SMM面试

1. 什么是框架

2. Spring

2.1. 对Spring来说,最重要的是容器,容器管理着Bean的生命周期,控制着Bean的依赖注入

2.1.1. 两个容器接口

2.1.1.1. BeanBactory

2.1.1.2. ApplicationContext

2.1.1.2.1. ClassPathXmlApplication

2.1.1.2.2. FileSystemXmlApplication

2.1.1.2.3. WebXmlApplication

2.2. IOC和AOP

2.2.1. 配置分解+反射 代理的设计模式

2.3. 事务管理

2.3.1. 编程时事务管理

2.3.1.1. 使用TransactionTemplate

2.3.2. 声明式事务管理

2.3.2.1. 建立在AOP上,对方法前后进行拦截,将事务处理的功能编织到拦截的方法中 @Transactionnal

2.3.3. 最好用声明式,是非侵入式的开发公式.唯一不足的是只能到方法级别不能像编程式一样作用到代码块

2.4. AOP

2.4.1. 1事务处理 2权限判断 3日志管理

2.4.2. 静态代理

2.4.2.1. AspectJ:在编译阶段生成AOP代理类

2.4.3. 动态代理

2.4.3.1. 不会去修改字节码,而是在运行的时候在内存中临时为方法生成一个AOP对象

2.4.3.2. JDK动态代理

2.4.3.2.1. 通过反射来接收被代理的类 只提供接口的代理,不支持类的代理

2.4.3.3. CGLIB动态代理

2.4.3.3.1. 运行时动态地生成某个类的子类 可以在运行时生成指定类的一个子类对象,并覆盖其中特定代码进行增强

2.4.4. 术语

2.4.4.1. 通知

2.4.4.1.1. Before After After-returning After-throwing Around 定义了切面何时使用

2.4.4.2. 连接点

2.4.4.2.1. 可以插入一个切面的点

2.4.4.3. 切点

2.4.4.3.1. 匹配通知所要织入的一个或者多个连接点

2.4.4.4. 切面

2.4.4.4.1. 通知和切点的集合

2.5. IOC

2.5.1. 通常来说,我们需要使用依赖对象的功能的时候,需要自己去创建对象.而使用IOC的话,我们不用直接在代码里面组装组件和服务,而是在配置文件里面进行定义,之后又IOC容器进行依赖注入,将他们组装起来.

2.5.1.1. 1.解耦合(将bean之间的依赖关系转化为关联关系)

2.5.1.2. 由ioc容器来控制对象的生命周期和对象间的关系

2.5.2. 依赖注入的方式

2.5.2.1. 构造器注入

2.5.2.1.1. 通过容器触发一个类的构造器传入参数进行注入

2.5.2.2. setter注入

2.5.2.2.1. 容器实例化bean之后,调用该bean的setter方法进行注入

2.5.3. bean的作用域

2.5.3.1. singleton

2.5.3.1.1. bean在一个ioc容器中只有一个实例

2.5.3.2. prototype

2.5.3.2.1. 每次从容器中调用bean的时候都会返回一个新的实例

2.5.3.3. request:每一次http请求都会创建一个新的bean;session:同一个httpSession共享一个bean;globalSession:统一全局session共享一个bean

2.5.4. bean的生命周期

2.5.4.1. 1.实例化Bean

2.5.4.1.1. ApplicationContext容器启动结束后就实例化所有的bean(通过获取BeanDefinition对象中的信息进行实例化,实例化对象被包装在BeanWrapper中

2.5.4.2. 2.设置对象属性(依赖注入)

2.5.4.2.1. 根据BeanDefinition中信息进行依赖注入,通过BeanWrapper提供的接口完成依赖注入

2.5.4.3. 3.注入Aware接口

2.5.4.3.1. 如果实现了BeanNameAware,Spring传递bean的ID到setBeanName中

2.5.4.3.2. 如果实现了BeanFactoryAware接口,Spring传递beanfactory给setBeanFactory方法

2.5.4.4. 4.调用BeanPostProcessor

2.5.4.4.1. 如果对对象一些自定义的处理,可以通过BeanPostProcessor接口实现

2.5.4.4.2. 1.postProcessBeforeInitialzation:在InitiazationBean前执行,前置处理

2.5.4.4.3. 2.postProcessAfterInitialization,后置处理

2.5.4.5. 5.InitializingBean和init-method

2.5.4.5.1. 调用afterPropertiesSet()方法,增加我们自定义的逻辑

2.5.4.6. 6.DisposableBean

2.5.4.6.1. 调用destroy()方法在销毁之前执行指定逻辑

2.5.5. Spring中注入java集合

2.5.5.1. 1.list:一列值 2.set:一组值,不允许有相同的值 3.map:一组键值对,任意类型 4.props:一组键值对,只能String类型

2.5.6. 自动装配的方式

2.5.6.1. 1.no 2.byName 3.byType 4.constructor 5.autodetect

2.6. 注解

2.6.1. @Configuration

2.6.1.1. 表示该类当作一个bean

2.6.1.1.1. @bean

2.6.1.1.1.1. 表示方法返回一个对象

2.6.2. @Aurowire和@Resource

2.6.2.1. 自动注入,把setter/getter方法省略

2.6.2.1.1. // 该BeanPostProcessor将自动对标注了@Autowire的Bean进行注入

2.6.2.1.2. 在application.xml中加入

2.6.2.2. 前者byType注入,后者默认ByName注入,可以通过type属性使用ByType注入

2.6.3. @Component

2.6.3.1. 配置自动扫描包路径下的bean

2.6.3.1.1. 某个类的头上带有特定的注解@Component，@Repository，@Service，@Controller，就会将这个对象作为Bean注册进Spring容器。

2.6.3.1.2.

2.6.4. @ResponseBody

2.6.4.1. 将方法的返回值以特定的格式写入到response的body区域，进而将数据返回给客户端。当方法上面没有写ResponseBody,底层会将方法的返回值封装为ModelAndView对象。

3. Spring MVC

3.1. Spring MVC执行流程

3.2. 简述mvc模式

3.3. springmvc和structs

3.3.1. 1.核心控制器servlet和filter 2.前者基于方法，只有一个servlet实例；后者基于对象，每次都要实例化一个action

3.4. 重定向和转发

3.4.1. return "redirect:/xx" return "forward:/xx

3.5. post乱码问题

3.5.1. 在web.xml中配置CharacterEncoFilter过滤器,设置成**utf-8;**

3.5.2. Get乱码:1.修改tomcat配置文件 2.对参数进行重新编码

3.6. 常见问题

3.6.1. 怎么取得request或者session:直接在方法的形参中声明就会自动传入

3.6.2. 方法里得到前台参数:直接声明,名称要一样

3.6.3. 多个参数属于同一对象:直接声明该对象

4. MyBatis

4.1. ORM(对象关系映射)框架, 封装了JDBC,只需要关注sql本身

4.2. 动态SQL

4.2.1. 对于一些查询可能会指定多个查询条件,但不一定;需要动态生成sql语句:if trim where set foreach

4.2.2. 一般通过if节点实现

4.3. mybatis的执行顺序

4.3.1. 1.创建SqlSessionFactory 2.通过SQLSessionFactory创建SqlSession 3.通过sqlsession执行数据库操作
4.session.commit()提交事务 5.调用session.close()关闭会话

4.4. 使用mappers接口调用时要求

4.4.1. 1.接口方法名要和mapper.xml中sql的id一样 2.方法返回类型要和mapper.xml中resultType相同 3.方法的输入类型要和xx中parameterType相同 4.方法类路径是xx中namespace

4.5. #{}和${}

4.5.1. #预编译处理,$字符串替换

4.6. 将结果封装成目标对象

4.6.1. 属性名和字段名做映射处理

4.6.1.1. 使用resultMap来映射

4.6.2. 别名处理

4.7. 一对一和一对多

4.7.1. 联合查询

4.7.1.1. 几个表联合查询,只查询一次

4.7.2. 嵌套查询

4.7.2.1. 先查一个表,通过表结果的外键id再去另一个表查询

4.8. 延迟加载

4.8.1. 只支持association和collection关联对象的延迟加载,a指的是一对一,c指的是一对多查询,可以通过lazyLoadingEnable=true/false配置

4.9. 模糊查询

4.9.1. 1.添加sql通配符 select * from where bar like #{value}

4.9.2. 2.sql语句拼接 select * from x where bar like **"%"#{value}"%"**

4.10. 在mapper中传递多个参数

4.10.1. 1.#{0} #{1}按照顺序 2.使用@param("username")注解 3.多个参数封装成map.put("xx",xx)

5. 注解

5.1. @Configuration

5.1.1. 表示该类当作一个bean

5.1.1.1. @bean

5.1.1.1.1. 表示方法返回一个对象

5.2. @Aurowire和@Resource

5.2.1. 自动注入,把setter/getter方法省略

5.2.1.1. // 该BeanPostProcessor将自动对标注了@Autowire的Bean进行注入

5.2.1.2. 在application.xml中加入

5.2.2. 前者byType注入,后者默认ByName注入,可以通过type属性使用ByType注入

5.3. @Component

5.3.1. 配置自动扫描包路径下的bean

5.3.1.1. 某个类的头上带有特定的注解@Component，@Repository，@Service，@Controller，就会将这个对象作为Bean注册进Spring容器。

5.3.1.2.

5.4. @ResponseBody

5.4.1. 将方法的返回值以特定的格式写入到response的body区域(http响应正文中)，进而将数据返回给客户端。不写的时候会将方法的返回值封装为ModelAndView对象

5.4.1.1. 注解实现将conreoller方法返回对象转化为json对象响应给客户。

5.5. @RequestBody：注解实现接收http请求的json数据，将json转换为java对象。

5.6. @RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径。

5.7. @Component:@Controller @Service @Repository

6. 注解

6.1. @Configuration

6.1.1. 表示该类当作一个bean

6.1.1.1. @bean

6.1.1.1.1. 表示方法返回一个对象

6.2. @Aurowire和@Resource

6.2.1. 自动注入,把setter/getter方法省略

6.2.1.1. // 该BeanPostProcessor将自动对标注了@Autowire的Bean进行注入

6.2.1.2. 在application.xml中加入

6.2.2. 前者byType注入,后者默认ByName注入,可以通过type属性使用ByType注入

6.3. @Component

6.3.1. 配置自动扫描包路径下的bean

6.3.1.1. 某个类的头上带有特定的注解@Component，@Repository，@Service，@Controller，就会将这个对象作为Bean注册进Spring容器。

6.3.1.2.

6.4. @ResponseBody

6.4.1. 将方法的返回值以特定的格式写入到response的body区域(http响应正文中)，进而将数据返回给客户端。不写的时候会将方法的返回值封装为ModelAndView对象

6.4.1.1. 注解实现将conreoller方法返回对象转化为json对象响应给客户。

6.5. @RequestBody：注解实现接收http请求的json数据，将json转换为java对象。

6.6. @RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径。

6.7. @Component:@Controller @Service @Repository