**spring、springMVC、springBoot**

Spring是基础框架，提供ioc和aop等功能

SpringMVC是Spring的Web模块，用于构建Web应用，处理http请求并返回数据

SpringBoot是Spring的扩展，简化了Spring应用的开发和部署，提供了自动配置、内嵌服务器（如 Tomcat）, 通过Starter起步依赖，简化jar包的引用

**spring优缺点？**

优点：Spring提供ioc和aop，方便集成各种优秀框架

缺点：Spring依赖反射，反射影响性能

**spring 有哪些主要模块？**

spring core：框架的最基础部分，提供 ioc 和依赖注入。

spring aop：提供面向切面编程

spring beans：提供BeanFactory，Spring将管理对象称为Bean。

spring test：提供测试的支持

spring Web：提供针对 Web 开发的集成特性

spring context：构建于 core的基础上，提供了一种框架式的对象访问方法。

**Spring框架中用到了哪些设计模式？**

代理模式：Spring AOP功能的实现。单例模式：Spring中的bean默认都是单例的。

工厂模式：Spring使用工厂模式通过BeanFactory和ApplicationContext创建bean对象

**BeanFactory 和 ApplicationContext有什么区别？**

两者都可以当做Spring的容器。ApplicationContext是BeanFactory的子接口。BeanFactroy延迟加载Bean，只有在使用到某个Bean时，才对该Bean进行实例化。

ApplicationContext在容器启动时一次性创建了所有的Bean。

**ApplicationContext通常的实现是什么？**

FileSystemXmlApplicationContext ：此容器从一个XML文件中加载beans的定义

ClassPathXmlApplicationContext：此容器从一个XML文件中加载beans的定义，将在classpath里找bean配置。

AnnotationConfigApplicationContext：基于java配置类来创建

AnnotationConfigServletWebServerApplication: 基于java配置类来创建，并且还可以用于web环境

**ioc**

IOC，控制反转，由主动new产生对象转换为由Ioc容器提供对象，对象创建控制权由程序转移到外部容器。由spring控制对象的生命周期和对象间的关系。

**控制反转(IoC)有什么作用**

管理对象的创建和依赖关系的维护。

解耦，由容器去维护具体的对象

**Spring IoC 的实现机制**

Spring 中的 IoC 的实现原理就是工厂模式加反射机制

**Spring 的 IoC支持哪些功能**

依赖注入、自动装配、指定初始化方法和销毁方法

**Spring的依赖注入**

依赖注入是当某个Java实例需要另一个Java实例时，创建被调用者的工作由Spring容器来完成，然后注入调用者。

在容器中建立bean与bean之间的依赖关系的整个过程，称为依赖注入

**依赖注入的基本原则**

容器全权负责组件的装配，把符合依赖关系的对象通过属性setter或者是构造器传递给需要的对象。

**Spring beans**

Spring bean 表示受到Spring管理的对象，它们被Spring IOC容器初始化，装配，和管理。

被Ioc容器创建或管理的对象统称为bean

**一个Spring Bean定义包含什么？**

包含如何创建一个bean，它的生命周期及它的依赖。

**依赖注入方式**

setter 属性注入

构造方法注入

**Spring中的bean的作用域有哪些？**

1.singleton：唯一bean实例，Spring中的bean默认都是单例的。

2.prototype：每次请求bean都会创建一个新的bean实例。

3.request：每次HTTP请求都会创建一个bean，该bean仅在当前HTTP request内有效。

4.session：每次HTTP请求都会创建一个bean，该bean仅在当前HTTP session内有效。在⼀个http session中，定义⼀个Bean实例

5.application：在⼀个http servlet Context中，定义⼀个Bean实例

**spring 中的bean是线程安全的吗？**

对于单例Bean，所有线程都共享一个单例Bean，存在资源的竞争，不是线程安全的。

controller、service和dao层本身并不是线程安全的，但是如果只是多线程调用一个实例的方法，会在内存中复制变量，这是线程各自的工作内存，是安全的。

大部分时间Bean是无状态的，是线程安全的，比如Dao。有状态就是有数据存储功能，无状态就是不会保存数据。

**Spring如何处理线程并发问题？**

ThreadLocal会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal。

**Spring中的bean生命周期**

大体为：实例化——>属性注入——>初始化——>销毁

解析注解配置的类，得到BeanDefinition

通过BeanDefinition反射创建Bean对象

对Bean对象进行属性填充

回调实现了Aware接口的方法，如BeanNameAware

调用BeanPostProcessor的初始化前方法

调用init初始化方法

调用BeanPostProcessor的初始化后方法，此处会进行AOP

将创建的Bean对象放入一个Map中

业务使用Bean对象

Spring容器关闭时调用DisposableBean的destroy()方法

**@Autowired注解自动装配流程**

使用@Autowired会先在容器中查询对应类型的bean：

如果查询结果只有一个，直接装配；

如果查询结果不止一个，那么@Autowired会根据变量名称来查找；

如果查询结果为空，那么会抛出异常。解决方法时，使用required=false。

**自动装配有哪些局限性？**

不能自动装配简单的属性，如基本数据类型

**自动装配**

Ioc容器根据bean所依赖的资源在容器中自动查找并注入到bean中的过程称为自动装配

**spring自动装配bean有哪些方式？**

按名称，根据 bean 的名称注入对象。必须保障容器中具有指定名称的bean

按类型，根据类型注入对象。必须保障容器中相同类型的bean唯一

spring 配置文件中 <bean> 节点的 autowire 参数可以控制 bean 自动装配的方式

no：默认值，表示没有自动装配，应使用显式 bean 引用进行装配

byName：它根据 bean 的名称注入对象依赖项。

byType：它根据类型注入对象依赖项。

构造函数：通过构造函数来注入依赖项，需要设置大量的参数。

autodetect：容器首先通过构造函数使用 autowire 装配，如果不能，则通过 byType 自动装配。

**Spring注解**

@Component、@Controller、@Service、@Repository使用在类上用于实例化Bean

@Autowired 使用在字段上用于根据类型依赖注入

@Qualifier 结合@Autowired一起使用用于根据名称依赖注入

@Resource 相当于@Autowired+@Qualifier，按照名称进行注入

@Value 注入普通属性

@Scope 标注Bean的作用范围

@PostConstruct标注该方法是Bean的初始化方法

@PreDestroy标注该方法是Bean的销毁方法

@Configuration 指定一个配置类，当创建容器时会从该类上加载注解

@ComponentScan指定初始化容器时要扫描的包。

作用和在 Spring 的 xml 配置文件中的

<context:component-scan base-package="com.itheima"/>一样

@Bean标注将方法的返回值存储到Spring容器中

@PropertySource 用于加载.properties 文件中的配置

@Import 用于导入其他配置类

**什么是AOP**

AOP，面向切面编程，就是把可重用的功能提取出来，然后将这些通用功能在合适的时候织入到应用程序中，AOP可以拦截指定的方法并且对方法增强。比如事务管理、权限控制、日志记录、性能统计等。

**AOP的作用**

在程序运行期间，在不修改源码的情况下对目标方法进行功能增强。

**AOP的优势**

减少重复代码，降低了模块间的耦合度。

**解释一下Spring AOP里面的几个名词**

（1）切面（Aspect）：切面是切点和通知的结合。

（2）连接点（Join point）：连接点是在应用执行过程中能够插入切面的一个点。可以被增强的方法叫做连接点。

（3）通知/增强（Advice）：增强逻辑所在的方法叫做通知。通知指当到达特定切入点处应该附加执行的代码。

（4）切入点（Pointcut）：切点定义了需要在哪些连接点上执行通知。真正被增强的方法叫做切点

（5）引入（Introduction）：添加额外的字段或者方法到被通知的类

（6）目标对象（Target Object）：被通知的对象

（7）代理：一个类被AOP织入增强后，就产生一个结果代理类。

（8）织入（Weaving）：织入是把切面应用到目标对象并创建新的代理对象的过程。

**Spring通知有哪些类型？**

前置通知（Before）：指定增强的方法在切入点方法之前执行

后置通知（After-returning）：指定增强的方法在切入点方法之后执行

环绕通知（Around）：指定增强的方法在切入点方法之前和之后都执行

异常通知（After-throwing）：指定增强的方法在出现异常时执行，catch块

最终通知（After）：无论如何都会在切入点方法之后执行，finally块

**如何理解 Spring 中的代理？**

把切面应用到目标对象并创建新的代理对象称为代理。

Advice + Target Object = Proxy

**JDK动态代理和CGLIB动态代理的区别**

JDK动态代理只能对实现了接口的类生成代理

CGLIB是针对类实现代理，对指定的类生成一个子类，覆盖其中的方法（继承）可以代理没有实现接口的目标类。CGLIB在运行时通过动态生成目标类的子类来实现代理

**@Component和@Bean的区别是什么？**

作用对象不同。@Component注解作用于类，@Bean注解作用于方法。

@Bean可以将第三方库的类装配到Spring容器中

**spring 事务实现方式有哪些？**

声明式事务：在配置文件中配置（推荐使用），分为基于XML的声明式事务和基于注解的声明式事务

编程式事务：在代码中硬编码（不推荐使用）

**spring 的事务实现方式和隔离级别？**

在某个方法上增加@Transactional注解，就可以开启事务，这个方法中的所有sql都会在一个事务中执行，统一成功或失败

spring事务隔离级别就是数据库的隔离级别，外加一个默认级别，数据库默认是哪种就是哪种

Read uncommitted(读未提交)、Read committed(读已提交)、Repeatable read(可重复读)、Serializable(可串行化)

数据库配置的隔离级别会被spring覆盖

**Spring的事务传播行为(Seata)**

事务传播行为用来描述由某一个事务传播行为修饰的方法被嵌套进另一个方法的时事务如何传播。

public void methodA(){

methodB();

//doSomething

}

@Transaction(Propagation=XXX)

public void methodB(){

//doSomething

}

代码中methodA()方法嵌套调用了methodB()方法，methodB()的事务传播行为由@Transaction(Propagation=XXX)设置决定。这里需要注意的是methodA()并没有开启事务，某一个事务传播行为修饰的方法并不是必须要在开启事务的外围方法中调用。

REQUIRED：A方法调用B方法，B看A有没有事务，有就用A的事务，没有就新建一个事务（@Transactional，加在b方法上）

SUPPORTS：A方法调用B方法，B看A有没有事务，有就用A的事务，没有就以非事务运行

MANDATORY：A方法调用B方法，B看A有没有事务，有就用A的事务，没有就抛出异常

REQUIRED\_NEW：不管有没有事务都会创建一个新的事务

NEVER：B以非事务运行，如果A存在事务，则抛出异常

NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按REQUIRED属性执行。

**Spring事务失效场景**

1、数据库本身不支持事务

2、事务方法所在的类没有加载到IOC容器中

3、方法没有被public修饰

4、方法用final修饰

5. 多线程调用。同一个事务，只能用同一个数据库连接。而多线程场景下，拿到的数据库连接是不一样的，即是属于不同事务。

在@Transactional注解中如果不配置rollbackFor属性,那么事务只会在遇到RuntimeException的时候才会回滚,加上rollbackFor=Exception.class,可以让事务在遇到非运行时异常时也回滚

**@RefreshScope**

通过 @RefreshScope 注解，可以在不重启应用的前提下动态刷新某些 Bean 的属性