山东大学 计算机科学与技术 学院

现代软件开发 课程实验报告

|  |  |  |
| --- | --- | --- |
| 学号： | 姓名： | 邮箱： |
| 实验题目：Spring框架基础 | | |
| 实验过程中遇到和解决的问题：  （记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）  3.1  （1）.配置文件applicationContext.xml是基于Spring文档中的IoC容器部分创建的。该文档讨论了如何配置和使用Spring的IoC容器。被引入代码中的地方通常是在应用程序的主入口或配置文件中。它用于指定Spring容器如何进行依赖注入和管理应用程序的各种组件。  打开applicationContext.xml得到：    由图得参考了    配置文件在    （2）.在项目中将使用者Test类与被使用者TestDaoImpl解耦的方式是通过使用第三方容器进行依赖注入。常见的第三方容器是Spring Framework的IoC容器。在该项目中，应用程序的配置文件（例如applicationContext.xml）中会定义bean的配置，将Test类与TestDaoImpl注入到容器中。通过在Test类中声明对TestDaoImpl接口的依赖，并由容器负责实例化和注入具体的实现对象，实现了解耦。这体现了依赖注入（DI）的概念。  （3）.创建新的实现类TestDaoImpl2，实现TestDao接口，并重写sayHello()方法以输出新的内容。然后，在配置文件applicationContext.xml中找到与TestDaoImpl相关的bean定义，将该定义中的类路径从TestDaoImpl修改为TestDaoImpl2。最后编译和运行程序，main方法将使用新的TestDaoImpl2实现类，并输出替换后的内容。  （4）.与ch1\_1相同，使用容器组装依赖关系即可  3.2  （1）.多了这个包    没有使用applicationContext.xml而是将它的功能放在了类中    （2）.逻辑清晰: DAO层负责与数据库的交互，Service层负责业务逻辑的处理，Controller层负责接收请求和返回响应。分层可以让代码更加清晰和易于理解。  每层分为接口和接口的实现是为了强调面向接口编程和依赖倒置原则。通过定义接口，可以将实际的实现与抽象接口分离，降低模块间的耦合度，并支持代码的可扩展性和可维护性。接口的实现类可以根据具体需求进行替换或扩展，而不需要修改其他层次的代码。  （3）.      （4）.不会，因为如果将TestDaoImpl和TestServiceImpl改名，将不会对项目的其余部分产生影响。这是因为依赖注入（DI）是通过接口进行的，而不是直接引用具体的实现类。只要新的类实现了对应的接口，且在配置文件中做了相应的修改，DI仍然可以将新的实现注入到其他组件中。  （5）.在例题ch1\_2的MainApp.java中，tc是通过依赖注入（DI）方式获得的。这是通过在MainApp.java类中使用@Autowired注解，将TestController自动注入到MainApp中的属性tc上实现的。Spring容器负责创建TestController实例，并将其注入到MainApp中。  （6）.按照以下步骤进行修改：首先在TestController类上使用@Component注解，并指定一个自定义的名称；  然后，在MainApp类中，使用@Qualifier注解指定要注入的TestController bean的名称：  java  public class MainApp {  @Autowired  @Qualifier("testTC")  private TestController tc;  //...  }  这样，通过在MainApp中使用@Qualifier注解指定了名称为"testTC"的TestController bean，可以实现通过该名字调用的效果。  3.3  （1）.    （2）.集中在配置文件中的@Bean注解与ch1\_2中分散在各处的@Bean注解在声明和使用上有一些不同之处。在配置文件中，我们可以将多个Bean的配置集中在一个类中，通过在方法上添加@Bean注解来声明这些Bean。这样做可以提高代码的可读性和维护性，因为所有的配置都集中在一个地方，方便查看和修改。  （3）.如果修改了dao包或service包中的类名，可能会影响其他部分。这是因为在配置文件中，我们使用了类名作为方法名来声明Bean。  如果修改了类名，则需要相应地修改配置文件中的方法名，以保持一致。如果忘记更新配置文件，可能会导致应用程序在运行时产生错误或找不到所需的Bean。  （4）.  跟之前一样，需要在配置文件中使用@Bean注解来配置该实例。我们需要在配置文件中声明一个方法，使用@Bean注解并返回TestController类的一个实例。当应用程序启动时，Spring容器会扫描配置文件，创建TestController的实例，并将其注入到需要它的地方。  3.4  （1）.包括AspectJ Weaver和AspectJ Runtime库。    （2）.              要定义切入点方法，在切面类中创建一个方法，并用@Pointcut注解进行标记。然后，在其他通知方法中，可以使用切入点方法的名称来引用它。  要在通知方法上添加注解，需将其简单地用所需的AspectJ注解进行标注，如@Before、@After、@Around等。  （3）.在配置文件中，可能会使用到@EnableAspectJAutoProxy等注解，该注解启用了对AspectJ切面和代理的支持。    （4）ch1\_4中被切入的目标类是TestService。但是，目标类的源代码没有被修改。增强是通过AOP实现的，在运行时将切面代码织入应用程序中。切面拦截定义的连接点（例如方法执行），并根据需要应用通知方法。  （5）在ch1\_4的main()方法中，通过依赖注入（DI），TestDao.class被实例化为一个Bean。要使其工作，需要将TestDao类配置为一个组件（例如使用@Component或相关的构造型注解）。然后，当Spring容器启动时，会扫描组件并自动创建Bean实例。  （6）例如，如果要拦截TargetClass中的specificMethod()方法，可以将切入表达式更新为execution(\*com.example.TargetClass.specificMethod(..))。这样，切面只会在该特定方法周围应用通知。  3.5  （1）在ch1\_5项目的instance包中，有三个类分别是：  BeanInstanceFactory：通过静态工厂方法创建Bean的实例。  BeanInstanceFactory2：通过实例工厂方法创建Bean的实例。  BeanInstanceFactory3：通过实例工厂方法创建Bean的实例，且将Bean的实例保存在实例变量中。  这些类提供不同的方式来实例化Bean，但其实现的目标都是创建Bean的实例。  （2）  @Bean注解用于声明一个方法，该方法将返回一个特定的Bean实例。方法名将成为Bean的名称，可以使用value属性来指定Bean的名称，例如@Bean(value="beanClass")。  value属性可以省略不写，因为如果不指定属性的名称，默认情况下将使用方法名作为Bean的名称。  （3）在main()方法中多次使用appCon.getBean("beanInstanceFactory")创建Bean实例，其类的id相同，是因为默认情况下，Spring容器会使用单例模式来管理Bean。在单例模式下，容器会保留一个Bean的实例，并在后续的请求中返回相同的实例。  （4）如果在配置类中的@Bean注解的下一行，添加@Scope("prototype")，即 @Bean(name="beanInstanceFactory", scope = "prototype")，则运行结果会不同。使用@Scope("prototype")将Bean的作用域设置为原型模式，每次请求Bean时，容器都会创建一个新的实例。  3.6  因为Bean的作用域不同；在ch1\_6示例中，使用了@Scope注解来定义Bean的作用域。其中，singleton表示单例模式，prototype表示原型模式等。  当多次创建原型模式（prototype）的Bean实例时，其类的id会不同。这是因为每次请求原型模式的Bean时，容器都会创建一个新的实例，而不是返回之前创建的实例。  3.7  结果如下 | | |
| 结果分析与体会：   1. 在例子中，各部分的角色如下： 2. 配置类（AppConfig）：负责定义和配置Bean，使用注解（如@Bean）来声明Bean的创建方式和属性。（2）Bean类（BeanInstanceFactory、BeanInstanceFactory2、BeanInstanceFactory3）：具体的Bean实现类，用于创建Bean的实例。（3）Spring容器：负责管理Bean的生命周期、依赖注入等任务。在例子中，通过创建容器对象（AnnotationConfigApplicationContext）来加载配置类，并使用getBean()方法获取所需的Bean实例。   执行流程如下：   1. 创建容器对象（AnnotationConfigApplicationContext）。 2. 加载配置类（AppConfig）。 3. 根据配置类中的定义，创建Bean的实例。 4. 在需要使用Bean的地方，通过容器获取Bean的实例。 5. 执行相应的业务逻辑，包括依赖注入、方法调用等。 6. 容器负责管理Bean的生命周期，包括初始化和销毁（如果有定义相应的方法）。 7. 依赖注入（DI）是指在对象创建的过程中，注入其所依赖的其他对象。Spring实现依赖注入的方法主要有构造函数注入、Setter方法注入和字段注入。关键注解有：@Autowired：自动装配依赖关系。可以用于构造函数、Setter方法、字段和方法上。@Qualifier：指定具体的bean名称，用于细化@Autowired的注入方式。@Value：注入简单值或表达式。   IOC/DI的简化UML图示例：      在上述示例中，Bean实例持有一个依赖项（Dependency）的引用，并通过依赖注入的方式将具体的依赖项注入到Bean实例中。容器负责创建和管理Bean实例以及解析依赖关系。   1. 面向切面编程（AOP）技术通过将横切关注点（例如日志、事务管理等）与核心业务逻辑解耦，从而实现非侵入式增强。它的原理是通过代理和动态字节码生成，在程序运行时动态地织入增强逻辑。关键概念和原理包括：   （1）切面（Aspect）：横切关注点的模块化单位，包含切点和增强逻辑  （2）连接点（Join Point）：在程序执行过程中可以插入横切逻辑的特定点，如方法调用、异常抛出等。  （3）切点（Pointcut）：用于匹配连接点的表达式，确定在哪些连接点上应用增强逻辑。  （4）增强（Advice）：在特定的切点上执行的逻辑，例如前置通知、后置通知、环绕通知等。  （5）织入（Weaving）：将切面和目标对象编织在一起的过程，可以在编译时、加载时或运行时完成。  面向对象编程（OOP）是一种编程范式，关注于对象的行为和数据封装。它将系统划分为一组相互协作的对象，并通过定义类、封装、继承等方式实现软件的组织和复用。  AOP与OOP的区别在于：  AOP关注于在程序运行过程中的横切关注点，通过动态代理方式实现增强，从而可以解决横切关注点的复用和解耦问题。  OOP关注于对象的封装、继承和多态等特性，以实现良好的对象设计和抽象。  AOP与OOP可以相互结合使用，AOP可以作为OOP的一个补充，提供横切关注点的解决方案，使得系统更具扩展性和可维护性。  4.访问Spring官方网站并查找相关文档和API文档。[https://spring.io/。可以找到Spring框架的各种文档、教程、示例代码和API文档。](https://spring.io/%E3%80%82%E5%9C%A8%E8%AF%A5%E7%BD%91%E7%AB%99%E4%B8%8A%EF%BC%8C%E4%BD%A0%E5%8F%AF%E4%BB%A5%E6%89%BE%E5%88%B0Spring%E6%A1%86%E6%9E%B6%E7%9A%84%E5%90%84%E7%A7%8D%E6%96%87%E6%A1%A3%E3%80%81%E6%95%99%E7%A8%8B%E3%80%81%E7%A4%BA%E4%BE%8B%E4%BB%A3%E7%A0%81%E5%92%8CAPI%E6%96%87%E6%A1%A3%E3%80%82%E4%BD%A0%E5%8F%AF%E4%BB%A5%E4%BD%BF%E7%94%A8%E7%BD%91%E7%AB%99%E5%AF%BC%E8%88%AA%E6%88%96%E6%90%9C%E7%B4%A2%E5%8A%9F%E8%83%BD%E6%9D%A5%E6%89%BE%E5%88%B0%E4%BD%A0%E9%9C%80%E8%A6%81%E7%9A%84API%E6%96%87%E6%A1%A3%E3%80%82)  5.在上述例子中，使用getBean()方法来获取Bean实例时，参数可以是Bean的名称或Bean的类型（.class名称）。区别如下：  使用Bean的名称作为参数，例如getBean("test")，是根据配置类中@Bean注解的方法名或value属性来指定的，可以自定义Bean的名称。使用Bean的名称时，可以根据名称获取对应的Bean实例。  使用Bean的类型作为参数，例如getBean(BeanClass.class)，是根据配置类中方法的返回类型进行匹配，Spring容器将根据类型查找对应的Bean实例。使用Bean的类型时，可以方便地获取特定类型的Bean实例，比较方便。在大多数情况下，根据类型获取Bean实例是最常见的用法，而根据名称获取Bean实例通常用于需要自定义名称的特殊情况。 | | |