

企业级开发框架知识问答分析-51 问

1 SpringBoot 脚手架分析与设计

1.1 SpringBoot 是什么？

Spring Boot 是一个全新的 Java 软件开发框架，很多人现在把它理解为一个脚手架。它基于快速构建理念，通过约定大于配置，开箱即用的方式，来简化 Spring 项目的初始搭建以及开发过程，提高开发效率。

1.2 SpringBoot 核心优势是什么？

SpringBoot 为我们的开发提供了起步依赖(Starter Dependency)、自动配置(Auto Configuration)、健康检查(Actuator)、嵌入式服务(Tomcat, Jetty)等核心特性，基于这些特性和优势可以更好的服务我们的开发过程。可以更好的简化项目构建、代码编写、项目配置、项目部署等，可以说 springboot 技术是大势所趋。

1.3 SpringBoot 核心配置文件的类型？

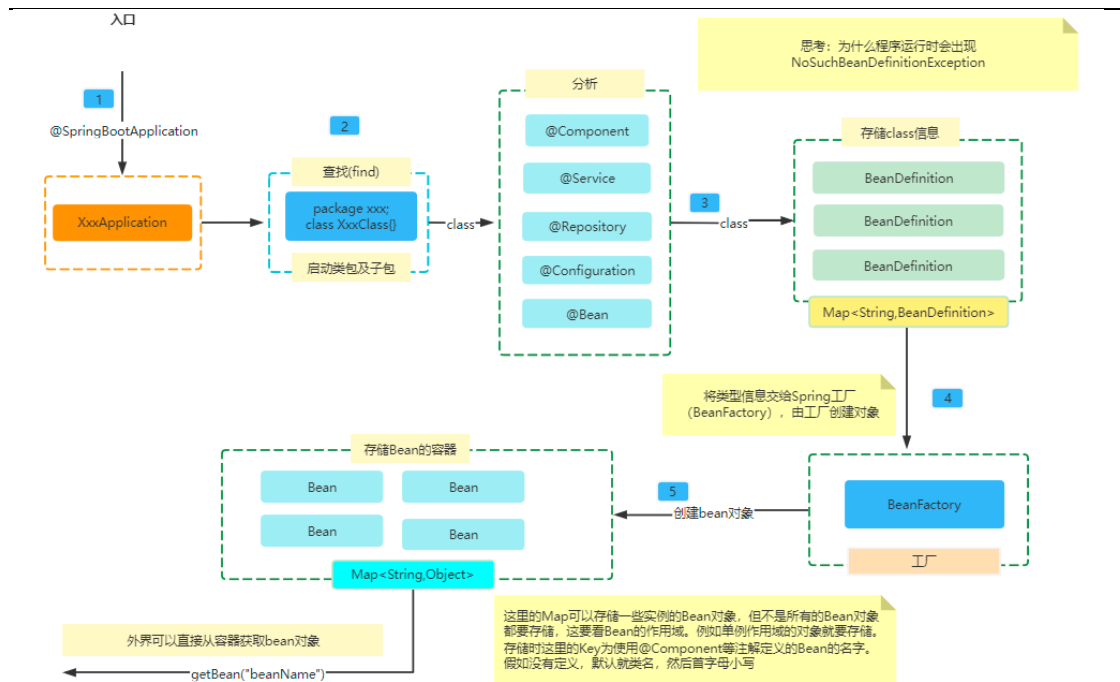
Spring Boot 提供了后缀为 properties、yml、factories 等类型的配置文件。

1.4 如何学习 SpringBoot 呢？

先知其然（要清楚该功能要解决什么问题），然后动手实践，逐步追奔溯源、知其所以然。

1.5 SpringBoot 的启动过程是怎样的？

在 SpringBoot 启动时，其大概过程，如图所示：



其基本启动过程描述如下:

- 1) 基于配置加载类(通过 ClassLoader 将指定位置的类读到内存->底层通过线程调用 IO 从磁盘读取到内存)。
- 2) 对类进行分析(创建字节码对象-Class 类型,通过反射获取器配置信息)。
- 3) 对于指定配置(例如由 spring 特定注解描述)的对象存储其配置信息(借助 BeanDefinition 对象存储)。
- 4) 基于 BeanDefinition 对象中 class 的配置构建类的实例(Bea 对象),并进行 bean 对象的管理(可能会存储到 bean 池)。

2 Spring 框架分析与设计

2.1Spring 是一个什么框架?

Spring 是一个资源整合框架,其核心是资源整合,然后以一种更加科学的方式对外提供服务,例如提高对象的应用效率,降低系统开销,提高代码的可维护性等等。其官方网址为 spring.io。

2.2Spring 框架中有哪些重要的模块?

2.3 Spring 框架是如何构建对象？

Spring 框架中所有 Bean 对象都是通过 BeanFactory 接口类型的工厂对象构建的，此工厂底层会基于 Java 中的反射技术进行实现，首先获取类的字节码对象，然后基于字节码对象获取构造方法对象，最后基于构造方法对象构建类的实例对象。

2.4 如何理解 spring 中 IOC 设计？

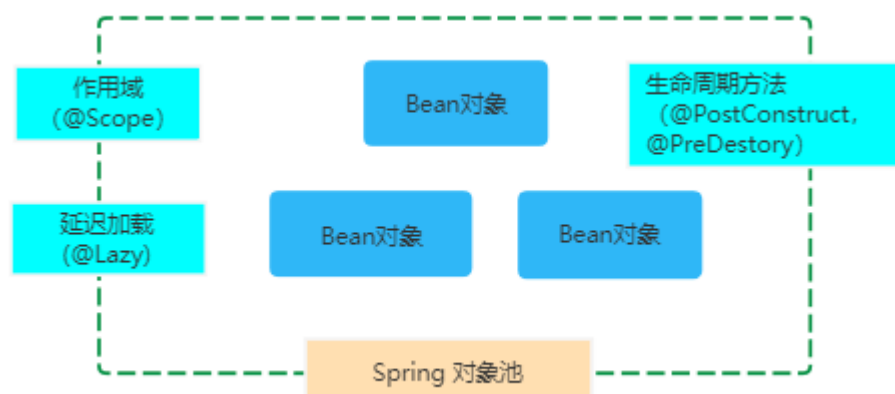
IOC 是一种设计思想，我们称之为控制反转，本质上讲解的是对象控制权问题。基于这种设计可以让初学者将对象的控制权转交给第三方，由第三方专业团队管理和应用对象。这样可以更好的避免对象的非正确使用方式，进而更好改善对象在内存中的科学应用。这个 IOC 设计从生活的角度可以理解为由股票操盘手负责帮你进行资金管理，由父母包办你的婚姻。再简单总结一下的话，IOC 可以理解为饭来张口、衣来伸手。

2.5 为什么要将对象交给 spring 管理？

Spring 为我们的对象赋予了很多个更加科学的特性，例如延迟加载，作用域，生命周期方法以及运行时的自动依赖注入(降低耦合，提高程序的可维护性)机制等，基于这些特性可以搞好的提高对象的应用性能以及程序的可扩展性。

2.6 Spring 框架中的 Bean 有什么特性？

Spring 框架为了更加科学的管理和应用 Bean 对象，为其设计相关特性，例如：懒加载(@Lazy)、作用域(@Scope)以及生命周期方法。



2.7 Spring 中用于定义 Bean 生命周期方法的注解？

- 1) @PostConstruct
- 2) @PreDestroy

2.8 Spring 中用于实现依赖注入的注解？

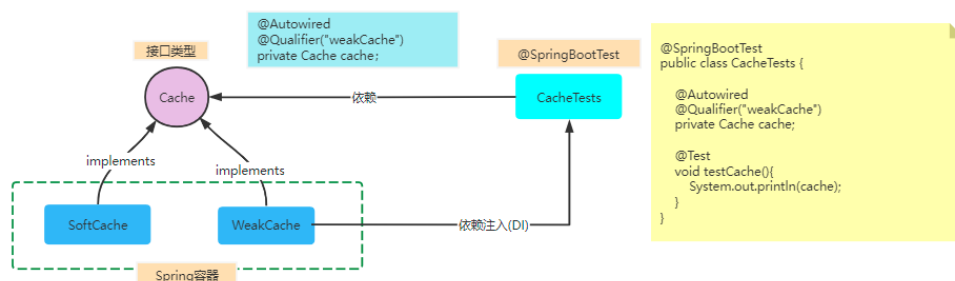
- 1) @Autowired
- 2) @Resource

2.9 @Autowired 注解有什么作用？

@Autowired 由 spring 框架定义，用于描述类中属性或相关方法(例如构造方法)。Spring 框架在项目运行时假如发现由他管理的 Bean 对象中有使用 @Autowired 注解描述的属性或方法，可以按照指定规则为属性赋值(DI)。其基本规则是：首先要检测容器中是否有与属性或方法参数类型相匹配的对象，假如存在并且只有一个则直接注入。其次，假如检测到有多个，还会按照 @Autowired 描述的属性或方法参数名查找是否有名字匹配的对象，有则直接注入，没有则抛出异常。最后，假如我们有明确要求，必须要注入类型为指定类型，名字为指定名字的对象还可以使用 @Qualifier 注解对其属性或参数进行描述(此注解必须配合 @Autowired 注解使用)。

2.10 描述下 @Qualifier 注解的作用？

@Qualifier 注解用于描述属性或方法参数，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 @Autowired 注解结合使用以消除这种混淆，指定需要装配的确切的 bean。



2.11 @Component 和 @Configuration 注解的异同点？

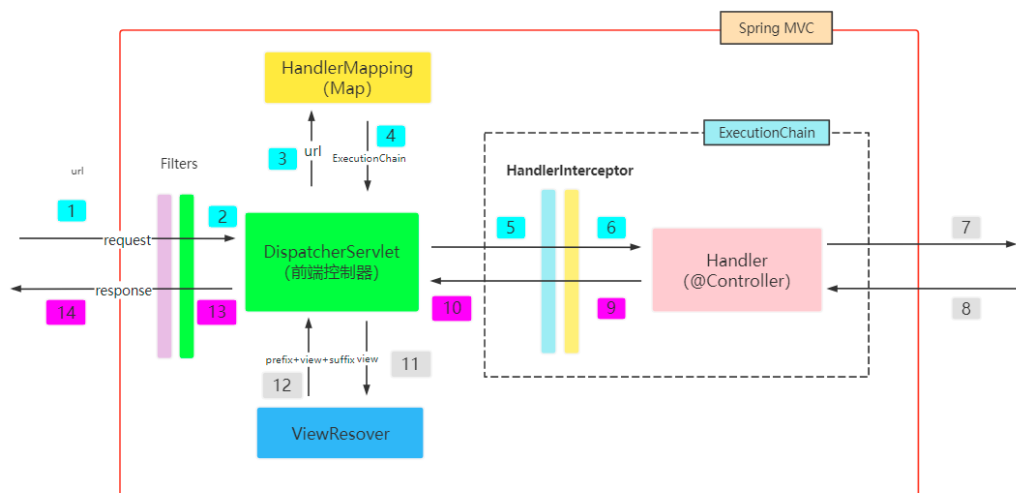
@Component 注解通常用于描述一般 bean 对象，比如具备一定通用性的对象。
@Configuration 注解通常用于描述 spring 工程中的配置类，是一个增强版的 @Component 注解，在配置类中定义一些由 @Bean 注解描述的方法，然后通过这些方法对一些自己定义或第三方的 Bean 进行对象的创建和初始化。当我们使用这 @Configuration 注解描述一个配置类时，Spring 框架底层会为这个 @Configuration 注解描述的配置类创建一个 CGLIB 代理对象，然后由代理对象调用 @Bean 注解描述的方法，同时底层会检测方法返回的 Bean 是否已创建，假如已创建则不再创建。使用 @Component 注解描述类时，系统底层并不会为此类创建代理对象，只是创建当前类的对象，然后调用 @Bean 注解描述的方法，创建和初始化 Bean，方法每调用一次就会创建一个新的对象。

2.12 说说你对 Spring 中 JdbcTemplate 的理解？

JdbcTemplate 是 Spring 中提供的一个封装了 JDBC 操作的模板类，此类中基于模板方法模式定义了很多 JDBC 模板方法，简化了 JDBC 的一些基本操作步骤，可以更好提高我们的开发效率。此类在运行时需要由 Spring 注入一个 DataSource 对象，然后基于 DataSource 可以获取一个连接池，从池中获取访问数据库的连接。

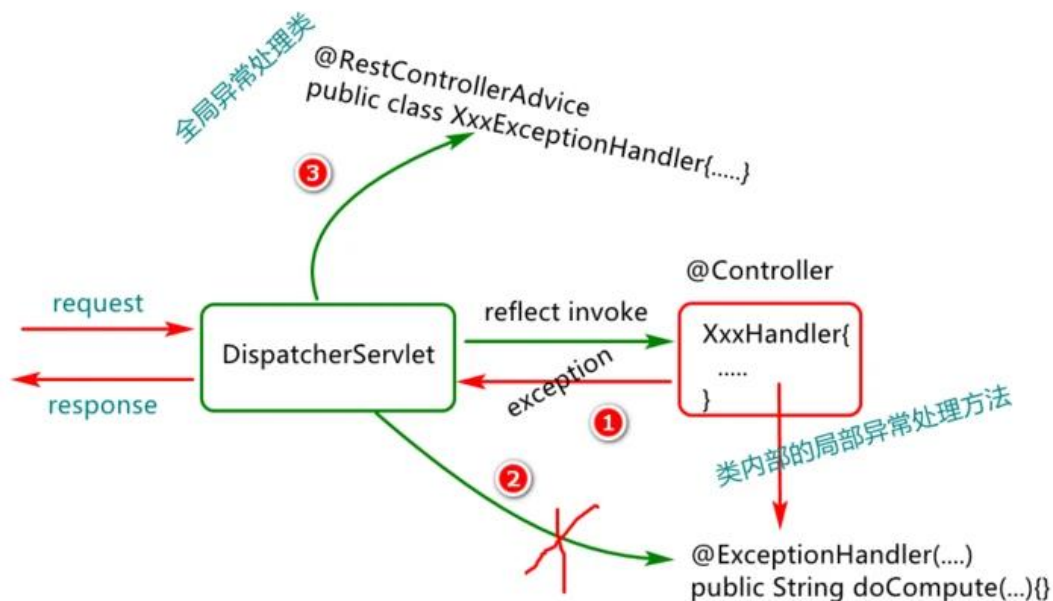
2.13 如何理解 Spring 中的 MVC 设计？

MVC 是一种分层架构设计思想，是 Model, View, Controller 的缩写，是为了将复杂问题简单化而提出的一种分而治之的设计套路，基于这种套路来提高代码的可维护性和可扩展性。Spring MVC 仅仅是 Spring 框架中 WEB 模块基于 MVC 设计思想的落地实现，简化了 web 请求和响应的处理过程。



2.14 Spring MVC 中的异常是如何处理的？

Spring 工程中 Web 模块可以基于 @RestControllerAdvice 注解定义全局异常处理类，然后借助全局异常处理规范进行异常处理，大概过程如图所示：



2.15 说几个 Spring 中用于定义组件的注解？

- 3) @Component
- 4) @Service
- 5) @Controller
- 6) @Configuration
- 7) @Repository
- 8) @Bean

2.16 说几个 Spring MVC 中描述方法的注解？

- 1) @RequestMapping
- 2) @PostMapping
- 3) @GetMapping
- 4) @DeleteMapping
- 5) @PutMapping
- 6) @PatchMapping
- 7) @ResponseBody

2.17 说几个 Spring MVC 中描述方法参数的注解？

- 1) @RequestParam
- 2) @PathVariable
- 3) @RequestBody
- 4) @RequestPart

2.18 说说 Spring MVC 中的拦截器？

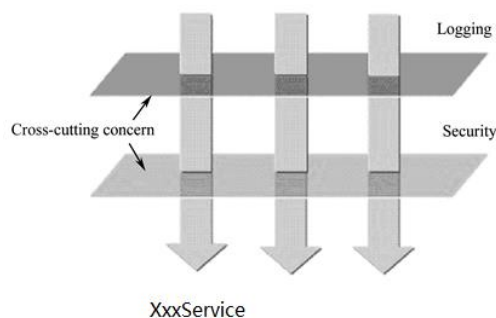
Spring 框架中定义的拦截器属于 Spring MVC 模块执行链中的一个对象，其类型为 `HandlerInterceptor`，基于这个类型的拦截器可以对后端 `@Controller` 注解描述的对象进行请求和响应的拦截。用于进行一些预处理操作。

2.19 @ResponseBody 注解的作用？

`@ResponseBody` 用于描述类或方法，用于告诉底层可以将方法的返回值转换为 json 格式的字符串，然后再响应到客户端。假如类中所有方法都需要使用 `@ResponseBody` 注解进行标记，则可以直接使用 `@RestController` 对类进行描述。

2.20 如何理解 Spring 中的 AOP 设计？

AOP (Aspect Orient Programming) 是一种设计思想，是软件设计领域中的面向切面编程，它是面向对象编程(OOP)的一种补充和完善。实际项目中我们通常将面向对象理解为一个静态过程(例如一个系统有多少个模块，一个模块有哪些对象，对象有哪些属性)，面向切面理解为一个动态过程(在对象运行时动态织入一些扩展功能或控制对象执行)。如图所示：



2.21 SpringBoot 工程中如何定义切面？

基于@Aspect 注解进行描述，并交给 Spring 管理。

2.22 SpringBoot 工程中的切面内部如何定义切入点？

基于@Pointcut 注解并借助相关表达式进行实现。

2.23 SpringBoot 工程中的切面内部可以定义哪些通知类型？

- 1) @Around
- 2) @Before
- 3) @After
- 4) @AfterReturning
- 5) @AfterThrowing

2.24 Spring 框架常用设计模式有哪些？

工厂模式、建造模式、策略模式、代理模式、单例模式、模板方法模式、享元模式、适配器模式、。。。

2.25 Spring AOP 实现原理是怎样的？

Spring 中 AOP 设计，是在系统启动时为目标类型创建子类或兄弟类型对象,这样的对象我们通常会称之为动态代理对象，然后通过动态代理对象为目标对象实现功能的扩展，如图所示：

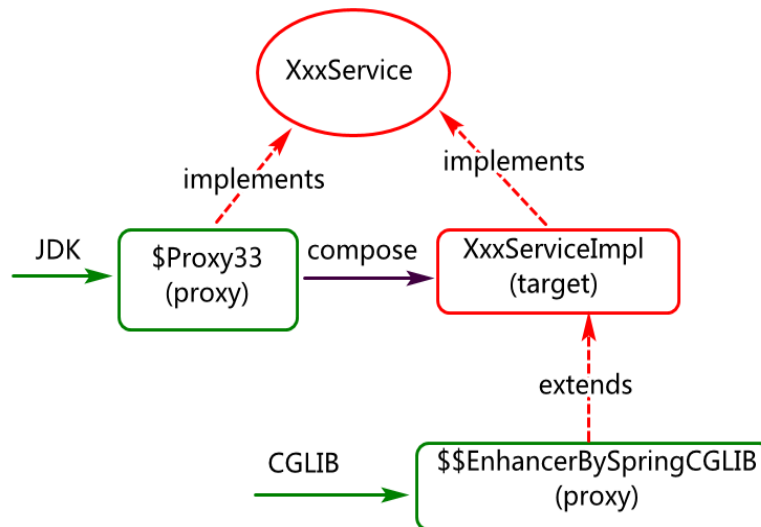


图-3

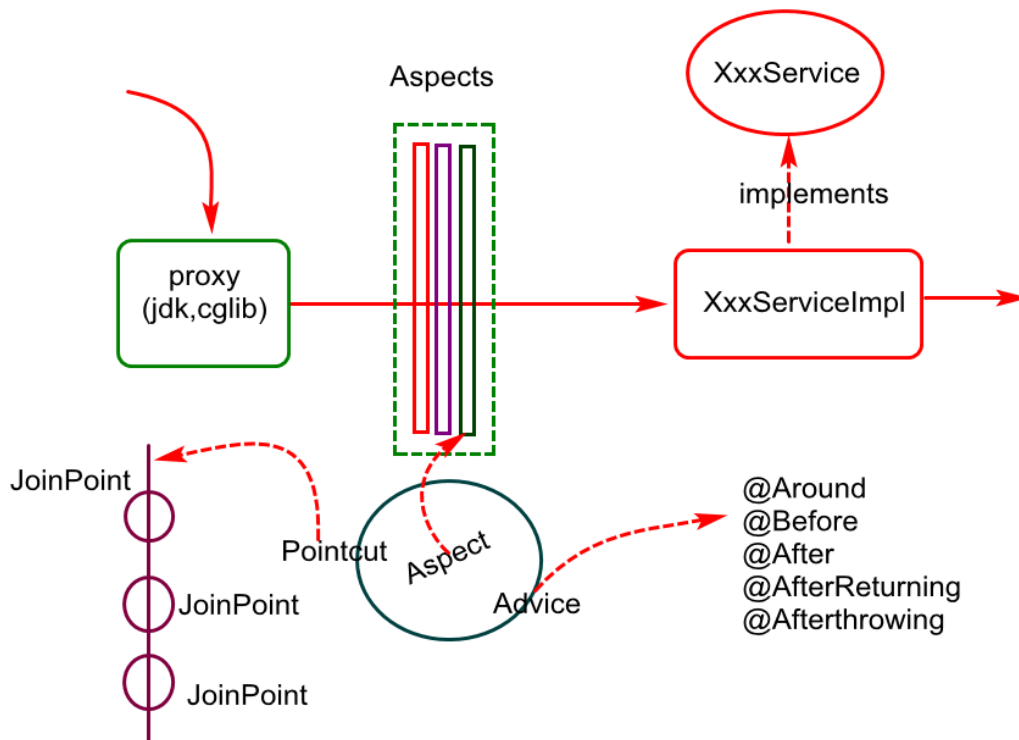
其中，为目标类型(XxxServiceImpl)创建其代理对象方式有两种：

第一：JDK 代理，借助 JDK 官方 API 为目标对象类型创建其兄弟类型对象，但是目标对象类型需要实现相应接口。

第二，CGLIB 代理，借助 CGLIB 库为目标对象类型创建其子类类型对象，但是目标对象类型不能使用 final 修饰。

2.26 Spring AOP 切面有哪些关键元素？

- 1) 切入点(pointcut):定义了切入扩展业务逻辑的位置(哪些方法运行时切入扩展业务),一般会通过表达式进行相关定义,一个切面中可以定义多个切入点。
- 2) 通知(Advice): 内部封装扩展业务逻辑的具体方法对象,一个切面中可以有多个通知(在切面的某个特定位置上执行的动作(扩展功能)。
- 3) 连接点(joinpoint):程序执行过程中,封装了某个正在执行的目标方法信息的对象,可以通过此对象获取具体的目标方法信息,甚至去调用目标方法。连接点与切入点定义如图所示:



我们可以简单的将机场的一个安检口理解为连接点，多个安检口为切入点，安全检查过程看成是 通知。总之，概念很晦涩难懂，多做例子，做完就会清晰。先可以按白话去理解。

▪ Spring AOP 的应用场景有哪些？

- 1) 日志记录
- 2) 事务控制
- 3) 权限控制
- 4) 缓存应用
- 5) 异步操作

2.27 Spring AOP 中用于描述通知方法的注解？

- 1) @Around
- 2) @Before
- 3) @After
- 4) @AfterThrowing
- 5) @AfterReturning

2.28 说说 Spring 框架中支持的事务方式？

Spring 框架中支持的事务方式主要有两种：

- 1) 编程式事务管理：这意味你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。
- 2) 声明式事务管理：这意味着你可以将业务代码和事务管理分离，你只需用注解和 XML 配置来管理事务。

2.29 Spring 中声明式事务的实现方式？

在类或方法上添加@Transaction 注解。

2.30 Spring 中定义了哪些事务隔离级别？

- 1) DEFAULT
- 2) READ_UNCOMMITTED
- 3) READ_COMMITTED
- 4) REPEATABLE_READ
- 5) SERIALIZABLE

2.31 SpringBoot 中如何启动异步任务？

- 1) 在启动类或配置上添加@EnableAsync 注解
- 2) 在执行异步方法的类上添加@Async 注解

2.32 Spring 中常见哪些设计模式有哪些？

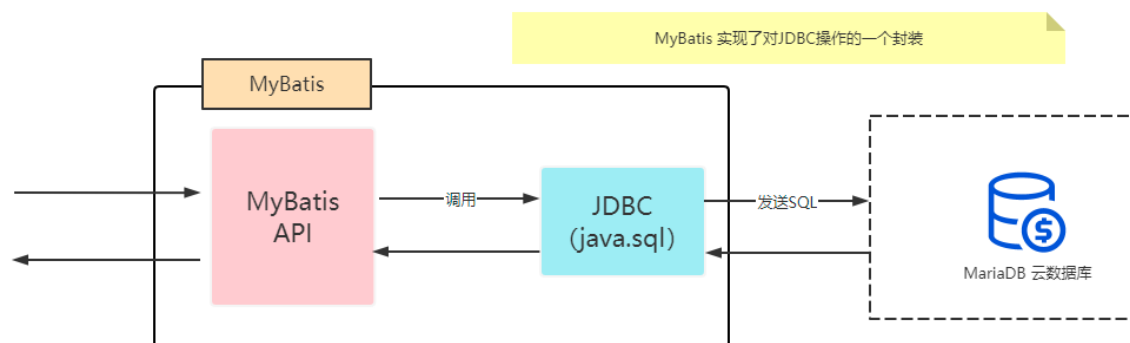
- 1) 建造模式：BeanDefinitionBuilder
- 2) 简单工厂模式：BeanFactory。
- 3) 工厂方法模式：ProxyFactoryBean。
- 4) 单例模式：单例 Bean 对象。
- 5) 代理模式：Aop Proxy。
- 6) 策略模式：AOP 代理策略, SimpleInstantiationStrategy。
- 7) 适配器模式：AdvisorAdapter。
- 8) 模版方法模式：JdbcTemplate。

- 9) 责任链模式: HandlerInterceptor。
- 10) 观察者模式: ApplicationListener

3 MyBatis 框架分析与设计

3.1 MyBatis 是什么？

MyBatis 是一个优秀的 Java 持久层(数据访问层)框架，由 apache 的 ibatis 演变而来，它通过 XML 或注解方式将对象与 SQL 关联起来，实现了对 JDBC 操作的封装，简化 JDBC 代码手动设置参数和手动进行结果集映射的过程。



3.2 MyBatis 常用官方网址？

- 1) <http://mybatis.org/mybatis-3> (mybatis 基础)
- 2) <http://mybatis.org/spring> (传统 spring 工程下 mybatis 的整合)
- 3) <http://mybatis.org/spring/boot.html> (sb 工程下 mybatis 的整合)

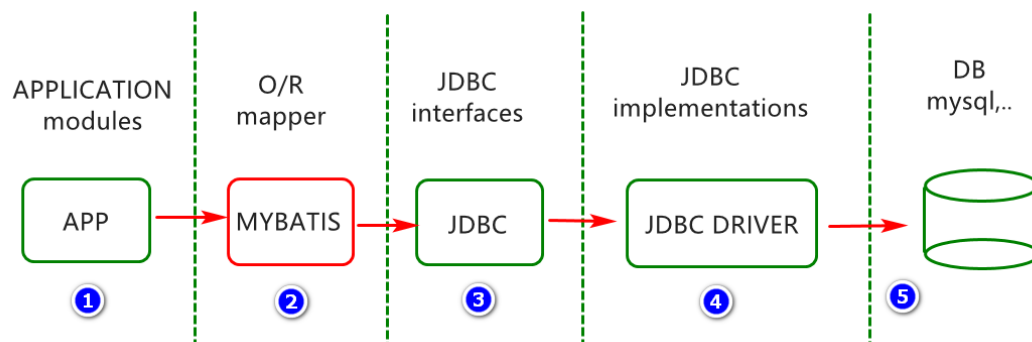
3.3 MyBatis 框架有什么优势？

- 1) 可以更好的实现 JAVA 代码与 SQL 语句的分离，提高可维护性。
- 2) 通过动态 SQL 可以更好的适配灵活的需求变更。
- 3) 通过 SQL 映射简化了 JDBC 代码中 SQL 参数及结果集处理。
- 4) 合理的架构设计，提高了系统的稳定性，访问性能，可扩展性。

3.4 MyBatis 框架有什么劣势？

- 1) SQL 语句编写的工作量相对较大。（相对 hibernate 框架）
- 2) SQL 语句依赖于数据库，移植性相对较差。（不是最大劣势）

3.5 MyBatis 的应用架构是怎样的？

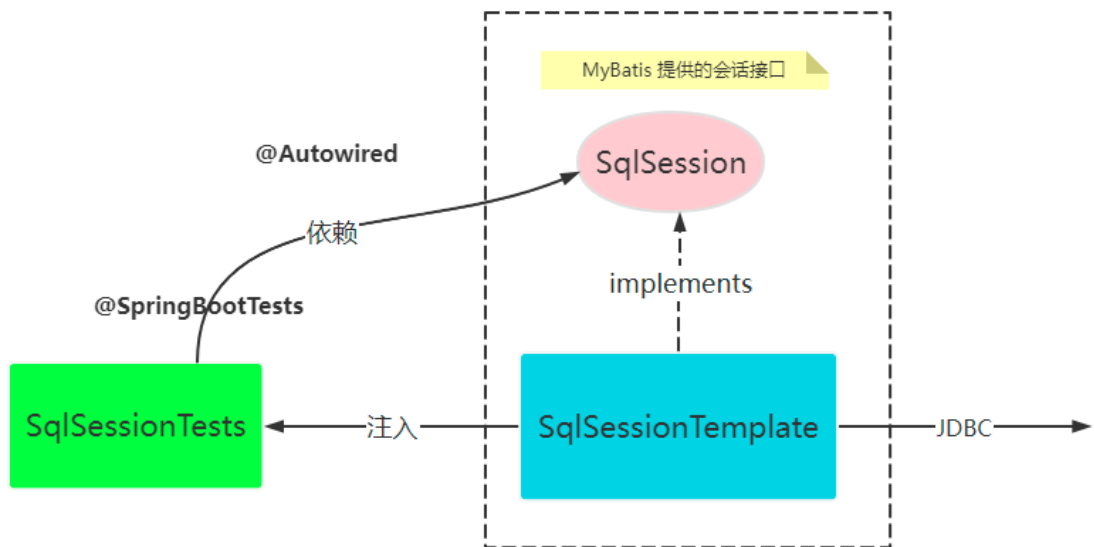


3.6 SpringBoot 工程下 MyBatis 框架的整合？

- 1) 确定已经配置好数据源(DataSource)
- 2) 添加依赖(mybatis-spring-boot-starter)
- 3) 基础配置(可选，基于业务实践有选择性的进行配置)
- 4) 创建数据访问接口及方法并定义 sql 映射
- 5) 定义单元测试类，对具体业务方法进行单元测试。

3.7 MyBatis 中的核心 API 有哪些？

- 1) SqlSessionFactory
- 2) SqlSession
- 3) DefaultSqlSession
- 4) SqlSessionTemplate



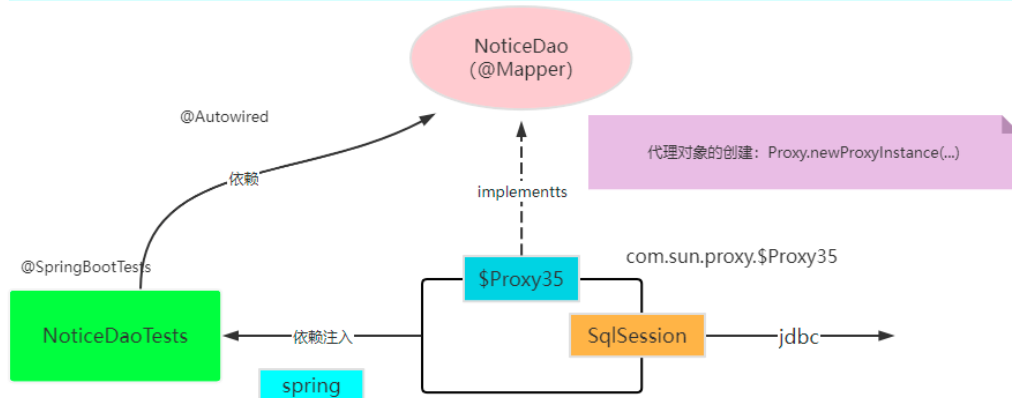
问题:

- 1) `SqlSessionTemplate` 应用在什么场景? (Spring 整合 mybatis 时, 默认使用的是 `SqlSessionTemplate` 对象实现与数据库的交互)
- 2) `SqlSessionTemplate` 对象用到了什么设计模式? (模板方法模式, 回顾 `JdbcTemplate` 对象)

3.8 MyBatis 中 @Mapper 注解的作用？

此注解由 mybatis 官方提供, 用于描述 "数据持久层接口", 当项目启动时, 系统底层会自动扫描启动类所在包以及子包中的类, 假如发现某个接口上使用 `@Mapper` 注解进行描述, 底层会为此接口创建一个实现类, 在实现类的内部定义基于 `SqlSession` 对象的会话过程。然后还会将这个类的实例交给 spring 管理。

当我们掌握了基于SqlSession与数据库会话的基本过程后，又产生了一些思考，这个会话的步骤能否再次进行封装和简化，我们只关注业务和SQL语句的编写，至于框架如何与数据库通讯，我们不考虑，底层可以对其再次进行抽象。



3.9 MyBatis 中定义 SQL 映射的注解？

- 5) @Select
- 6) @Insert
- 7) @Delete
- 8) @Update

3.10 Mybatis 框架中提供的 ResultMap 元素的作用？

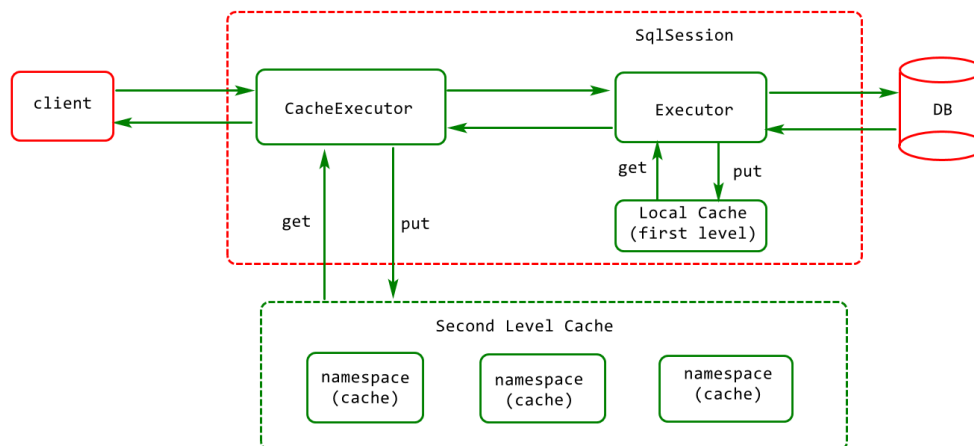
ResultMap 是 MyBatis 框架中用于实现高级映射的一个元素，它可以完成数据库表中字段名与 javaBean 对象中属性名不一致时的映射问题，同时提供了一对一，一对多，多对多的高级映射方式。

3.11 MyBatis 中有哪些动态 SQL 元素？

- 1) if
- 2) choose (when, otherwise)
- 3) trim (where, set)
- 4) foreach

3.12 MyBatis 的缓存架构是怎样的？

MyBatis 为了提高其查询的性能，提供了一些数据缓存特性，我们可以将这些缓存分为两大类，一级缓存和二级缓存，一级缓存可以理解为 SqlSession 级缓存，同一个 SqlSession 会话执行多次同样的 SQL 查询，查询结果可以从缓存获取。二级缓存默认是没有开启的，需要在映射文件或数据层对象中进行开启，可以多个 SqlSession 会话共享同一个缓存。



3.13 MyBatis 中的插件及实现原理是怎样的？

插件是 Mybatis 框架中的一个最重要的功能，能够基于此功能实现特定组件的特定方法进行增强。例如对查询操作进行分页设计，类似功能可以在 mybatis 中基于 **Interceptor**（拦截器）接口进行实现。其原理并不复杂就在在创建相关组件对象时为其创建代理对象，然后通过代理对象为其目标业务做功能增强。

3.14 MyBatis 框架中有哪些设计模式？

建造模式、工厂模式、单例模式、策略模式、代理模式、装饰模式、享元模式、门面模式、适配器模式、。。。。

4 总结 (Summary)

本篇文章重点对企业中常用的 Spring, SpringBoot, MyBatis 等框架常见问题进行了分析和答疑。实际记忆需要理论结合实践，逐步进行落地实现。