# Springboot面试题

**默认的日志方式**

默认使用logback作为日志框架，是log4j的改进版本

**Spring如何实现两bean按顺序加载**

@DependsOn：指定Bean创建的顺序

@Order：指定Bean注入的顺序

**Spring中bean的作用域**

Singleton(单例)

prototype(多例)

request(同一次请求一个实例)

session(同一会话一个实例)

globalSession

**什么是注解，使用场景**

提供类，方法，属性的附加信息，通过反射机制在运行时读取注解，并完成注解中的操作

用于类上： @Controller @Service

用于方法上: @PostMapping

用于属性上: @Autowried

**spring AOP中代理使用的默认策略**

目标对象实现了接口，则默认使用JDK动态代理

目标对象没有实现接口，则默认使用CGLIB动态代理

目标对象实现了接口，程序里面也可以指定使用CGLIB动态代理

**使用spring框架的好处**

提供bean的管理，aop,ioc

IOC:控制反转，spring管理我们需要注入的bean,方便;依赖注入，

**Springboot自动装配**

springboot会扫描主程序所在包和子包里的配置类，通过反射生成对应的bean，交给spring容器管理

**Springboot如何处理事务**

使用@Transactional(rollbackFor = Exception.class)

**Slf4j和log4j的区别**

SLF4J：日志门面，提供日志接口，便于在不同日志框架间切换。调用 Log4j记录日志

Log4j: 具体的日志框架，提供丰富的日志功能。

# Mybatis

**mybatis一级缓存和二级缓存**

一级缓存：默认开启，同sqlsession中,相同的sql会直接去缓存中获取结果，增/删/改/commit()/close()后清空缓存

二级缓存：mapper级别，同mapper下共享缓存，不推荐使用，增/删/改后清空缓存

**Mybatis如何防止SQL注入，**#{}和${}的区别

#{} 编译时会把参数用？占位符替代，编译好后，再用参数替换

${} 将参数拼接在sql上

**当实体类中的属性和表中的字段名不一样怎么办**

1.在查询sql中为字段值定义别名，让别名和实体类属性名一致

2.通过 <result property = “” column =””/> 将实体类属性名和字段名一一映射对应

**如何获得自动生成的主键值**

主键采取自增策略后，在insert方法执行后主键就被传入到插入参数对象中

# Mybatisplus

**Mybatis框架有什么好处**

继承 BaseMapper 接口，即可直接使用如 insert、deleteById、updateById、selectById 等方法,内置分页插件，支持逻辑删除，自动填充创建时间，更新时间等。内置缓存机制，减少和数据库的访问，安全性更高(防止sql注入)

**Spring优点**

轻量级javaEE框架

包含依赖注入，声明式事务，AOP

好处：轻量级；控制反转(IOC)：实现松散耦合；面向切面编程(AOP)：将业务逻辑和系统服务区分开

**aop的通知方式**

前置通知：方法执行前

后置通知：方法执行后(不管方法执行成功/失败)

返回通知：方法成功执行后

异常通知：方法抛出异常后

环绕通知：方法执行前后

**Aop中的织入**

将切面逻辑，插入到连接点的过程，spring aop在运行时进行织入

**编程式事务和声明式事务**

声明式事务：注解或 XM管理事务

编程式事务：写代码的方式管理事务

**@Transactional事务失效的情况**

同个类中直接调用方法会失效，将自身作为bean注入

当前对象是new的，不是spring注入进来的

方法修饰符不是public, 需要改成public

默认只在运行时异常回滚，出现受检异常不会回滚，添加rollbackFor = Exception.class

方法中捕获异常后没有抛出，spring认为程序正常，没有回滚

**Spring Bean的注入方式**

将bean注入容器的方式

1.配置类(@Configuration)中使用@Bean表示将Bean注入IOC容器中

2.通过组件注解@Component @Controller @Service @Repository 将Bean类注入容器中

3.import注解导入

4.使用FactoryBean接口

**将IOC容器中的Bean注入到类中**

通过@Autowired/@Resource注解 进行属性注入，set方式注入，构造方法注入

**自动装配方式有哪些**

no:不自动装配

byName:根据Bean的名字进行自动装配

byType:根据Bean的类型进行自动装配

constructor：Bean与构造器的参数类型相同进行自动装配

autodetect:先使用constructor进行装配，失败再使用byType

使用注解@Autowired

**Bean的生命周期**

实例化Bean

设置对象属性(依赖注入)

处理Aware接口

BeanPostProcessor

initializingBean于init-method

DisposableBean

destroy-method

**spring如何解决循环依赖**

依赖分类

构造函数注入形成的循环依赖，在构造函数中完成初始化（无法解决）

setter注入构成的循环依赖，在setter方法中完成初始化（可以解决）

prototype(多例作用域)中bean的循环依赖（无法解决）

解决依赖的方式：

通过延迟注入属性

spring中的三级缓存

一级缓存 singletonObject 完全实例化，属性赋值完成的单例对象缓存

二级缓存 earlySingletonObjects 提前曝光的单例对象缓存，尚未完成属性的Bean,用于检测循环依赖

三级缓存 singletonFactories 单例对象工厂的缓存，加入前提是执行了构造函数

解决缓存的具体过程

实例化对象A,调用A的构造函数，放入三级缓存

往A中注入依赖，需要对象B

实例化对象B,调用B的构造函数，放入三级缓存

往B中注入依赖，需要对象A,并且A在三级缓存中，并将A从三级缓存中删除，放入二级缓存

B依赖注入完成，将B从二级和三级缓存中删除，放入一级缓存

将B注入给A,继续注入其他依赖，直到注入完成

A注入完后，将A从二级和三级缓存中删除，放入一级缓存

循环依赖处理结束

以下是 Spring 使用三级缓存解决循环依赖的详细流程：

开始创建 Bean A：

检查 singletonObjects 缓存中是否已有 Bean A 的实例。

如果没有，开始实例化 Bean A，并将 Bean A 的工厂对象放入 singletonFactories 缓存。

实例化 Bean A：

调用 Bean A 的构造器，创建 Bean A 的实例，但不填充依赖。

将创建的 Bean A 实例（部分初始化状态）放入 earlySingletonObjects 缓存。

标记 Bean A 为正在创建中。

解析 Bean A 的依赖：

当解析到 Bean A 依赖 Bean B 时，尝试从 singletonObjects 获取 Bean B。

如果 Bean B 尚未创建，进入 Bean B 的创建流程。

开始创建 Bean B：

检查 singletonObjects 缓存中是否已有 Bean B 的实例。

如果没有，开始实例化 Bean B，并将 Bean B 的工厂对象放入 singletonFactories 缓存。

实例化 Bean B：

调用 Bean B 的构造器，创建 Bean B 的实例，但不填充依赖。

将创建的 Bean B 实例（部分初始化状态）放入 earlySingletonObjects 缓存。

标记 Bean B 为正在创建中。

解析 Bean B 的依赖：

当解析到 Bean B 依赖 Bean A 时，不从 singletonObjects 获取，而是从 earlySingletonObjects 获取 Bean A 的早期引用。

继续创建 Bean B：

使用从 earlySingletonObjects 中获取的 Bean A 的早期引用，完成 Bean B 的依赖填充。

完成 Bean B 的初始化，包括调用初始化方法等。

将完全初始化的 Bean B 放入 singletonObjects 缓存，并从 earlySingletonObjects 和 singletonFactories 中移除 Bean B 的条目。

继续创建 Bean A：

使用从 earlySingletonObjects 中获取的 Bean B 的早期引用，完成 Bean A 的依赖填充。

完成 Bean A 的初始化，包括调用初始化方法等。

将完全初始化的 Bean A 放入 singletonObjects 缓存，并从 earlySingletonObjects 和 singletonFactories 中移除 Bean A 的条目。

**@Lazy解决循环依赖**

Spring 在第一次真正需要该 Bean 时才初始化它，避免循环依赖

**springboot中解决循环依赖的方式**

方式1：使用Autowired注解,让spring决定合适的注入时机

方式2：使用@Lazy注解 延迟加载依赖，注入依赖时，先注入代理对象，当首次使用时再创建对象完成注入

方式3 不使用构造函数注入，改为使用setter方式注入,当依赖最终被使用时才进行注入

**IOC控制反转**（松耦合，更优良）

控制：不用new创建对象，将设计好的对象交给IOC容器来创建和管理生命周期

反转：不是对象主动获取依赖，而是容器帮我们查找对象，对象被动的接受依赖对象

步骤：定义bean的java类，使用xml配置文件，解析bean的配置，反射创建类

**DI依赖注入**

依赖关系写入配置文件中，即对象注入属性，需要再IOC的基础上完成

依赖注入方式：构造方法注入，setter方法注入，接口注入

**spring注解**

@Component 组件，没有明确的角色

@Service 在业务逻辑层使用（service层）

@Controller 在表现层使用，控制器的声明，返回的是跳转路径

@RestController -返回json数据

@Repository 在数据访问层使用（dao层）

@Autowired 按照byType类型自动注入

@Resource(name="cat")(看起来更清晰)有两个属性name和type, 默认byname自动注入

@@Transactional事务注解,原子性、一致性、隔离性、持久性

**BeanFactory和ApplicationContext的区别**

BeanFactory:spring中最低层的接口，提供简单的容器功能，实例化对象和拿对象

ApplicationContext:继承BeanFactory接口，更高级的容器，功能更多

**springmvc框架**

组件

前端控制器DispatcherServlet:接收请求，相应结果，

处理器映射器HandlerMapping：更具请求的URL来查找Handler

处理器适配器HandlerAdapter:

视图解析器ViewResolver进行 视图解析

视图jsp页面

**AOP原理和作用,使用场景**

AOP面向切面编程：将多个类的公共行为封装到一个模块中，减少重复代码

需要对多个对象引入一个公共行为，如记录日志，权限控制，缓存优化，事务管理等，减少代码重复

实现方式：

动态代理技术：利用截取消息

JDK动态代理:java提供的动态代理技术，在运行时创建接口的代理实例，在代理实例中织入代码

CGLib动态代理：采用字节码技术，在运行时创建子类代理实例，目标接口不存在时会采用该方式，在子类实例中织入代码

**SpringMVC执行流程**

用户请求发送给前端控制器

前端控制器收到请求，调用处理器映射器，找到对应的处理器，返回给前端控制器

前端控制器器调用处理器适配器

经过适配，调用处理器

处理器执行完返回ModelAndView

适配器将ModelAndView返回给前端控制器

前端控制器传给视图解析器进行解析返回具体View

前端控制器对view进行渲染，响应给用户

**springboot 框架**

@SpringBootApplication - 表面是主类-最重要

@Mapper

@Select

@Controller

@Autowired

@RequestMapping

@Data

**springboot解决循环依赖的方式**

方式1：使用Autowired注解,进行属性(field)注入，让spring决定合适的注入时机

方式2：使用@Lazy注解 延迟加载依赖，注入依赖时，先注入代理对象，当首次使用时再创建对象完成注入

方式3 不使用构造函数注入，改为使用setter方式注入,当依赖最终被使用时才进行注入

**采用MVC分层的好处**

开发人只关注某一层，不同层可以复用，降低不同层间的耦合，开发更标准

**springboot和spring的关系**

简化spring应用程序的创建和配置，可以快速构建应用，减少了xml文件的配置

**IOC控制反转（松耦合，更优良），使用场景**

控制：不使用传统的new进行创建对象，而是将设计好的对象交给IOC容器来控制创建和管理生命周期

反转：不是对象主动获取依赖，而是容器帮我们查找对象，对象被动的接受依赖对象

步骤：定义bean的java类，使用xml配置文件，解析bean的配置，反射创建类

DI依赖注入

依赖关系写入配置文件中，即对象注入属性，需要再IOC的基础上完成

依赖注入方式：构造方法注入，setter方法注入，接口注入

**ioc容器在什么时候初始化bean**

在spring启动时，查找并加载需要被Spring

**springboot中自动装配方式有哪两种**

@Autowired注解：只按照类型进行注入

@Resource注解：默认按照名字注入，找不到则按照类型注入

**spring如何解决循环依赖**

1. 改为使用setter方式注入

2. 使用@Lazy注解

3. 使用PostConstruct注解

对一个字段+，一个字段-如何保证原子性，（例如，一个账户-50元，一个账户+50元）

通过事务来保证两个操作同时完成/同时没完成，@Transaction

**springmvc中常用的注解**

@Controller:接收请求，返回页面

@RestController:包含@Controller和@ResponseBody，接收请求，返回json结果

@RequestMapping:指定请i去路径

@GetMapping、@PostMapping、@PutMapping、@DeleteMapping：指定请求方式和路径

@PathVariable:接收路径中的参数

@RequestParam：接收param参数

@RequestBody: 接收body参数

**controller中如何对外暴露接口，外部如何访问接口**

在控制层使用@RequestMapping指定访问路径和方法方式 来对外暴露接口

外部通过url来访问接口

**controller单例的好处**

减少系统的开销，不需要多例，单例是线程不安全的

不要在controller中定义成员变量，万一必须要定义一个非静态成员变量时候，则通过注解@Scope("prototype")，将其设置为多例模式

**mybatis一级缓存和二级缓存**

默认情况下只开启一级缓存

一级缓存

在同一个SqlSession中，执行相同的SQL查询，第一次回去查询数据库，并放入缓存

第二次相同的查询会从缓存中取，缓存中找不到再去数据库中查询

发生了曾删该操作后缓存会被清空

二级缓存

是mapper级别的，默认没有开启，第一次调用mapper下的

为什么不使用原生jdbc,而选择使用mybatis框架[1]

原生jdbc没办法实现java代码和sql之间的解耦

每次都要自己创建连接，使用起来繁琐，存在冗余代码

查询起来特别麻烦

**mybatis resultType和resultMap的区别**

resultType:实体类字段和数据库字段一一对应时使用

<select id="query" resultType="student">

select \* from student

</select>

resultMap:实体类字段和数据库字段不对应时使用

<select id="query" resultMap="studentResult">

select \* from student

</select>

<resultMap type="student" id="studentResult">

<result property="studentId" column="student\_id"/>

<result property="studentName" column="student\_name"/>

</resultMap>

**mybatis中 #{} 和 ${} 的区别**

#{}参数占位符：sql预编译（将参数位置用？来替换），对应的变量自动加上单引号，防止sql注入

${}字符串拼接符：sql拼接，对应的变量不会加上单引号

**mybatis中用的标签**

foreach标签

<foreach item="id" collection="ids" open="(" close=")" separator=","></foreach>

[collection](https://so.csdn.net/so/search?q=collection&spm=1001.2101.3001.7020):指定数据的类型,（如果ids是字符串,也可以用ids.split(',')分割成字符串数组)  
item：遍历时生成的对象  
open：开始遍历时拼接的[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020)  
close：结束遍历时拼接的字符串  
separator：拼接sql使用的分隔符

include标签

<sql id="fields">

id,user\_name

</sql>

<select id="user.selectbyId" resultType="User">

select

<include refid="fields"/>

from user

</select>

choose标签

<select id="user.selectbyId" resultType="User" parameterType="User">

select

<include refid="fields"/>

from user where 1=1

<choose>

<when test="state == 1">

and state in (1,2,3)

</when>

<when test="state == 2">

and state in (4,5,6)

</when>

<otherwise>

and state in (7,8,9)

</otherwise>

</choose>

</select>

trim标签

prefix：拼接sql的前缀  
suffix：拼接sql的后缀  
prefixOverrides：去除sql语句前面的关键字符  
suffixOverrides：去除sql语句后面的关键字符

<select id="user.selectbyId" resultType="User" parameterType="User">

select \* from user

<trim prefix="WHERE" prefixOverrides="AND | OR">

<if test="name != null and name != ''"> AND user\_name like '%${name}%'</if>

<if test="sex != null and sex != ''"> AND sex = #{sex}</if>

</trim>

</select>

<!-- 假如说name和sex的值都不为空，SQL：select \* from user WHERE user\_name like '%xx%' and sex = 'x' -->

**Springboot中@Autowired、@Inject、@Resource的区别**

@Autowired:根据类型自动装配，多个符合类型的bean通过@Qualifier指定bean的名称

@Inject: 根据类型自动装配

@Resource: 根据名称自动装配，

**Spring中事务失效的场景**

调用同类的方法，自己类中注入自己

类不是自动注入的，而是new出来的,没有被spring管理

Catch住异常后，没有向上抛出，rollbackfor = Exception.class

方法不是public