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核心基础实验 | | | |
| 实验学时：2学时 | | 实验日期： 2023-10-21 | |
| 实验目的：  1.能够分析Spring Boot项目的基本配置的源码、启动类、核心注解、starters  2.能够使用多种方式读取SpringBoot应用配置  3.能够理解自动配置原理  4.能够使用、配置日志  5.能够使用条件注解,使用@conditional元标记自定义条件 | | | |
| 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. 使用Spring Tools 4开发工具构建Spring Boot项目**  **参照第3章PPT 3.3.2 ， 使用Spring Tool Suite4，构建一个Spring Starter Project项目，Type选Maven，等项目构建完，如项目ch3\_3,测试运行。**    **2. 启动类和核心注解**  **用STS创建一个简单的Spring Starter Project,例如test4\_1的web项目。（如有红色提示，右击项目,Maven/Update Project...）**  **（1）启动类分析**  **打开src/main/java中的含有main()方法的启动类XXXApplication.java,**  **Q:其中的核心注解，及import的包是什么？**  其中的核心注解是@SpringBootApplication注解  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;:  **（2）核心组合注解@SpringBootApplication分析**  **打开项目Maven Dependencies中的依赖包，如spring-boot-autoconfigure-X.X.X.RELEASE.jar，展开其中的org.springframework.boot.autoconfigure,**  **在其中找到SpringBootApplication.class,双击打开其源码，找到 public @interface SpringBootApplication**  **源代码中，组合的注解有哪些？**  主要有@SpringBootConfiguration、@EnableAutoConfiguration、@ComponentScan等注解。  **（3）@SpringBootConfiguration注解分析**  **打开Maven Dependencies中的依赖包，如spring-boot-X.X.X.RELEASE.jar，打开org.springframework.boot,**  **在其中找到SpringBootConfigration.class,双击打开其源码，找到 public @interface SpringBootConfiguration**  **Q：其组合的注解有哪些？**  @Documented、@Configuration等注解  **（4）@EnableAutoConfiguration注解**  **打开项目Maven Dependencies中的依赖包，如spring-boot-autoconfigure-X.X.X.RELEASE.jar，展开找到org.springframework.boot.autoconfigure,展开，**  **在其中找到EnableAutoConfiguration.class,双击打开分析其源码。**  **打开项目的pom.xml文件，分析其中的<dependencies> </dependencies>，**  **点击代码窗口下方的Dependency Hierarchy标签，展开的Dependency Hierarchy窗口，**  **Q:spring-boot-starter-web 相关自动配置的内容有哪些？**  @EnableAutoConfiguration 注解  ENABLED\_OVERRIDE\_PROPERTY 常量  @Target, @Retention, @Documented, @Inherited 注解这些是元注解，用于标注 @EnableAutoConfiguration 注解的适用范围、保留策略、文档性质以及继承性质。  @AutoConfigurationPackage 和 @Import 注解这两个注解用于指示 Spring Boot 如何加载自动配置类，以及通过哪个选择器来进行加载。  **（5）@ComponentScan 注解**  **打开Maven Dependencies中的依赖包，如spring-context-X.X.X.RELEASE.jar，展开找到org.springframework.context.annotation，**  **展开在其中找到ComponentScan.class，分析其源码。**  这段源码定义了注解 @ComponentScan，它用于配置组件扫描的指令，通常与 @Configuration 类一起使用，提供了与 Spring XML 的 <context:component-scan> 元素相似的支持。@ComponentScan 注解：这是一个自定义注解，用于指示 Spring 容器如何进行组件扫描。它可以配置一些属性，以定义要扫描的包或类型，以及其他配置选项。  **3.1 定制Banner**  **打开网站 http://patorjk.com/software/taag，输入你想用的标题，复制、粘贴到一个banner.txt文件中。**  **将banner.txt文件复制到项目src/main/resources目录中，重启项目（Run as Application），查看控制台中显示。**  **3.2 关闭Banner**  **打开项目中的application.properties配置文件，在其中添加如下配置：**  **spring.main.banner-mode = off**  **重启项目，查看控制台中显示。**  通过观察控制台的输出，可以看到之前的标题：Hello，Spring Boot!已经不再显示，说明我们已经通过在配置文件中输入语句关闭了Banner。  **4. 全局配置文件application.properties**  **用STS 4打开课本项目ch4\_1。**  **分析其中的全局配置文件application.properties，测试运行。**  test.msg：这是一个自定义属性，可能用于配置应用程序中的消息文本。在示例中，它的值为 "read config"。  obj.sname 和 obj.sage：这两个属性可能用于配置应用程序中的对象的名称和年龄。在示例中，分别设置了对象的名称为 "chenheng" 和年龄为 88。  logging.level.root：这个属性用于配置日志系统的默认级别，示例中将其设置为 "info"，表示根日志级别为 INFO。  logging.level.org：这个属性配置了 org 包下的日志级别，示例中将其设置为 "warn"，表示该包下的日志级别为 WARN。  logging.level.com.ch.ch4\_1：这个属性配置了 com.ch.ch4\_1 包下的日志级别，示例中将其设置为 "debug"，表示该包下的日志级别为 DEBUG。  **4.1 能够使用多种方式读取SpringBoot应用配置(Environment类；@Value；@ConfigurationProperties；@PropertySource)**  **打开src/main/java下的com.ch.ch4\_1.controller包，**  **（1）Environment类.读取应用程序运行时的环境变量**  **分析EnvReaderConfigController.java代码，在浏览器地址栏用 http://localhost:8080/testEnv 测试。**  **Q:哪个方法取得了env中的属性？**  @RestController 注解：这个注解标识了这个类是一个 Spring MVC 控制器，它会处理 HTTP 请求并返回响应。@RestController 注解是 @Controller 和 @ResponseBody 注解的组合，表示这个类的方法将返回 JSON 或其他数据类型。  @Autowired 注解：这个注解用于将 Environment 类型的 Bean 自动注入到 env 字段中。Environment 是 Spring Framework 提供的用于访问应用程序环境属性的接口。  @RequestMapping("/testEnv") 注解：这个注解用于将 HTTP 请求映射到控制器的方法。具体来说，它将 /testEnv 路径映射到 testEnv() 方法。  **在浏览器地址栏用 http://localhost:8080/testEnv 测试结果如下：**    **（2）@Value注解读取配置文件内容**  **分析ValueReaderConfigController.java代码，在浏览器地址栏用http://localhost:8080/testValue测试。**  **Q:哪个变量被注入？**    变量 msg 被注入了。  **（3）@ConfigurationProperties建立配置文件中属性与对象的映射关系**  **分析com.ch.ch4\_1.model包中的StudentProperties.java代码，注意以下注解的使用：**  **@Component//使用Component注解，声明一个组件，被控制器依赖注入**  **@ConfigurationProperties(prefix = "obj")//obj为配置文件中key的前缀**  **分析controller包中的ConfigurationPropertiesController.java代码，在浏览器地址栏用**  **http://localhost:8080/testConfigurationProperties 测试。**  **Q:配置文件中哪些属性怎样注入了对象？**  **在浏览器地址栏用http://localhost:8080/testConfigurationProperties 测试结果如下：**    在上述代码中，通过使用@ConfigurationProperties注解和StudentProperties类，以下属性从配置文件中注入到StudentProperties对象中：  obj.sname：这个属性的值被注入到StudentProperties对象的sname字段，表示学生的名字。  obj.sage：这个属性的值被注入到StudentProperties对象的sage字段，表示学生的年龄。  **（4）@PropertySource注解配置项目的其他配置文件**  **分析controller包中的PropertySourceValueReaderOhterController.java代码，在浏览器地址栏用**  **http://localhost:8080/testProperty 测试。**  **Q:其他的配置文件有哪些？**  **在浏览器地址栏用http://localhost:8080/ testProperty 测试结果如下：**    在上述代码中，通过@PropertySource注解加载了两个其他配置文件，它们分别是：  test.properties：这是第一个其他配置文件，包含了属性my.msg。  ok.properties：这是第二个其他配置文件，包含了属性your.msg。  这两个配置文件都被@PropertySource注解指定，允许从这些文件中读取属性值，并在控制器中使用这些属性。  **4.2 日志配置：**  **（1）分析controller包中的LogTestController.java代码，在浏览器地址栏用**  **http://localhost:8080/testLog 测试。**  **测试结果如下：**    **（2）打开src/main/resources包下的application.properties文件，分析其中的日志配置;**  **打开c:/log/my.log， 查看log文件中的信息。**    **4.3 Spring Boot的自动配置原理**  **分析@EnableAutoConfiguration组合注解所调用方法系列，了解@EnableAutoConfiguration注解自动配置的原理。**  @EnableAutoConfiguration注解可以让Spring Boot根据当前应用项目所依赖的jar自动配置项目的相关配置。例如，在Spring Boot项目的pom.xml文件中添加了spring-boot-starter-web依赖，Spring Boot项目会自动添加Tomcat和Spring MVC的依赖，同时对Tomcat和Spring MVC进行自动配置。  **Q:其最终读取配置文件是哪个文件夹下的什么文件名？**  最终读取的配置文件位于src/main/resources目录的application.yml  **5. Spring Boot的条件注解，项目ch4\_1。**  **自定义条件，分析com.ch.ch4\_1.conditional包下代码，**  **Q: 自定义条件的实现类是？实现了什么接口？重写的其中什么方法？**  自定义条件的实现类是MyCondition类和YourCondition类。  这两个类都实现了org.springframework.context.annotation.Condition接口，并重写了接口中的matches方法。  **Q: MessagePrint接口及其实现类的作用？**  MessagePrint 接口：这是一个接口，定义了一个名为 showMessage() 的方法，用于返回一条消息。  该接口的主要作用是为不同的条件化装配提供一个统一的接口，让条件化创建的Bean都能实现相同的方法，从而统一处理不同条件下的消息显示。  MyMessagePrint 类：这是 MessagePrint 接口的一个实现类。showMessage() 方法返回字符串 "test.properties文件存在。"，表示当特定条件满足时，应该显示的消息。  YourMessagePrint 类：这同样是 MessagePrint 接口的一个实现类。showMessage() 方法返回字符串 "test.properties文件不存在！"，表示当特定条件满足时，应该显示的消息。  **Q：配置类ConditionConfig中@Bean和@Conditional注解的作用？**  @Bean 注解：  @Bean 注解用于在 Spring 容器中定义 Bean，它告诉 Spring 容器如何创建和配置 Bean 实例。  在 ConditionConfig 配置类中，有两个方法都使用了 @Bean 注解，分别是 myMessage() 和 yourMessage() 方法，它们用于定义两个不同的 Bean。  @Conditional 注解：  @Conditional 注解用于条件化地控制是否应该创建一个特定的 Bean。  在 ConditionConfig 配置类中，@Conditional 注解被用于两个 @Bean 方法，分别引用了 MyCondition 和 YourCondition 类。这意味着根据条件的评估结果，只有一个方法中的 Bean 会被创建。  例如，如果 MyCondition 的条件为 true，则创建 myMessage Bean；如果 YourCondition 的条件为 true，则创建 yourMessage Bean。根据条件的不同，选择性地创建不同的 Bean。  **Q: 测试类：TestMain 如何加载配置类？解释运行结果。**    TestMain测试类的main()函数通过创建一个 AnnotationConfigApplicationContext 对象来加载配置类 ConditionConfig.class，然后调用 context.getBean(MessagePrint.class) 来获取一个 MessagePrint 类型的 Bean。接下来，它调用 mp.showMessage() 来打印 Bean 返回的消息。  通过 context = new AnnotationConfigApplicationContext(ConditionConfig.class)，我们创建了一个 Spring 应用上下文，并加载了配置类 ConditionConfig.class。  在 ConditionConfig 配置类中，有两个 @Bean 方法，分别是 myMessage() 和 yourMessage()，它们使用了 @Conditional 注解，一个引用了 MyCondition 类，另一个引用了 YourCondition 类。这意味着根据条件的不同，将创建不同的 Bean。  通过 context.getBean(MessagePrint.class)，我们获取了一个 MessagePrint 类型的 Bean。根据 ConditionConfig 中的条件化配置，实际创建了一个 MyMessagePrint 或 YourMessagePrint 的 Bean 实例。  接着，调用 mp.showMessage()，它会执行 MyMessagePrint 或 YourMessagePrint 的 showMessage() 方法，返回不同的消息。  运行结果取决于 MyCondition 和 YourCondition 中的条件判断逻辑以及配置文件的存在与否。如果 MyCondition 的条件为 true（test.properties 存在），则会创建 MyMessagePrint，并打印 "test.properties文件存在。"。如果 YourCondition 的条件为 true（test.properties 不存在），则会创建 YourMessagePrint，并打印 "test.properties文件不存在！"。因此，运行结果将根据条件的不同而不同。 | | | |
| 思考题解答：  **1.SpringBoot的自动配置是怎样运行的？**  Spring Boot的自动配置是基于一系列的条件，这些条件可以根据类路径上的类、配置属性、Bean的存在等信息来确定是否应用自动配置。这些条件可以在应用程序的配置文件中进行自定义，以满足特定的需求。  Spring Boot还提供了许多可扩展的自动配置选项，您可以根据自己的需求进行配置。您可以使用@EnableAutoConfiguration注解来启用Spring Boot的自动配置，或者使用@SpringBootApplication注解，它包含了@EnableAutoConfiguration注解。 | | | |