

5.1 Spring面试题总结

作者：Guide哥。

介绍：Github 70k Star 项目 [JavaGuide](#)（公众号同名）作者。每周都会在公众号更新一些自己原创干货。公众号后台回复“1”领取Java工程师必备学习资料+面试突击pdf。

这篇文章主要是想通过一些问题，加深大家对于 Spring 的理解，所以不会涉及太多的代码！这篇文章整理了挺长时间，下面的很多问题我自己在使用 Spring 的过程中也并没有注意，自己也是临时查阅了很多资料和书籍补上的。网上也有一些很多关于 Spring 常见问题/面试题整理的文章，我感觉大部分都是互相 copy，而且很多问题也不是很好，有些回答也存在问题。所以，自己花了一周的业余时间整理了一下，希望对大家有帮助。

5.1.1. 什么是 Spring 框架？

Spring 是一种轻量级开发框架，旨在提高开发人员的开发效率以及系统的可维护性。Spring 官网：<https://spring.io/>。

我们一般说 Spring 框架指的都是 Spring Framework，它是很多模块的集合，使用这些模块可以很方便地协助我们进行开发。这些模块是：核心容器、数据访问/集成、Web、AOP（面向切面编程）、工具、消息和测试模块。比如：Core Container 中的 Core 组件是 Spring 所有组件的核心，Beans 组件和 Context 组件是实现 IOC 和 依赖注入的基础，AOP 组件用来实现面向切面编程。

Spring 官网列出的 Spring 的 6 个特征：

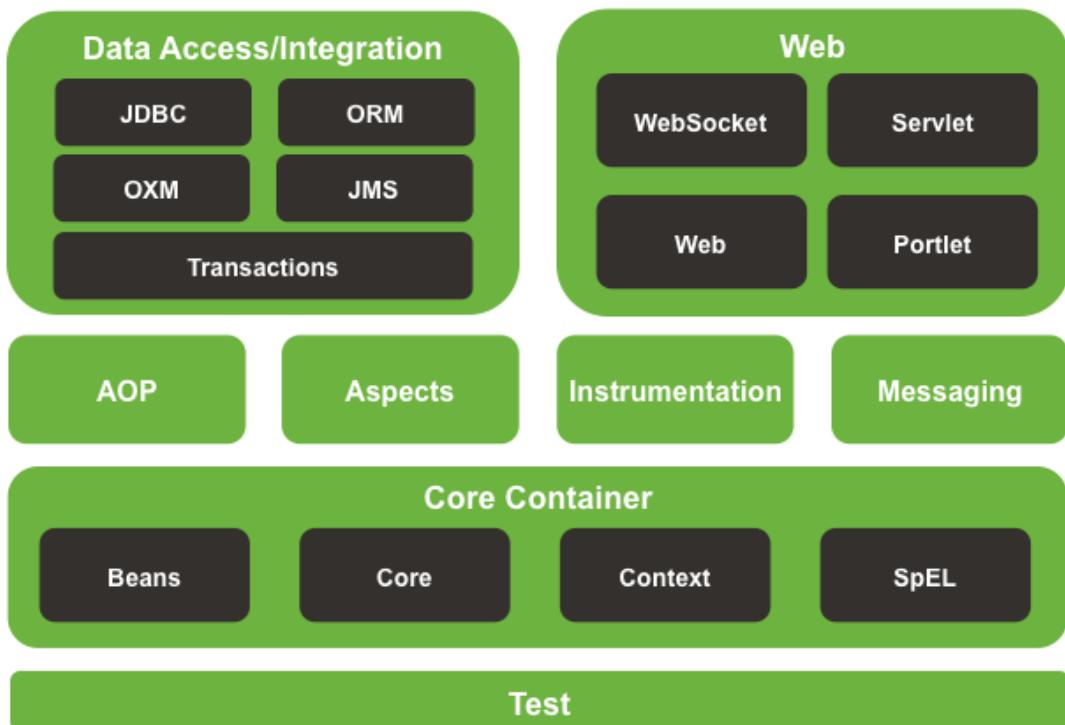
- **核心技术**：依赖注入(DI), AOP, 事件(events), 资源, i18n, 验证, 数据绑定, 类型转换, SpEL。
- **测试**：模拟对象, TestContext 框架, Spring MVC 测试, WebTestClient。
- **数据访问**：事务, DAO 支持, JDBC, ORM, 编组 XML。
- **Web 支持**：Spring MVC 和 Spring WebFlux Web 框架。
- **集成**：远程处理, JMS, JCA, JMX, 电子邮件, 任务, 调度, 缓存。
- **语言**：Kotlin, Groovy, 动态语言。

5.1.2 列举一些重要的 Spring 模块？

下图对应的是 Spring4.x 版本。目前最新的5.x版本中 Web 模块的 Portlet 组件已经被废弃掉，同时增加了用于异步响应式处理的 WebFlux 组件。



Spring Framework Runtime

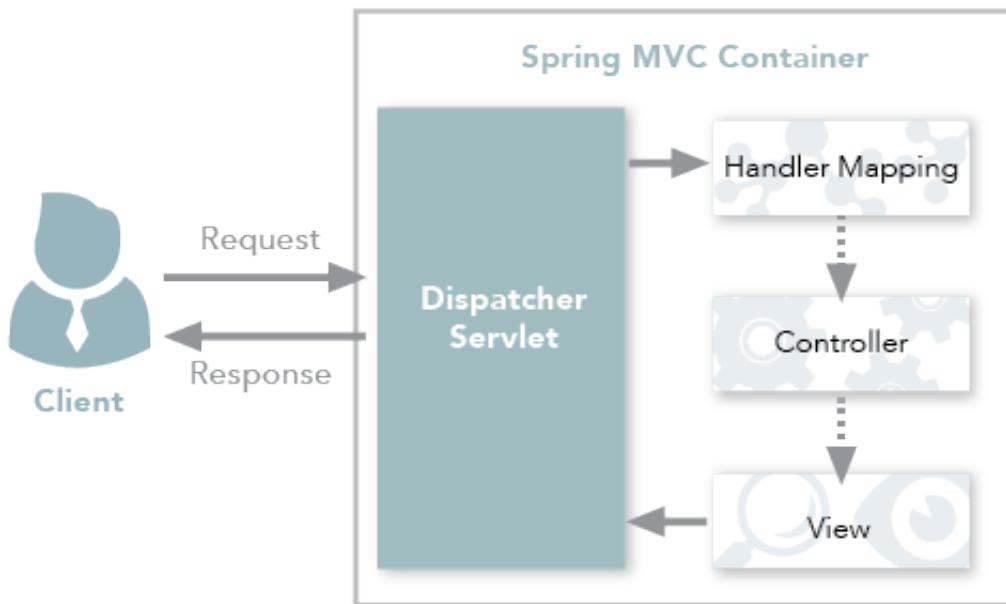


- **Spring Core**: 基础,可以说 Spring 其他所有的功能都需要依赖于该类库。主要提供 IoC 依赖注入功能。
- **Spring Aspects** : 该模块为与AspectJ的集成提供支持。
- **Spring AOP** : 提供了面向切面的编程实现。
- **Spring JDBC** : Java数据库连接。
- **Spring JMS** : Java消息服务。
- **Spring ORM** : 用于支持Hibernate等ORM工具。
- **Spring Web** : 为创建Web应用程序提供支持。
- **Spring Test** : 提供了对 JUnit 和 TestNG 测试的支持。

5.1.3 @RestController vs @Controller

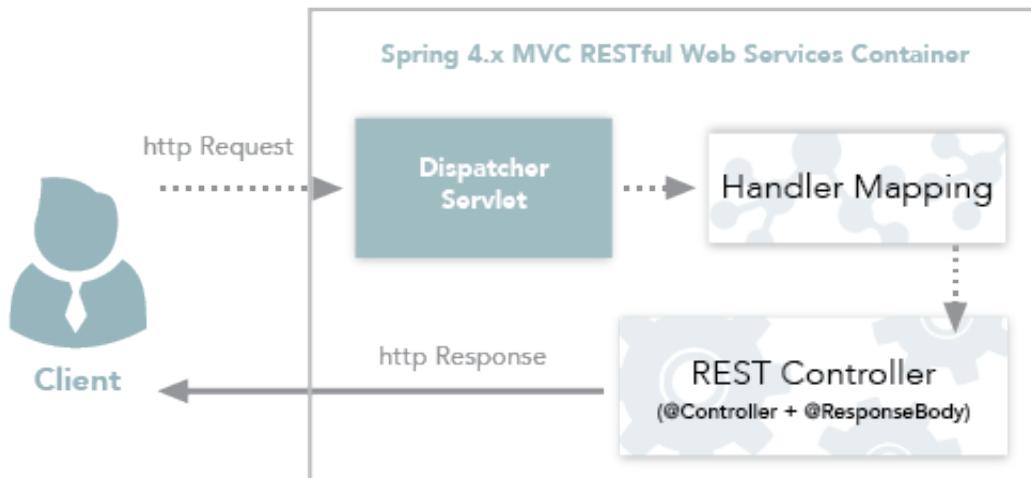
Controller 返回一个页面

单独使用 `@Controller` 不加 `@ResponseBody` 的话一般使用在要返回一个视图的情况，这种情况属于比较传统的Spring MVC 的应用，对应于前后端不分离的情况。



@RestController 返回JSON 或 XML 形式数据

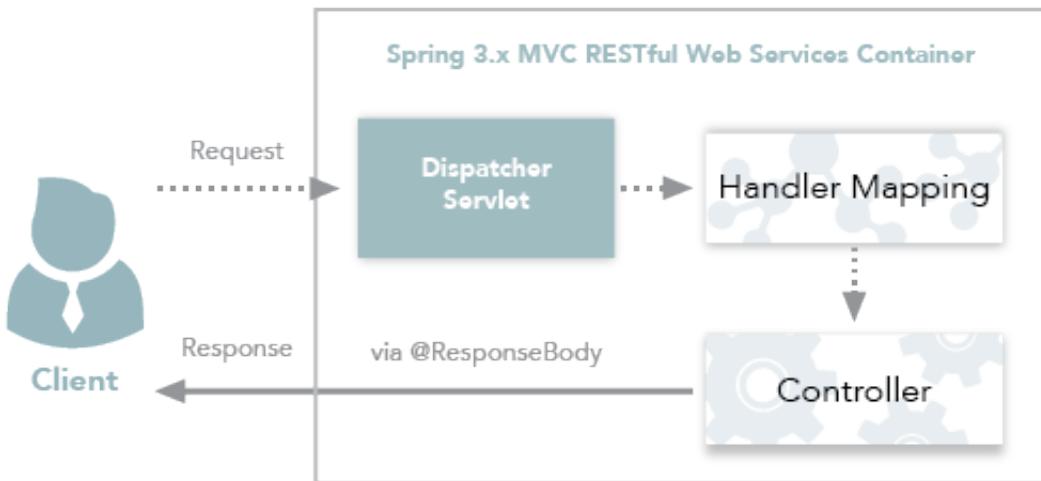
但 `@RestController` 只返回对象，对象数据直接以 JSON 或 XML 形式写入 HTTP 响应 (Response) 中，这种情况属于 RESTful Web 服务，这也是目前日常开发所接触的最常用的情况（前后端分离）。



@Controller +@ResponseBody 返回JSON 或 XML 形式数据

如果你需要在 Spring 4 之前开发 RESTful Web 服务的话，你需要使用 `@Controller` 并结合 `@ResponseBody` 注解，也就是说 `@Controller + @ResponseBody = @RestController` (Spring 4 之后新加的注解)。

@ResponseBody 注解的作用是将 Controller 的方法返回的对象通过适当的转换器转换为指定的格式之后，写入到HTTP响应(Response)对象的 body 中，通常用来返回 JSON 或者 XML 数据，返回 JSON 数据的情况比较多。



Reference:

- <https://dzone.com/articles/spring-framework-restcontroller-vs-controller> (图片来源)
- <https://javarevisited.blogspot.com/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html?m=1>

5.1.4 Spring IOC & AOP

谈谈自己对于 Spring IoC 和 AOP 的理解

IoC

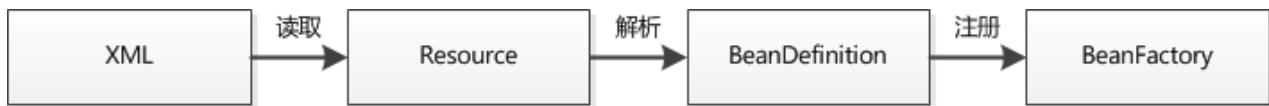
IoC (Inverse of Control:控制反转) 是一种设计思想，就是 将原本在程序中手动创建对象的控制权，交由Spring框架来管理。 IoC 在其他语言中也有应用，并非 Spring 特有。 IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个Map (key, value) ,Map 中存放的是各种对象。

将对象之间的相互依赖关系交给 IoC 容器来管理，并由 IoC 容器完成对象的注入。这样可以很大程度上简化应用的开发，把应用从复杂的依赖关系中解放出来。 IoC 容器就像是一个工厂一样，当我们需要创建一个对象的时候，只需要配置好配置文件/注解即可，完全不用考虑对象是如何被创建出来的。在实际项目中一个 Service 类可能有几百甚至上千个类作为它的底层，假如我们需要实例化这个 Service，你可能要每次都要搞清这个 Service 所有底层类的构造函数，这可能会把人逼疯。如果利用 IoC 的话，你只需要配置好，然后在需要的地方引用就行了，这大大增加了项目的可维护性且降低了开发难度。

Spring 时代我们一般通过 XML 文件来配置 Bean，后来开发人员觉得 XML 文件来配置不太好，于是 SpringBoot 注解配置就慢慢开始流行起来。

推荐阅读：<https://www.zhihu.com/question/23277575/answer/169698662>

Spring IoC的初始化过程：



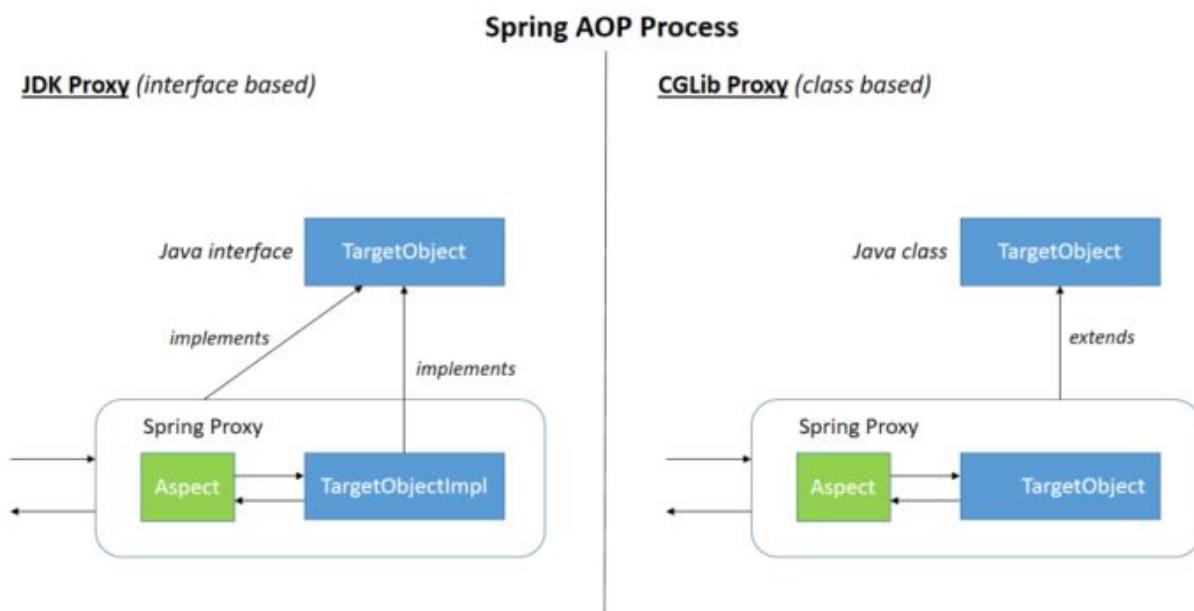
IoC源码阅读

- <https://javadoop.com/post/spring-ioc>

AOP

AOP(Aspect-Oriented Programming:面向切面编程)能够将那些与业务无关，却为业务模块所共同调用的逻辑或责任（例如事务处理、日志管理、权限控制等）封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可拓展性和可维护性。

Spring AOP就是基于动态代理的，如果要代理的对象，实现了某个接口，那么Spring AOP会使用JDK Proxy，去创建代理对象，而对于没有实现接口的对象，就无法使用 JDK Proxy 去进行代理了，这时候Spring AOP会使用Cglib，这时候Spring AOP会使用 **Cglib** 生成一个被代理对象的子类来作为代理，如下图所示：



当然你也可以使用 AspectJ ,Spring AOP 已经集成了AspectJ ， AspectJ 应该算的上是 Java 生态系统中最完整的 AOP 框架了。

使用 AOP 之后我们可以把一些通用功能抽象出来，在需要用到的地方直接使用即可，这样大大简化了代码量。我们需要增加新功能时也方便，这样也提高了系统扩展性。日志功能、事务管理等等场景都用到了 AOP。

Spring AOP 和 AspectJ AOP 有什么区别？

Spring AOP 属于运行时增强，而 AspectJ 是编译时增强。Spring AOP 基于代理(Proxying)，而 AspectJ 基于字节码操作(Bytecode Manipulation)。

Spring AOP 已经集成了 AspectJ，AspectJ 应该算是 Java 生态系统中最完整的 AOP 框架了。AspectJ 相比于 Spring AOP 功能更加强大，但是 Spring AOP 相对来说更简单，

如果我们的切面比较少，那么两者性能差异不大。但是，当切面太多的话，最好选择 AspectJ，它比 Spring AOP 快很多。

作者：Guide哥。

介绍：Github 70k Star 项目 [JavaGuide](#)（公众号同名）作者。每周都会在公众号更新一些自己原创干货。公众号后台回复“1”领取Java工程师必备学习资料+面试突击pdf。

5.1.5 Spring bean

Spring 中的 bean 的作用域有哪些？

- singleton：唯一 bean 实例，Spring 中的 bean 默认都是单例的。
- prototype：每次请求都会创建一个新的 bean 实例。
- request：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有效。
- session：每一次HTTP请求都会产生一个新的 bean，该bean仅在当前 HTTP session 内有效。
- global-session：全局session作用域，仅仅在基于portlet的web应用中才有意义，Spring5已经没有了。Portlet是能够生成语义代码(例如：HTML)片段的小型Java Web插件。它们基于portlet容器，可以像servlet一样处理HTTP请求。但是，与 servlet 不同，每个 portlet 都有不同的会话

Spring 中的单例 bean 的线程安全问题了解吗？

大部分时候我们并没有在系统中使用多线程，所以很少有人会关注这个问题。单例 bean 存在线程问题，主要是因为当多个线程操作同一个对象的时候，对这个对象的非静态成员变量的写操作会存在线程安全问题。

常见的有两种解决办法：

1. 在Bean对象中尽量避免定义可变的成员变量（不太现实）。
2. 在类中定义一个ThreadLocal成员变量，将需要的可变成员变量保存在 ThreadLocal 中（推荐的一种方式）。

@Component 和 @Bean 的区别是什么？

1. 作用对象不同：@Component 注解作用于类，而 @Bean 注解作用于方法。
2. @Component 通常是通过类路径扫描来自动侦测以及自动装配到Spring容器中（我们可以使用 @ComponentScan 注解定义要扫描的路径从中找出标识了需要装配的类自动装配到 Spring 的 bean 容器中）。@Bean 注解通常是在标有该注解的方法中定义产生这个 bean，@Bean 告诉了Spring这是某个类的示例，当我需要用它的时候还给我。
3. @Bean 注解比 Component 注解的自定义性更强，而且很多地方我们只能通过 @Bean 注解来注册bean。比如当我们引用第三方库中的类需要装配到 Spring 容器时，则只能通过 @Bean 来实现。

@Bean 注解使用示例：

```
@Configuration  
public class AppConfig {  
    @Bean  
    public TransferService transferService() {  
        return new TransferServiceImpl();  
    }  
}
```

上面的代码相当于下面的 xml 配置

```
<beans>  
    <bean id="transferService" class="com.acme.TransferServiceImpl"/>  
</beans>
```

下面这个例子是通过 @Component 无法实现的。

```

@Bean
public OneService getService(status) {
    case (status) {
        when 1:
            return new serviceImpl1();
        when 2:
            return new serviceImpl2();
        when 3:
            return new serviceImpl3();
    }
}

```

将一个类声明为Spring的 bean 的注解有哪些？

我们一般使用 `@Autowired` 注解自动装配 bean，要想把类标识成可用于 `@Autowired` 注解自动装配的 bean 的类，采用以下注解可实现：

- `@Component`：通用的注解，可标注任意类为 Spring 组件。如果一个 Bean 不知道属于哪个层，可以使用 `@Component` 注解标注。
- `@Repository`：对应持久层即 Dao 层，主要用于数据库相关操作。
- `@Service`：对应服务层，主要涉及一些复杂的逻辑，需要用到 Dao 层。
- `@Controller`：对应 Spring MVC 控制层，主要用户接受用户请求并调用 Service 层返回数据给前端页面。

Spring 中的 bean 生命周期？

这部分网上有很多文章都讲到了，下面的内容整理自：<https://yemengying.com/2016/07/14/spring-bean-life-cycle/>，除了这篇文章，再推荐一篇很不错的文章：<https://www.cnblogs.com/zrtqsk/p/3735273.html>。

- Bean 容器找到配置文件中 Spring Bean 的定义。
- Bean 容器利用 Java Reflection API 创建一个 Bean 的实例。
- 如果涉及到一些属性值 利用 `set()` 方法设置一些属性值。
- 如果 Bean 实现了 `BeanNameAware` 接口，调用 `setBeanName()` 方法，传入 Bean 的名字。
- 如果 Bean 实现了 `BeanClassLoaderAware` 接口，调用 `setBeanClassLoader()` 方法，传入 ClassLoader 对象的实例。
- 与上面的类似，如果实现了其他 `*.Aware` 接口，就调用相应的方法。
- 如果有和加载这个 Bean 的 Spring 容器相关的 `BeanPostProcessor` 对象，执行 `postProcessBeforeInitialization()` 方法
- 如果 Bean 实现了 `InitializingBean` 接口，执行 `afterPropertiesSet()` 方法。
- 如果 Bean 在配置文件中的定义包含 `init-method` 属性，执行指定的方法。

- 如果有和加载这个 Bean 的 Spring 容器相关的 BeanPostProcessor 对象，执行 postProcessAfterInitialization() 方法
- 当要销毁 Bean 的时候，如果 Bean 实现了 DisposableBean 接口，执行 destroy() 方法。
- 当要销毁 Bean 的时候，如果 Bean 在配置文件中的定义包含 destroy-method 属性，执行指定的方法。

图示：

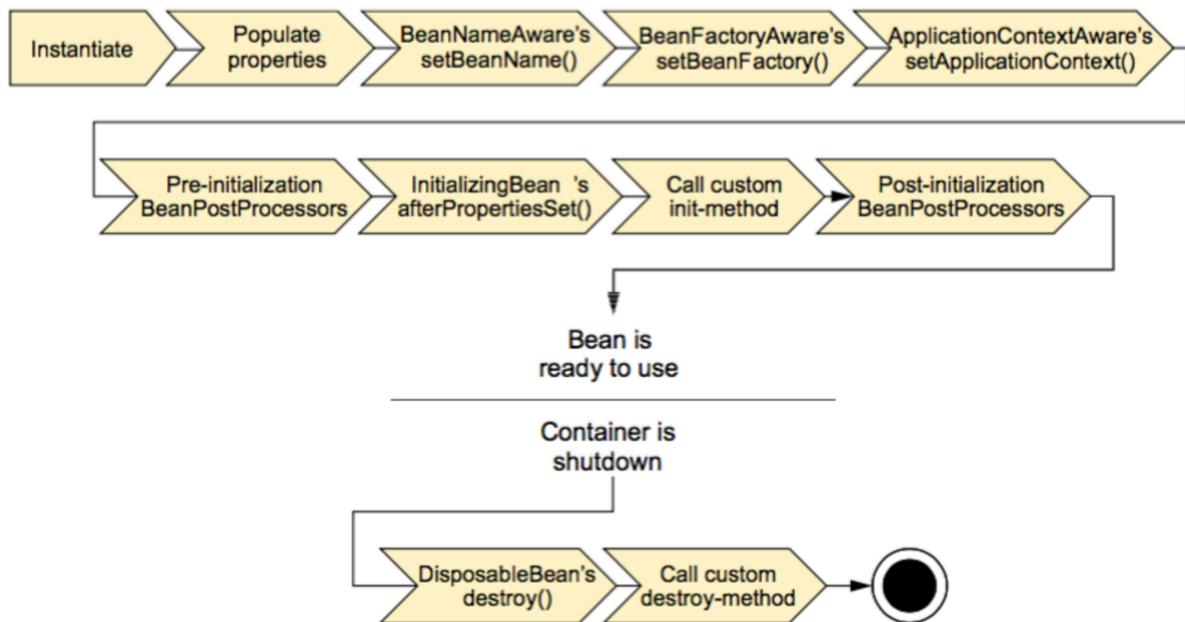
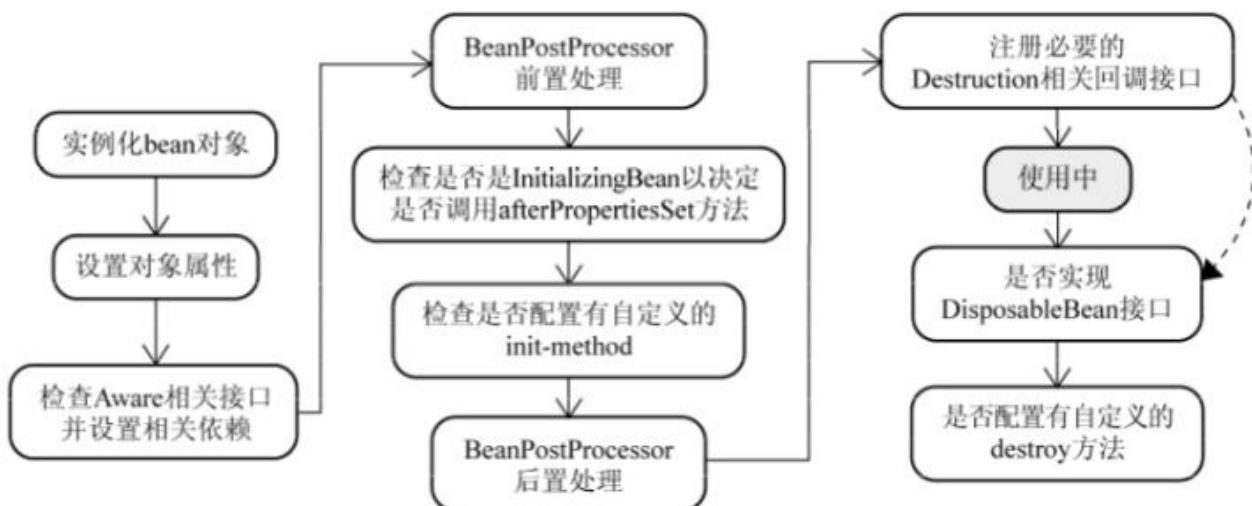


Figure 1.5 A bean goes through several steps between creation and destruction in the Spring container. Each step is an opportunity to customize how the bean is managed in Spring.

与之比较类似的中文版本：



5.1.6 Spring MVC

说说自己对于 Spring MVC 了解？

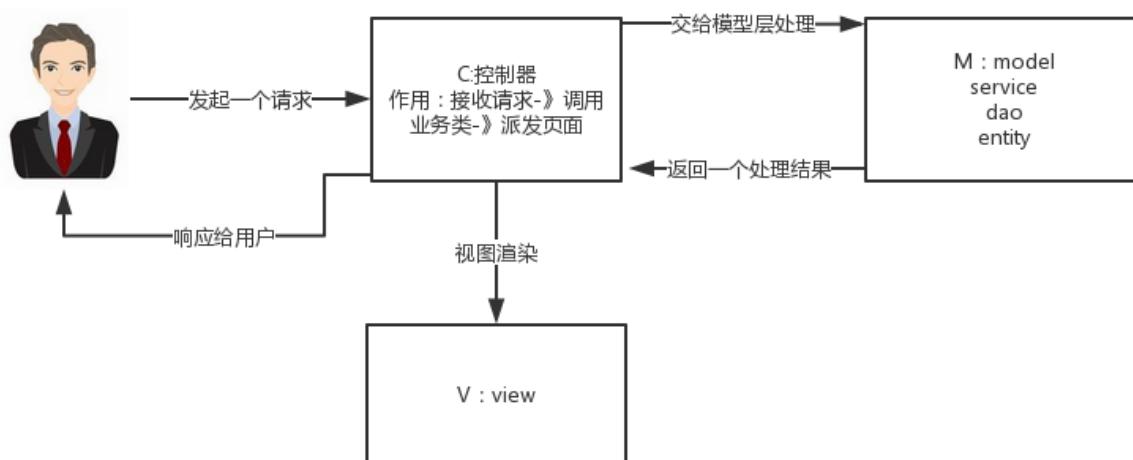
谈到这个问题，我们不得不提之前 Model1 和 Model2 这两个没有 Spring MVC 的时代。

- **Model1 时代**：很多学 Java 后端比较晚的朋友可能并没有接触过 Model1 模式下的 JavaWeb 应用开发。在 Model1 模式下，整个 Web 应用几乎全部用 JSP 页面组成，只用少量的 JavaBean 来处理数据库连接、访问等操作。这个模式下 JSP 即是控制层又是表现层。显而易见，这种模式存在很多问题。比如①将控制逻辑和表现逻辑混杂在一起，导致代码重用率极低；②前端和后端相互依赖，难以进行测试并且开发效率极低；
- **Model2 时代**：学过 Servlet 并做过相关 Demo 的朋友应该了解“Java Bean(Model)+JSP (View,) +Servlet (Controller) ”这种开发模式，这就是早期的 JavaWeb MVC 开发模式。Model:系统涉及的数据，也就是 dao 和 bean。View：展示模型中的数据，只是用来展示。Controller：处理用户请求都发送给，返回数据给 JSP 并展示给用户。

Model2 模式下还存在很多问题，Model2 的抽象和封装程度还远远不够，使用 Model2 进行开发时不可避免地会重复造轮子，这就大大降低了程序的可维护性和复用性。于是很多 JavaWeb 开发相关的 MVC 框架应运而生比如 Struts2，但是 Struts2 比较笨重。随着 Spring 轻量级开发框架的流行，Spring 生态圈出现了 Spring MVC 框架，Spring MVC 是当前最优秀的 MVC 框架。相比于 Struts2，Spring MVC 使用更加简单和方便，开发效率更高，并且 Spring MVC 运行速度更快。

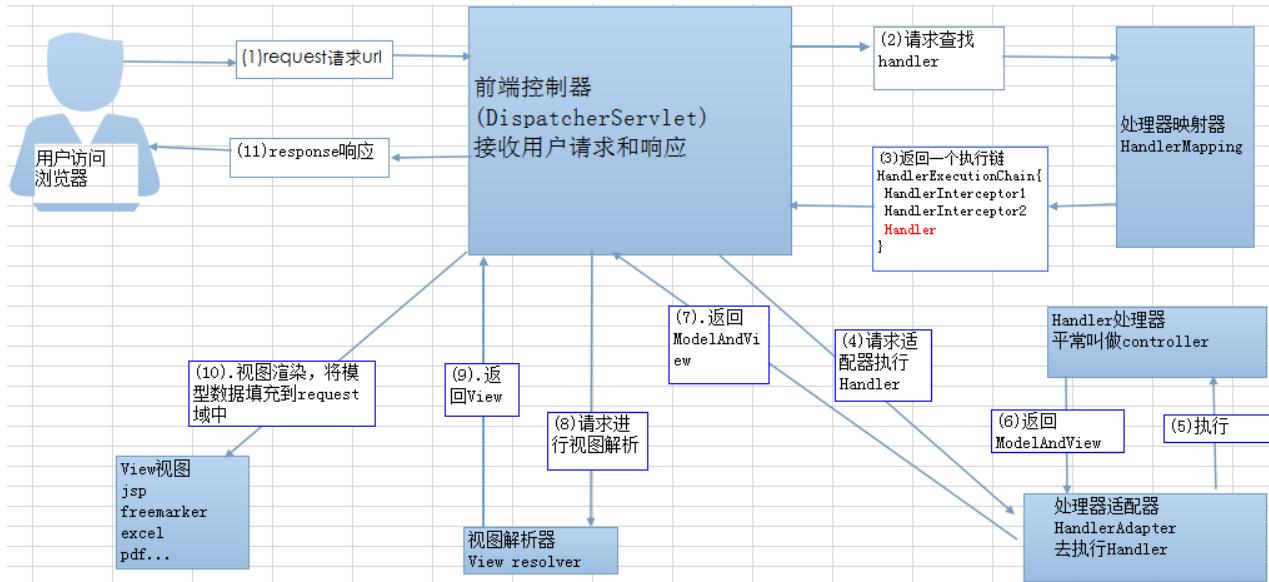
MVC 是一种设计模式，Spring MVC 是一款很优秀的 MVC 框架。Spring MVC 可以帮助我们进行更简洁的 Web 层的开发，并且它天生与 Spring 框架集成。Spring MVC 下我们一般把后端项目分为 Service 层（处理业务）、Dao 层（数据库操作）、Entity 层（实体类）、Controller 层（控制层，返回数据给前台页面）。

Spring MVC 的简单原理图如下：



SpringMVC 工作原理了解吗？

原理如下图所示：



上图的一个笔误的小问题：Spring MVC 的入口函数也就是前端控制器 DispatcherServlet 的作用是接收请求，响应结果。

流程说明（重要）：

1. 客户端（浏览器）发送请求，直接请求到 DispatcherServlet。
2. DispatcherServlet 根据请求信息调用 HandlerMapping，解析请求对应的 Handler。
3. 解析到对应的 Handler（也就是我们平常说的 Controller 控制器）后，开始由 HandlerAdapter 适配器处理。
4. HandlerAdapter 会根据 Handler 来调用真正的处理器来处理请求，并处理相应的业务逻辑。
5. 处理器处理完业务后，会返回一个 ModelAndView 对象，Model 是返回的数据对象，View 是个逻辑上的 View。
6. ViewResolver 会根据逻辑 View 查找实际的 View。
7. DispatcherServlet 把返回的 Model 传给 View（视图渲染）。
8. 把 View 返回给请求者（浏览器）

5.1.7 Spring 框架中用到了哪些设计模式？

关于下面一些设计模式的详细介绍，可以看笔者前段时间的原创文章《[面试官：“谈谈Spring中都用到了那些设计模式？”](#)》。

- **工厂设计模式**：Spring 使用工厂模式通过 BeanFactory、ApplicationContext 创建 bean 对象。
- **代理设计模式**：Spring AOP 功能的实现。
- **单例设计模式**：Spring 中的 Bean 默认都是单例的。

- **包装器设计模式**：我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。
- **观察者模式**: Spring 事件驱动模型就是观察者模式很经典的一个应用。
- **适配器模式** :Spring AOP 的增强或通知(Advice)使用到了适配器模式、spring MVC 中也是用到了适配器模式适配 Controller 。
-

5.1.8 Spring 事务

Spring 管理事务的方式有几种？

1. 编程式事务，在代码中硬编码。(不推荐使用)
2. 声明式事务，在配置文件中配置（推荐使用）

声明式事务又分为两种：

1. 基于XML的声明式事务
2. 基于注解的声明式事务

Spring 事务中的隔离级别有哪几种？

TransactionDefinition 接口中定义了五个表示隔离级别的常量：

- **TransactionDefinition.ISOLATION_DEFAULT**: 使用后端数据库默认的隔离级别，Mysql 默认采用的 REPEATABLE_READ 隔离级别 Oracle 默认采用的 READ_COMMITTED 隔离级别。
- **TransactionDefinition.ISOLATION_READ_UNCOMMITTED**: 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读
- **TransactionDefinition.ISOLATION_READ_COMMITTED**: 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生
- **TransactionDefinition.ISOLATION_REPEATABLE_READ**: 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。
- **TransactionDefinition.ISOLATION_SERIALIZABLE**: 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

Spring 事务中哪几种事务传播行为？

支持当前事务的情况：

- **TransactionDefinition.PROPAGATION_REQUIRED**: 如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。
- **TransactionDefinition.PROPAGATION_SUPPORTS**: 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。
- **TransactionDefinition.PROPAGATION_MANDATORY**: 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。（mandatory：强制性）

不支持当前事务的情况：

- **TransactionDefinition.PROPAGATIONQUIRES_NEW**: 创建一个新的事务，如果当前存在事务，则把当前事务挂起。
- **TransactionDefinition.PROPAGATION_NOT_SUPPORTED**: 以非事务方式运行，如果当前存在事务，则把当前事务挂起。
- **TransactionDefinition.PROPAGATION_NEVER**: 以非事务方式运行，如果当前存在事务，则抛出异常。

其他情况：

- **TransactionDefinition.PROPAGATION_NESTED**: 如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于 TransactionDefinition.PROPAGATION_REQUIRED。

@Transactional(rollbackFor = Exception.class)注解了解吗？

我们知道：Exception分为运行时异常RuntimeException和非运行时异常。事务管理对于企业应用来说是至关重要的，即使出现异常情况，它也可以保证数据的一致性。

当 @Transactional 注解作用于类上时，该类的所有 public 方法将都具有该类型的事务属性，同时，我们也可以在方法级别使用该标注来覆盖类级别的定义。如果类或者方法加了这个注解，那么这个类里面的方法抛出异常，就会回滚，数据库里面的数据也会回滚。

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

关于 @Transactional 注解推荐阅读的文章：

- [透彻的掌握 Spring 中@Transactional 的使用](#)

5.1.9 JPA

如何使用JPA在数据库中非持久化一个字段？

假如我们有下面一个类：

```
Entity(name="USER")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "ID")
    private Long id;

    @Column(name="USER_NAME")
    private String userName;

    @Column(name="PASSWORD")
    private String password;

    private String secrect;

}
```

如果我们想让 `secrect` 这个字段不被持久化，也就是不被数据库存储怎么办？我们可以采用下面几种方法：

```
static String transient1; // not persistent because of static
final String transient2 = "Satish"; // not persistent because of final
transient String transient3; // not persistent because of transient
@Transient
String transient4; // not persistent because of @Transient
```

一般使用后面两种方式比较多，我个人使用注解的方式比较多。

参考

- 《Spring 技术内幕》
- <http://www.cnblogs.com/wmystsxz/p/8820371.html>
- <https://www.journaldev.com/2696/spring-interview-questions-and-answers>
- <https://www.edureka.co/blog/interview-questions/spring-interview-questions/>
- <https://www.cnblogs.com/clwydjgs/p/9317849.html>
- <https://howtodoinjava.com/interview-questions/top-spring-interview-questions-with-answers/>
- <http://www.tomaszezula.com/2014/02/09/spring-series-part-5-component-vs-bean/>
- <https://stackoverflow.com/questions/34172888/difference-between-bean-and-autowired>

公众号

如果大家想要实时关注我更新的文章以及分享的干货的话，可以关注我的公众号。

《Java面试突击》：由本文档衍生的专为面试而生的《Java面试突击》V2.0 PDF 版本[公众号后台](#)回复 "Java面试突击" 即可免费领取！

Java工程师必备学习资源：一些Java工程师常用学习资源[公众号后台](#)回复关键字“1”即可免费无套路获取。



5.2 MyBatis面试题总结

本篇文章是JavaGuide收集自网络，原出处不明。

Mybatis 技术内幕系列博客，从原理和源码角度，介绍了其内部实现细节，无论是写的好与不好，我确实是用心写了，由于并不是介绍如何使用 Mybatis 的文章，所以，一些参数使用细节略掉了，我们的目标是介绍 Mybatis 的技术架构和重要组成部分，以及基本运行原理。

博客写的很辛苦，但是写出来却不一定好看，所谓开始很兴奋，过程很痛苦，结束很遗憾。要求不高，只要读者能从系列博客中，学习到一点其他博客所没有的技术点，作为作者，我就很欣慰了，我也读别人写的博客，通常对自己当前研究的技术，是很有帮助的。

尽管还有很多可写的内容，但是，我认为再写下去已经没有意义，任何其他小的功能点，都是在已经介绍的基本框架和基本原理下运行的，只有结束，才能有新的开始。写博客也积攒了一些经验，源码多了感觉就是复制黏贴，源码少了又觉得是空谈原理，将来再写博客，我希望是“精炼博文”，好读好懂美观读起来又不累，希望自己能再写一部开源分布式框架原理系列博客。

有胆就来，我出几道 Mybatis 面试题，看你能回答上来几道（都是我出的，可不是网上找的）。

5.2.1 #{} 和 \${} 的区别是什么？

注：这道题是面试官面试我同事的。

答：

- \${} 是 Properties 文件中的变量占位符，它可用于标签属性值和 sql 内部，属于静态文本替换，比如 \${driver} 会被静态替换为 com.mysql.jdbc.Driver 。
- #{} 是 sql 的参数占位符，Mybatis 会将 sql 中的 #{} 替换为?号，在 sql 执行前会使用 PreparedStatement 的参数设置方法，按序给 sql 的?号占位符设置参数值，比如 ps.setInt(0, parameterValue), #{{item.name}} 的取值方式为使用反射从参数对象中获取 item 对象的 name 属性值，相当于 param.getItem().getName() 。

5.2.2 Xml 映射文件中，除了常见的 select insert update delete 标签之外，还有哪些标签？

注：这道题是京东面试官面试我问的。

答：还有很多其他的标

签， <resultMap> 、 <parameterMap> 、 <sql> 、 <include> 、 <selectKey> ，加上动态 sql 的 9 个标签， trim where set foreach if choose when otherwise bind 等，其中为 sql 片段标签，通过 <include> 标签引入 sql 片段， <selectKey> 为不支持自增的主键生成策略标签。

5.2.3 最佳实践中，通常一个 Xml 映射文件，都会写一个 Dao 接口与之对应，请问，这个 Dao 接口的工作原理是什么？Dao 接口里的方法，参数不同时，方法能重载吗？

注：这道题也是京东面试官面试我时问的。

答：Dao 接口，就是人们常说的 Mapper 接口，接口的全限名，就是映射文件中的 namespace 的值，接口的方法名，就是映射文件中 MappedStatement 的 id 值，接口方法内的参数，就是传递给 sql 的参数。Mapper 接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为 key 值，可唯一定位一个 MappedStatement，举

例： com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到 namespace 为 com.mybatis3.mappers.StudentDao 下面 id = findStudentById 的 MappedStatement。在 Mybatis 中，每一个 <select>、<insert>、<update>、<delete> 标签，都会被解析为一个 MappedStatement 对象。

Dao 接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK 动态代理为 Dao 接口生成代理 proxy 对象，代理对象 proxy 会拦截接口方法，转而执行 MappedStatement 所代表的 sql，然后将 sql 执行结果返回。

5.2.4 Mybatis 是如何进行分页的？分页插件的原理是什么？

注：我出的。

答：Mybatis 使用 RowBounds 对象进行分页，它是针对 ResultSet 结果集执行的内存分页，而非物理分页，可以在 sql 内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql，根据 dialect 方言，添加对应的物理分页语句和物理分页参数。

举例： select _ from student，拦截 sql 后重写为： select t._ from (select * from student) t limit 0, 10

5.2.5 简述 Mybatis 的插件运行原理，以及如何编写一个插件。

注：我出的。

答：Mybatis 仅可以编写针对

ParameterHandler、ResultSetHandler、StatementHandler、Executor 这 4 种接口的插件，

Mybatis 使用 JDK 的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这 4 种接口对象的方法时，就会进入拦截方法，具体就是 InvocationHandler 的 invoke() 方

法，当然，只会拦截那些你指定需要拦截的方法。

实现 Mybatis 的 Interceptor 接口并复写 intercept() 方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

5.2.6 Mybatis 执行批量插入，能返回数据库主键列表吗？

注：我出的。

答：能， JDBC 都能， Mybatis 当然也能。

5.2.7 Mybatis 动态 sql 是做什么的？都有哪些动态 sql？能简述一下动态 sql 的执行原理不？

注：我出的。

答：Mybatis 动态 sql 可以让我们在 Xml 映射文件内，以标签的形式编写动态 sql，完成逻辑判断和动态拼接 sql 的功能，Mybatis 提供了 9 种动态 sql 标签

trim|where|set|foreach|if|choose|when|otherwise|bind 。

其执行原理为，使用 OGNL 从 sql 参数对象中计算表达式的值，根据表达式的值动态拼接 sql，以此来完成动态 sql 的功能。

5.2.8 Mybatis 是如何将 sql 执行结果封装为目标对象并返回的？都有哪些映射形式？

注：我出的。

答：第一种是使用 <resultMap> 标签，逐一定义列名和对象属性名之间的映射关系。第二种是使用 sql 列的别名功能，将列别名书写为对象属性名，比如 T_NAME AS NAME，对象属性名一般是 name，小写，但是列名不区分大小写，Mybatis 会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成 T_NAME AS NaMe，Mybatis 一样可以正常工作。

有了列名与属性名的映射关系后，Mybatis 通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

5.2.9 Mybatis 能执行一对一、一对多的关联查询吗？都有哪些实现方式，以及它们之间的区别。

注：我出的。

答：能，Mybatis 不仅可以执行一对一、一对多的关联查询，还可以执行多对一，多对多的关联查询，多对一查询，其实就是一对一查询，只需要把 `selectOne()` 修改为 `selectList()` 即可；多对多查询，其实就是一对多查询，只需要把 `selectOne()` 修改为 `selectList()` 即可。

关联对象查询，有两种实现方式，一种是单独发送一个 sql 去查询关联对象，赋给主对象，然后返回主对象。另一种是使用嵌套查询，嵌套查询的含义为使用 `join` 查询，一部分列是 A 对象的属性值，另外一部分列是关联对象 B 的属性值，好处是只发一个 sql 查询，就可以把主对象和其关联对象查出来。

那么问题来了，`join` 查询出来 100 条记录，如何确定主对象是 5 个，而不是 100 个？其去重复的原理是 `<resultMap>` 标签内的 `<id>` 子标签，指定了唯一确定一条记录的 id 列，Mybatis 根据列值来完成 100 条记录的去重复功能，`<id>` 可以有多个，代表了联合主键的语意。

同样主对象的关联对象，也是根据这个原理去重复的，尽管一般情况下，只有主对象会有重复记录，关联对象一般不会重复。

举例：下面 `join` 查询出来 6 条记录，一、二列是 `Teacher` 对象列，第三列为 `Student` 对象列，Mybatis 去重复处理后，结果为 1 个老师 6 个学生，而不是 6 个老师 6 个学生。

t_id	t_name	s_id
1	teacher	38
1	teacher	39
1	teacher	40
1	teacher	41
1	teacher	42
1	teacher	43

5.2.10 Mybatis 是否支持延迟加载？如果支持，它的实现原理是什么？

注：我出的。

答：Mybatis 仅支持 `association` 关联对象和 `collection` 关联集合对象的延迟加载，`association` 指的就是一对一，`collection` 指的就是一对多查询。在 Mybatis 配置文件中，可以配置是否启用延迟加载 `lazyLoadingEnabled=true/false`。

它的原理是，使用 CGLIB 创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 `a.getB().getName()`，拦截器 `invoke()` 方法发现 `a.getB()` 是 null 值，那么就会单独发送事先保存好的查询关联 B 对象的 sql，把 B 查询上来，然后调用 `a.setB(b)`，于是 a 的对象 b 属性就有值了，接着完成 `a.getB().getName()` 方法的调用。这就是延迟加载的基本原理。

当然了，不光是 Mybatis，几乎所有的包括 Hibernate，支持延迟加载的原理都是一样的。

5.2.11 Mybatis 的 Xml 映射文件中，不同的 Xml 映射文件，id 是否可以重复？

注：我出的。

答：不同的 Xml 映射文件，如果配置了 namespace，那么 id 可以重复；如果没有配置 namespace，那么 id 不能重复；毕竟 namespace 不是必须的，只是最佳实践而已。

原因就是 namespace+id 是作为 `Map<String, MappedStatement>` 的 key 使用的，如果没有 namespace，就剩下 id，那么，id 重复会导致数据互相覆盖。有了 namespace，自然 id 就可以重复，namespace 不同，namespace+id 自然也就不同。

5.2.12 Mybatis 中如何执行批处理？

注：我出的。

答：使用 BatchExecutor 完成批处理。

5.2.13 Mybatis 都有哪些 Executor 执行器？它们之间的区别是什么？

注：我出的

答：Mybatis 有三种基本的 Executor 执行器，`SimpleExecutor`、`ReuseExecutor`、`BatchExecutor`。

`SimpleExecutor`：每执行一次 update 或 select，就开启一个 Statement 对象，用完立刻关闭 Statement 对象。

`ReuseExecutor`：执行 update 或 select，以 sql 作为 key 查找 Statement 对象，存在就使用，不存在就创建，用完后，不关闭 Statement 对象，而是放置于 `Map<String, Statement>` 内，供下一次使用。简言之，就是重复使用 Statement 对象。

`BatchExecutor`：执行 `update`（没有 `select`, JDBC 批处理不支持 `select`），将所有 sql 都添加到批处理中 (`addBatch()`)，等待统一执行 (`executeBatch()`)，它缓存了多个 Statement 对象，每个 Statement 对象都是 `addBatch()` 完毕后，等待逐一执行 `executeBatch()` 批处理。与 JDBC 批处理相同。

作用范围：Executor 的这些特点，都严格限制在 `SqlSession` 生命周期范围内。

5.2.14 Mybatis 中如何指定使用哪一种 Executor 执行器？

注：我出的

答：在 Mybatis 配置文件中，可以指定默认的 `ExecutorType` 执行器类型，也可以手动给 `DefaultSqlSessionFactory` 的创建 `SqlSession` 的方法传递 `ExecutorType` 类型参数。

5.2.15 Mybatis 是否可以映射 Enum 枚举类？

注：我出的

答：Mybatis 可以映射枚举类，不单可以映射枚举类，Mybatis 可以映射任何对象到表的一列上。映射方式为自定义一个 `TypeHandler`，实现 `TypeHandler` 的 `setParameter()` 和 `getResultSet()` 接口方法。`TypeHandler` 有两个作用，一是完成从 `javaType` 至 `jdbcType` 的转换，二是完成 `jdbcType` 至 `javaType` 的转换，体现为 `setParameter()` 和 `getResultSet()` 两个方法，分别代表设置 sql 问号占位符参数和获取列查询结果。

5.2.16 Mybatis 映射文件中，如果 A 标签通过 `include` 引用了 B 标签的内容，请问，B 标签能否定义在 A 标签的后面，还是说必须定义在 A 标签的前面？

注：我出的

答：虽然 Mybatis 解析 Xml 映射文件是按照顺序解析的，但是，被引用的 B 标签依然可以定义在任何地方，Mybatis 都可以正确识别。

原理是，Mybatis 解析 A 标签，发现 A 标签引用了 B 标签，但是 B 标签尚未解析到，尚不存在，此时，Mybatis 会将 A 标签标记为未解析状态，然后继续解析余下的标签，包含 B 标签，待所有标签解析完毕，Mybatis 会重新解析那些被标记为未解析的标签，此时再解析 A 标签时，B 标签已经存在，A 标签也就可以正常解析完成了。

5.2.17 简述 Mybatis 的 Xml 映射文件和 Mybatis 内部数据结构之间的映射关系？

注：我出的

答：Mybatis 将所有 Xml 配置信息都封装到 All-In-One 重量级对象 Configuration 内部。在 Xml 映射文件中，`<parameterMap>` 标签会被解析为 ParameterMap 对象，其每个子元素会被解析为 ParameterMapping 对象。`<resultMap>` 标签会被解析为 ResultMap 对象，其每个子元素会被解析为 ResultMapping 对象。每一个 `<select>、<insert>、<update>、<delete>` 标签均会被解析为 MappedStatement 对象，标签内的 sql 会被解析为 BoundSql 对象。

5.2.18 为什么说 Mybatis 是半自动 ORM 映射工具？它与全自动的区别在哪里？

注：我出的

答：Hibernate 属于全自动 ORM 映射工具，使用 Hibernate 查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而 Mybatis 在查询关联对象或关联集合对象时，需要手动编写 sql 来完成，所以，称之为半自动 ORM 映射工具。

面试题看似都很简单，但是想要能正确回答上来，必定是研究过源码且深入的人，而不是仅会使用的人或者用的很熟的人，以上所有面试题及其答案所涉及的内容，在我的 Mybatis 系列博客中都有详细讲解和原理分析。-----

5.3 Kafka面试题总结

5.3.1 Kafka 是什么？主要应用场景有哪些？

Kafka 是一个分布式流式处理平台。这到底是什么意思呢？

流平台具有三个关键功能：

1. 消息队列：发布和订阅消息流，这个功能类似于消息队列，这也是 Kafka 也被归类为消息队列的原因。
2. 容错的持久方式存储记录消息流：Kafka 会把消息持久化到磁盘，有效避免了消息丢失的风险。
3. 流式处理平台：在消息发布的时候进行处理，Kafka 提供了一个完整的流式处理类库。

Kafka 主要有两大应用场景：