上述两个模块运行了，并且也读写了外部数据库。增加了一条记录。原因是 springBootTest 会将数据访问层注入容器运行，所以会访问真实的数据库。  （6）用外部工具(如Navicat)打开springbootjpa数据库，查察其中student\_table的内容有变化吗？为什么？  内容发生了变化因为读写了外部数据库。 | | | |
| 思考题解答：  无 | | | |