原 java初、中、高级面试题必备——javaEE

2019年06月23日 13:50:59 在IT中穿梭旅行 阅读数 58 编辑

Spring

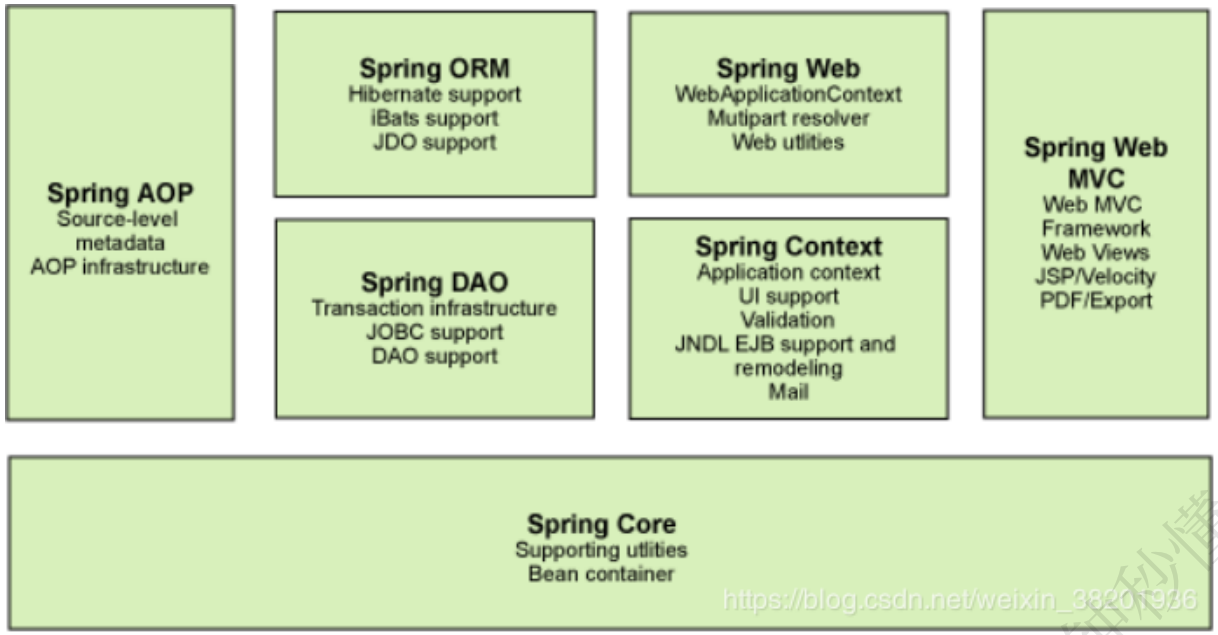
93. 什么是spring?

Spring 是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架。目的：解决企业应用程序开发的复杂度。框架优势之一，分层架构，简单来说就是减少代码量，使javaBean之间的关系更加明确。

94. 使用Spring框架的好处是什么?

- **轻量**：Spring 是轻量的，基本的版本大约2MB。
- **控制反转**：Spring通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们。
- **面向切面的编程(AOP)**：Spring支持面向切面的编程，并且把应用业务逻辑和系统服务分开。
- **容器**：Spring 包含并管理应用中对象的生命周期和配置。
- **MVC框架**：Spring的WEB框架是个精心设计的框架，是Web框架的一个很好的替代品。
- **事务管理**：Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）。
- **异常处理**：Spring 提供方便的API把具体技术相关的异常（比如由JDBC，Hibernate or JDO抛出的）转化为一致的unchecked 异常。

95.Spring由哪些模块组成?



组成 Spring 框架的每个模块（或组件）都可以单独存在，或者与其他一个或多个模块联合实现。每个模块的功能如下：

核心容器：核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转 （IOC） 模式**将应用程序的配置和依赖性规范与实际的应用程序代码分开。**

Spring Context：Spring Context是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。

Spring AOP：通过配置管理特性，Spring AOP 模块直接将面向切面的编程功能集成到了 Spring 框架中。所以，可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块给Spring 的应用程序中的对象提供了事务管理服务

Spring DAO：通过使用JDBC抽象和DAO模块，保证数据库代码的简洁，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。

Spring ORM：Spring 通过提供ORM模块，支持我们直接在JDBC之上使用一个对象/关系映射(ORM)工具，Spring 支持集成主流的ORM框架，如Hiberate,JDO 和 iBATIS SQL Maps。Spring的事务管理同样支持以上所有ORM框架及JDBC。

Spring Web 模块：Spring的WEB模块是构建在application context 模块基础之上，提供一个适合web应用的上下文。这个模块也包括支持多种面向web的任务，如透明地处理多个文件上传请求和程序级请求参数的绑定到你的业务对象。支持Jakarta Struts的集成。

Spring MVC 框架：MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

96.解释一下IOC?

IOC:控制反转。字面意思：控制权由主动变成被动，简单来说就是把复杂的系统分解成相互合作的对象，通过封装对象，以达成内部实现对外部透明。从而降低对象之间的耦合性。

图 IOC解耦过程

如图：引进中间位置的“第三方”，也就是IOC容器，使得A、B、C、D这4个对象没有了耦合关系，齿轮之间的传动全部依靠“第三方”了，全部对象的控制权全部上缴给“第三方”IOC容器，所以，IOC容器成了整个系统的关键核心，它起到了一种类似“粘合剂”的作用，把系统中的所有对象粘合在一起发挥作用，如果没有这个“粘合剂”，对象与对象之间会彼此失去联系，这就是有人把IOC容器比喻成“粘合剂”的由来。

举例说明：**没引入IOC容器前**，对象A依赖于对象B，那么对象A在初始化或者运行到某一点的时候，自己必须主动去创建对象B或者使用已经创建的对象B。无论是创建还是使用对象B，控制权都在自己手上。

引入IOC之后，对象A与对象B之间失去了直接联系，所以，当对象A运行到需要对象B的时候，IOC容器会主动创建一个对象B注入到对象A需要的地方。

项目实际应用IOC： 只需在Spring——> applicationContext.xml 配置相应的Bean以及设置相应的属性，让Spring容器生成类的实例对象以及管理对象。当Spring启动时，配置文件中的Bean会初始化好。

97.Spring 的IOC如何实现?

Spring提供了一个实例化和管理对象的地方，就是applicationContext.xml 配置文件 ， 开发者只需要将java类放入容器中，在容器中获得javaBeanID的唯一属性，就可以拿到javaBean 对象供开发者使用。

```
在applicationContext.xml中配置

<bean id="dept"  class="com.lyz.entity.Dept" >

</bean>
```

配置完成后直接调用就可以 **Spring默认实例化是一个单例模式**

```
@Test
public void testSpringIOC(){
    ApplicationContext  context = new  ClassPathXmlApplicationContext("applicationContext.xml");
    Dept dept1 = (Dept)context.getBean("dept");
    dept.setName("技术部");
    System.out.println(dept.getdName());
}
```

98.ApplicationContext通常的实现是什么?

- **ClassPathXmlApplicationContext**：此容器也从一个XML文件中加载beans的定义，这里，你需要正确设置classpath因为这个容器将在classpath里找bean配置。
- **FileSystemXmlApplicationContext**：此容器从一个XML文件中加载beans的定义，XML Bean 配置文件的全路径名必须提供给它的构造函数。
- **WebXmlApplicationContext**：此容器加载一个XML文件，此文件定义了一个WEB应用的所有bean。

依赖注入

99.请问什么是DI？并且简要说明一下DI是如何实现的?

DI（Dependency Injection）叫依赖注入。容器动态的将某种依赖关系注入到组件之中。

比如类A需要用到接口B中的方法，就需要为类A和接口B建立关联或依赖关系，只需要在类A中定义好用于关联接口B的方法（构造器或setter方法），将类A和接口B的实现类C放入容器中，通过对容器的配置来实现二者的关联。

依赖注入三种方式：

1. setter方法注入（设值注入）、
2. 构造器注入
3. 通过注解方式来注入

Spring支持setter注入和构造器注入，通常使用构造器注入来注入必须的依赖关系，对于可选的依赖关系，则setter注入是更好的选择，setter注入需要类提供无参构造器或者无参的静态工厂方法来创建对象。

100. 哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入?

你两种依赖方式都可以使用，构造器注入和Setter方法注入。最好的解决方案是用构造器参数实现强制依赖，setter方法实现可选依赖。

Spring Beans

101.什么是Spring beans?

Spring beans 是那些形成Spring应用的主干的java对象。它们被Spring IOC容器初始化，装配，和管理。这些beans通过容器中配置的元数据创建。比如，以XML文件中<bean/> 的形式定义。

102.spring 支持几种 bean 的作用域?

当通过spring容器创建一个Bean实例时，不仅可以完成Bean实例的实例化，还可以为Bean指定特定的作用域。Spring支持如下5种作用域：

- **singleton：单例模式**，在整个Spring IoC容器中，使用singleton定义的Bean将只有一个实例
- **prototype：原型模式，**每次通过容器的getBean方法获取prototype定义的Bean时，都将产生一个新的Bean实例
- **request**：对于每次HTTP请求，使用request定义的Bean都将产生一个新实例，即每次HTTP请求将会产生不同的Bean实例。只有在Web应用中使用Spring时，该作用域才有效
- **session**：对于每次HTTP Session，使用session定义的Bean豆浆产生一个新实例。同样只有在Web应用中使用Spring时，该作用域才有效

- **globalsession**：每个全局的HTTP Session，使用session定义的Bean都将产生一个新实例。典型情况下，仅在使用portlet context的时候有效。同样只有在Web应用中使用Spring时，该作用域才有效。

103. Spring框架中的单例bean是线程安全的吗?

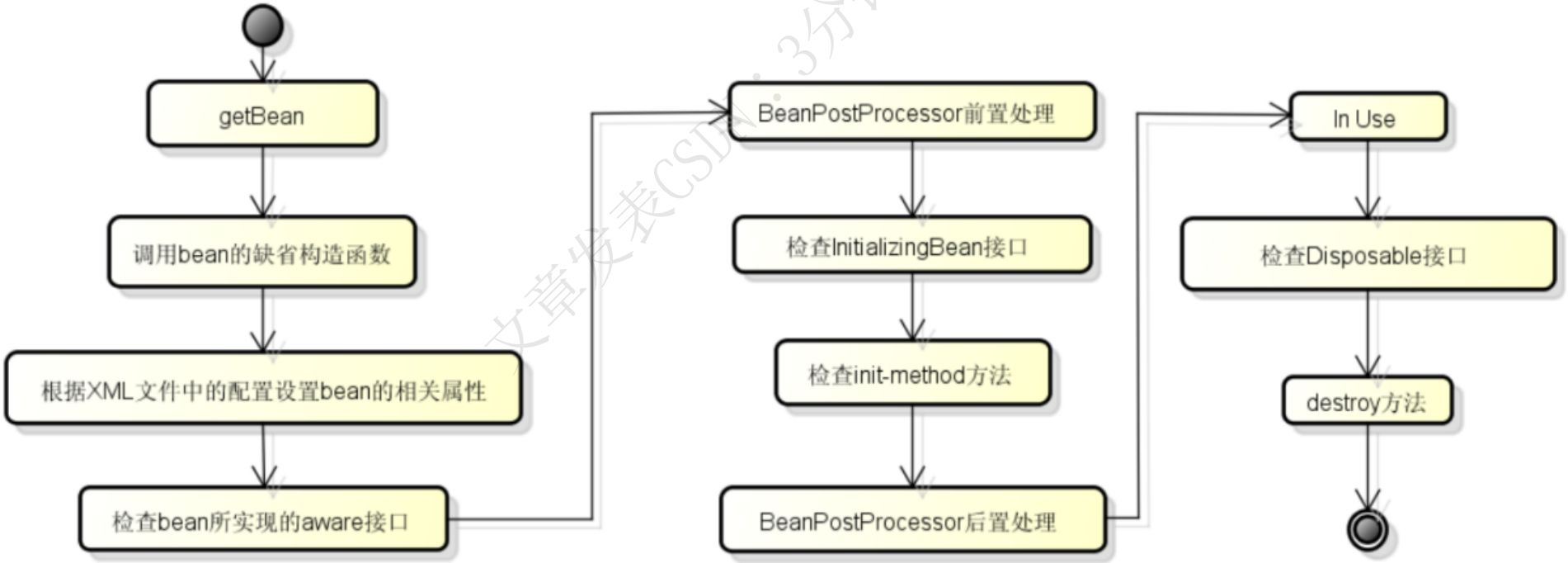
不，Spring框架中的单例bean不是线程安全的。

104. 哪些是重要的bean生命周期方法？ 你能重载它们吗?

有两个重要的bean 生命周期方法，第一个是setup， 它是在容器加载bean的时候被调用。第二个方法是 teardown 它是在容器卸载类的时候被调用。

The bean 标签有两个重要的属性（init-method和destroy-method）。用它们你可以自己定制初始化和注销方法。它们也有相应的注解（@PostConstruct和@PreDestroy）。

解释Spring框架中bean的生命周期



- Spring容器 从XML 文件中读取bean的定义，并实例化bean。
- Spring根据bean的定义填充所有的属性。
- 如果bean实现了BeanNameAware 接口，Spring 传递bean 的ID 到 setBeanName方法。
- 如果Bean 实现了 BeanFactoryAware 接口， Spring传递beanfactory 给setBeanFactory 方法。
- 如果有任何与bean相关联的BeanPostProcessors，Spring会在postProcesserBeforeInitialization()方法内调用它们。
- 如果bean实现IntializingBean了，调用它的afterPropertySet方法，如果bean声明了初始化方法，调用此初始化方法。
- 如果有BeanPostProcessors 和bean 关联，这些bean的postProcessAfterInitialization() 方法将被调用。
- 如果bean实现了 DisposableBean，它将调用destroy()方法。

105. 什么是bean的自动装配?

Spring 容器能够自动装配相互合作的bean，这意味着容器不需要<constructor-arg>和<property>配置，能通过Bean工厂自动处理bean之间的协作。

106. spring 自动装配 bean 有哪些方式?

Spring容器负责创建应用程序中的bean同时通过ID来协调这些对象之间的关系。作为开发人员，我们需要告诉Spring要创建哪些bean并且如何将其装配到一起。

spring中bean装配有两种方式：

- 隐式的bean发现机制和自动装配
- 在java代码或者XML中进行显示配置

当然这些方式也可以配合使用。

107.自动装配有哪些局限性？

自动装配的局限性是：

- **重写**： 你仍需用 <constructor-arg>和 <property> 配置来定义依赖，意味着总要重写自动装配。
- **基本数据类型**：你不能自动装配简单的属性，如基本数据类型，String字符串，和类。
- **模糊特性**：自动装配不如显式装配精确，如果有可能，建议使用显式装配。

Spring注解

108.Spring注解有哪些？.

@Autowired(按类型注入) 通过 @Autowired的使用来消除 set ， get方法。

```
public class HelloWorld{
    // 下面两种@Autowired只要使用一种即可
    @Autowired
    private UserDao userDao; // 用于字段上
}
```

@Qualifier 注解

当有多个相同类型的bean却只有一个需要自动装配时，将@Qualifier 注解和@Autowire 注解结合使用以消除这种混淆，指定需要装配的确切的bean。

@Required 注解

这个注解表明bean的属性必须在配置的时候设置，通过一个bean定义的显式的属性值或通过自动装配，若@Required注解的bean属性未被设置，容器将抛出BeanInitializationException。

@controller 和@RestController的区别

@RestController注解相当于@ResponseBody + @Controller合在一起的作用。

1) 如果只是使用@RestController注解Controller，则Controller中的方法无法返回jsp页面，或者html，配置的视图解析器不起作用，返回的内容就是Return 里的内容。

2) 如果需要返回到指定页面，则需要用 @Controller配合视图解析器InternalResourceViewResolver才行。

如果需要返回JSON，XML或自定义mediaType内容到页面，则需要在对应的方法上加上@ResponseBody注解。

AOP

109. 解释AOP

AOP 面向切面编程，是指面向核心业务编程，非核心业务交给第三方技术封装处理

AOP是一种面向切面的编程，允许程序模块化横向切割关注点，或横切典型的责任划分，如**日志和事务管理。**

例如**日志功能。**日志代码往往**横向**地散布在所有对象层次中，与核心功能的代码几乎毫无关系，**这种散布在各处的无关的代码被称为横切（cross cutting）**，在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP的作用就是：利用"横切"技术，剖解开封装的对象内部，将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。便于减少系统的重复代码，降低模块之间的耦合度。

110. Aspect 切面

AOP核心就是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组API提供横切功能。比如，一个日志模块可以被称作日志的AOP切面。根据需求的不同，一个应用程序可以有若干切面。在Spring AOP中，切面通过带有@Aspect注解的类实现。

111. 在Spring AOP 中，核心关注点和横切关注点的区别是什么?

AOP把软件系统分为两个部分：**核心关注点**和**横切关注点**。

业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。**横切关注点**的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，**比如权限认证、日志、事物。AOP的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。**

112 你如何理解AOP中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念?

切面：就是Spring里面关注的类

连接点： 连接点指的就是类当中的异常或者方法

切点：切点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。如果连接点相当于数据中的记录，那么切点相当于查询条件。(指的就是方法中的集合)

增强（Advice）：增强是织入到目标类连接点上的一段程序代码。Spring提供的增强接口都是带方位名的，如：BeforeAdvice、AfterReturningAdvice、ThrowsAdvice等。

引介（Introduction）：引介是一种特殊的增强，它为类添加一些属性和方法。这样，即使一个业务类原本没有实现某个接口，通过引介功能，可以动态的为该业务类添加接口的实现逻辑，让业务类成为这个接口的实现类。

织入（Weaving）：织入是将增强添加到目标类具体连接点上的过程，AOP有三种织入方式：①编译期织入：需要特殊的Java编译期（例如AspectJ的ajc）；②装载期织入：要求使用特殊的类加载器，在装载类的时候对类进行增强；③运行时织入：在运行时为目标类生成代理实现增强。Spring采用了动态代理的方式实现了运行时织入，而AspectJ采用了编译期织入和装载期织入的方式。

113.请问AOP的实现原理是什么?

考察点：动态代理

参考回答:

springAOP的底层实现是通过代理模式实现的。AOP代理主要分为**静态代理**和**动态代理**，静态代理的代表为AspectJ；所谓的静态代理就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强。

静态代理缺点：一个代理类只能代理一个业务类。如果业务类增加方法时，相应的代理类也要增加方法。

Spring AOP中的**动态代理**主要有两种方式，**JDK动态代理**和**CGLIB动态代理**。JDK动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

如果目标类没有实现接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理

JDK和CGLIB生成动态代理类的区别:

JDK动态代理只能针对实现了接口的类生成代理（实例化一个类）。此时代理对象和目标对象实现了相同的接口，目标对象作为代理对象的一个属性，具体接口实现中，可以在调用目标对象相应方法前后加上其他业务处理逻辑
CGLIB是针对类实现代理，主要是对指定的类生成一个子类（没有实例化一个类），覆盖其中的方法 。

默认的策略是如果目标类实现接口，则使用JDK动态代理技术，如果目标对象没有实现接口，则默认会采用CGLIB代理

114.请问aop的应用场景有哪些?

考察点：spring AOP

参考回答：

Authentication 权限 ， Caching 缓存 ， Synchronization 同步，Transactions 事务，性能检测，日志管理。

默认的策略是如果目标类实现接口，则使用JDK动态代理技术，如果目标对象没有实现接口，则默认会采用CGLIB代理

115. Spring支持的事务管理类型

Spring支持两种类型的事务管理：

- **编程式事务管理**：通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
- **声明式事务管理：通过**注解和XML配置来管理事务，将业务代码和事务管理分离。

116. Spring框架的事务管理有哪些优点?

- 它为不同的事务API 如 JTA，JDBC，Hibernate，JPA 和JDO，提供一个不变的编程模式。
- 它为编程式事务管理提供了一套简单的API而不是一些复杂的事务API如
- 它支持声明式事务管理。
- 它和Spring各种数据访问抽象层很好得集成。

117. 你更倾向用那种事务管理类型?

大多数Spring框架的用户选择声明式事务管理，因为它对应用代码的影响最小，只需要通过注解和XML配置来管理事务就可以

118.spring如何管理事务?

1. 在applicationContext.xml中导入AOP、tx的头标签
2. 配置一个事务管理者来管理数据库的事务。
3. 配置一个事务管理者来管理业务逻辑层（增删改查方法）的事务。

```
<!-- 配置一个事务管理者来管理数据库的事务-->

```

```
<bean id="transactionManager" class ="org.springframework.jdbc.datasource.DataSourceTransactionManager"> <property name="dataSource" ref="dataSource" /> </bean>

<!--配置事务管理者管理 哪一层的事务（管理业务逻辑层的事务），管理哪些类哪些方法的事务-->

< !-- -REQUIRED: PROPAGATION_REQUIRED =0 --  如何业务逻辑层没有事务，系统自动创建事务-->
< !-- -SUPPORTS: 如果没有配置事务，代码就以非事务方式运行-->
< !-- -MANDATORY: 没有事务抛异常>
< !-- -REQUIRED_NEW: 如何业务逻辑层没有事务，系统自动创建事务，有事务，把你的挂起不用-->
< !-- -NOT_SUPPORTED: 总是以非事务方式运行-->

< !-- -isolation: DEFAULT  调用ACID特性，执行数据库底层的事务隔离级别-->
< !-- -rollback-for="java.lang.Exception"  出现java的异常回滚 -->

<tx: advice  id ="advice"  transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="save*" propagation="REQUIRED" isolation="DEFAULT" rollback-for="java.lang.Exception"/>
    </tx:attributes>
</tx:advice>


<!--配置事务处理的切面类 -->

<aop:config>
    <aop:pointcut expression="execution("*com.lyz.service.*.*(..))" id="pc" />
    <aop:advisor advice-ref="advice" pointcut-ref="pc" />
</aop:config>
```

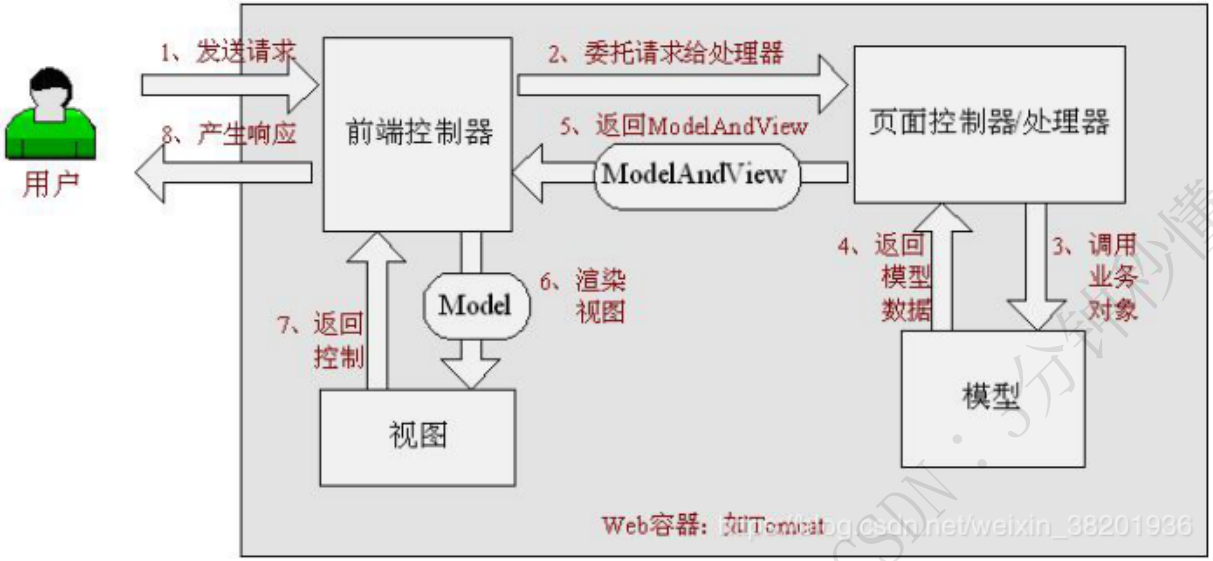

4.xml中配置完后，可以使用注解@Transactional

120.简单介绍一下SpringMVC？

一种基于java的实现了webMVC设计模式的请求驱动类型的轻量级Web框架，将Web层进行职责解耦。

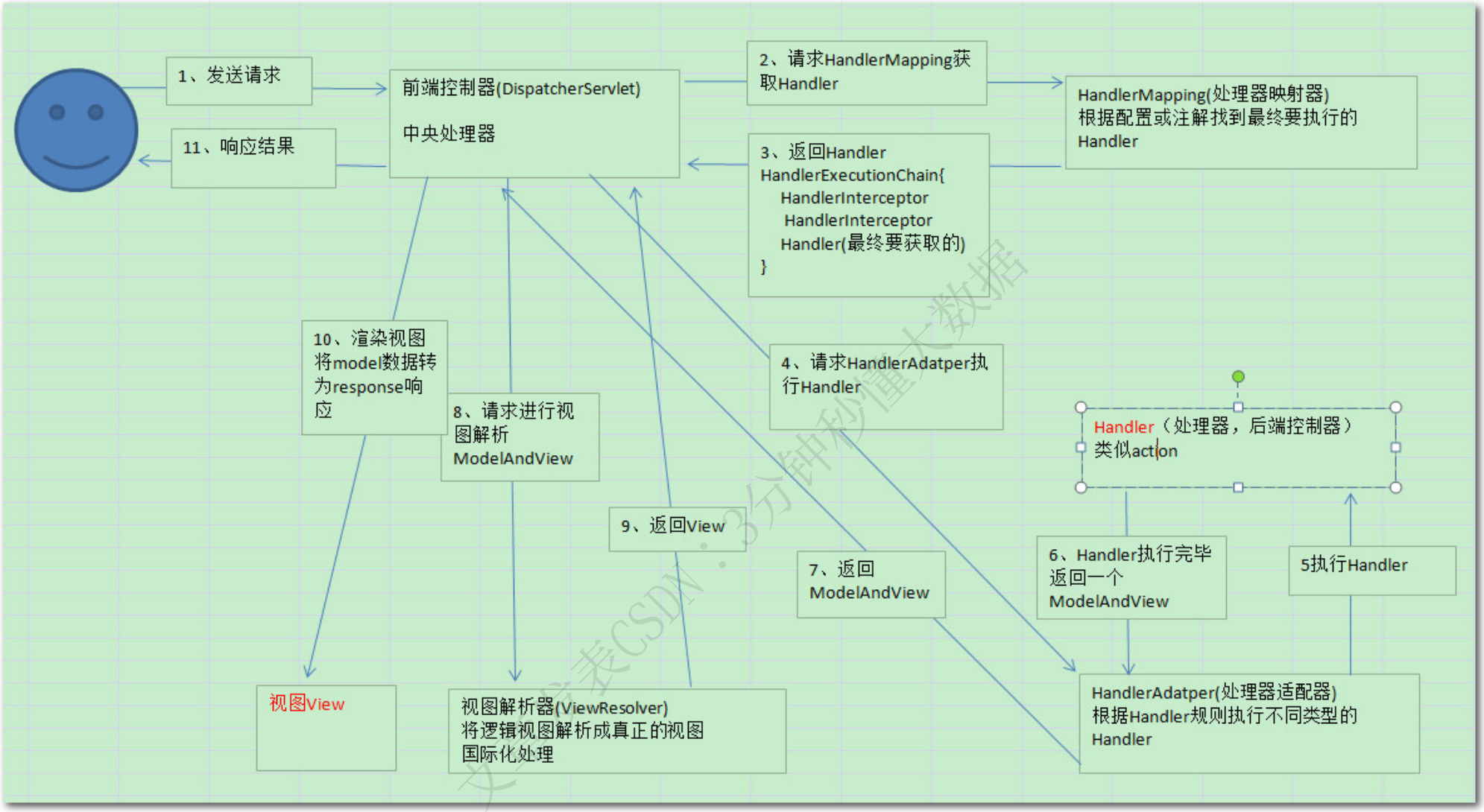
特点：天生与Spring框架集成。

121.SpringMVC 的工作原理是怎么样的?



1. 用户向前端控制器DispatcherServlet 发送请求。
2. 前端控制器根据请求信息如(URL、HTTP协议、请求头、请求参数、cookie)来选择哪一个页面控制器（HandlerMapper）进行处理,并把请求委托给它。
3. 页面控制器收到请求后，进行功能处理，首先收集和绑定请求参数到一个对象，并进行验证。
4. 然后将命令对象委托给业务对象进行处理，处理完之后返回一个ModelAndView(模型数据和逻辑视图名)。
5. 前端控制器收回控制权，然后根据返回的逻辑视图名，选择相应的视图进行渲染，并把模型数据传入以便视图渲染。
6. 前端控制器再次收回控制权，并将响应返回给用户。

122.SpringMVC的运行机制？以及运行机制的流程是什么?



1. 客户端请求提交到DispatcherServlet
2. 由DispatcherServlet控制器查询HandlerMapping，找到并分发到指定的Controller中。
4. Controller调用业务逻辑处理后，返回ModelAndView
5. DispatcherServlet查询一个或多个ViewResoler视图解析器，找到ModelAndView指定的视图
6. 视图负责将结果显示到客户端

123. 请说明一下springmvc和spring-boot区别是什么?

首先 springboot 是一个微服务架构，内嵌tomcat，无需布置JAR，war文件，简化Maven配置，自动配置Spring,只需要配置application.properties配置文件

区别在于，Spring是一个引擎，SpringMVC是基于Spring的一个MVC框架。

SpringBoot 是基于Spring4的条件注册的一套快速开发的整合包。

124.Spring MVC注解的优点是什么?

1、XML配置起来有时候冗长，此时注解可能是更好的选择，如jpa的实体映射；注解在处理一些不变的元数据时有时候比XML方便的多，比如springmvc的数据绑定，如果用xml写的代码会多的多；

2、注解最大的好处就是简化了XML配置；其实大部分注解一旦确定后很少会改变，所以在一些中小项目中使用注解反而提供了开发效率，所以没必要一头走到黑；

125.SpringBoot 启动及注解

主函数启动：

- @SpringBootApplication 申请让程序自动给程序进行必要配置
- @ResponseBody 该注解是将结果返回到HTTP response body上
- @RequestMapping 跳转路径，负责前端页面URL地址到Controller中的具体映射
- @Controller 用于定义控制器类，在Spring项目中由控制器负责将用户的URL请求转发到对应的服务接口（Service层）
- @GetMapping 对应查询，表明是一个查询URL映射
- @Autowired 作用将其他的类，接口引入，方法就可以直接调用
- @Service serviceImpl层的注解，对应的是业务层Bean
- @Transactional 事务的标签
- @Repository 对应数据访问层Bean，让Spring创建一个实例
- @Mapper 让mybatic和Dao层进行映射
- @Param 传入多个参数时，需要使用，用来匹配前后端参数是否一致。

126.简单介绍下Mybatis

mybatis是一个支持普通SQL查询，和高级映射的优秀持久化框架，mybatis消除了几乎所有JDBC代码和参数的手工设置，以及结果集的检索，使用XML或注解用于配置和原始映射，将接口和Java对象映射成数据库中的记录

127.Mybatis如何操作数据库的?

应用程序入口——> sqlSessionFactoryBuilder 作用：通过XML 配置文件创建configuration对象

然后通过 build方法 创建sqlSessionFactory对象。

SqlSession 对象的主要功能是完成一次数据库的访问和结果的映射

```
public class MybatisUtils {
        private static SqlSessionFactory  factory=null;
        static {
                try {
                        Reader reader=Resources.getResourceAsReader("mybatis.xml");
                        factory=new SqlSessionFactoryBuilder().build(reader);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public static SqlSession  getSqlSession() {
                return factory.openSession();
        }
        public static void close(SqlSession sqlSession) {
                if(sqlSession!=null) {
                        sqlSession.close();
                }
        }
}
```

写个添加方法

```
public class DeptService {
                        2
        public  static boolean saveDeptInfo(Dept dept) {
                SqlSession sqlSession=MybatisUtils.getSqlSession();
                DeptDaoMapper deptDaoMapper=sqlSession.getMapper(DeptDaoMapper.class);
                int flag=deptDaoMapper.saveDeptInfo(dept);
                sqlSession.commit();
                MybatisUtils.close(sqlSession);
                if(flag!=0)
                        return true;
                return false;

        }
}
```

128.MyBatis中命名空间（namespace）的作用是什么?

```
<mapper namespace="com.wlsj.fgw.dao.first.FirstShowDaoMapper">
```

namespace的作用是绑定dao层接口的，即面向接口编程，当namespace绑定接口后，就不用写接口实现类，mybatis会通过该绑定自动帮助找到对应方法执行的sql语句

129.MyBatis中的动态SQL是什么意思?

优点：1 简化我们拼接sql的操作。2 采用功能强大的基于OGNL(对象-图形导航语言)来消除其他元素。

- if - choose / when / otherwise - trim - where - set - foreach

130.简单说一下 JPA

JPA是java持久层API，持久化框架。

JPA 三方面技术 （1）ORM映射元数据，（2）API，（3）查询语句

（1）JPA支持XML和JDK5.0注解两种元数据的形式，元数据描述对象和表之间的映射关系，将实体对象持久化到数据库表中。

（2）API操作实体对象。

（3）查询语句——> 是面向对象而非面向数据库的。

131.你了解docker吗?

Docker 是一个开源的应用容器引擎，基于 Go 语言 并遵从Apache2.0协议开源。

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。

容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。

132.什么是Docker镜像?

Docker镜像是Docker容器的源代码，Docker镜像用于创建容器。使用build命令创建镜像

133.Docker容器有几种状态?

Docker容器可以有四种状态：运行、已暂停、重新启动、已退出。

134.Docker使用流程

1）创建Dockerfile后，您可以构建它以创建容器的镜像　　2）推送或拉取镜像。

135.Dockerfile中最常见的指令是什么?

Dockerfile中的一些常用指令如下：

- FROM：指定基础镜像
- LABEL：功能是为镜像指定标签
- RUN：运行指定的命令
- CMD：容器启动时要运行的命令

136.Dockerfile中的命令COPY和ADD命令有什么区别?

COPY与ADD的区别COPY的<src>只能是本地文件，其他用法一致

137.docker常用命令?

docker pull　拉取或者更新指定镜像

docker push　将镜像推送至远程仓库

docker rm　删除容器

docker rmi　删除镜像

docker images　列出所有镜像

docker ps　列出所有容器

Redis

138. 说一下Redis

Redis 是一个分布式的基于Key-Value类型的内存数据库。

优点：

- 性能非常出色，每秒处理超过10万次读写操作。
- 支持多种数据结构，单个value的最大限制为1GB，如String，List，Set,Sort Set, Hashes,
- 用List 来作FIFO双向链表，实现轻量级的高性能消息队列服务。
- 用set可以做高性能tag系统

缺点：

- 受物理限制，不能用作海量数据的高性能读写。
- 适合场景，较小数据量的高性能操作和运算。

适用场景：

会话缓存。全页缓存。消息队列。排行榜(集合set.有序集合，sortedSort）发布/订阅功能。

Redis的主从复制： 保证部分节点无法通信情况下，集群仍可用。集群使用主从复制模型，每个节点都会有N-1个复制品。

Redis数据结构: String—字符串（key-value 类型）
Hash—字典(hashmap) Redis的哈希结构可以使你像在数据库中更新一个属性一样只修改某一项属性值
List—列表 实现消息队列
Set—集合 利用唯一性
Sorted Set—有序集合 可以进行排序
可以实现数据持久化

139.请问redis的List能在什么场景下使用?

Redis 中list的数据结构实现是双向链表，所以可以非常便捷的应用于消息队列（生产者 / 消费者模型）。消息的生产者只需要通过lpush将消息放入 list，消费者便可以通过rpop取出该消息，并且可以保证消息的有序性。如果需要实现带有优先级的消息队列也可以选择sorted set。而pub/sub功能也可以用作发布者 / 订阅者模型的消息。

140.redis为什么是单线程?

因为CPU不是Redis的瓶颈。Redis的瓶颈最有可能是机器内存或者网络带宽。既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了。缺点：服务器其他核闲置。

141.讲一下redis的主从复制怎么做的?

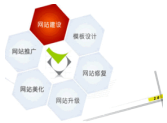
第一阶段：与master建立连接。

第二阶段：向master发起同步请求（SYNC）。

第三阶段：接受master发来的RDB数据。

第四阶段：载入RDB文件。

下一章链接：https://blog.csdn.net/weixin_38201936/article/details/93528040



我有个项目外包有人能做吗

项目外包