链滴

个人整理 - Java 后端面试题 - 框架篇

作者：valarchie

原文链接：https://ld246.com/article/1583688159439

来源网站：链滴

标 ★ 号为重要知识点

<h2 id="说一下IOC和AOP-">说一下 IOC 和 AOP？</h2>
IOC 是依赖反转，通过依赖注入 DI 实现 IOC。主要思想是面向抽象编程而不是面向具体实现编
。反转的意思是从依赖具体
实现变为依赖抽象。
AOP 是面向切面编程，通过代理对代码无侵入的添加功能，实现切面编程。比如事务、日志等
块。
代理机制主要有 jdk 提供的动态代理和 cglib 提供的字节码动态代码。
<h2 id="-Spring中Bean的生命周期-">★Spring 中 Bean 的生命周期。</h2>

首先这个 bean 在进行开始实例化的时候会先进行调用该类的构造函数，默认是单例的
然后去注入属性中的 bean
如果 bean 实现了 BeanNameAware 接口，spring 将 bean 的 id 传给 setBeanName()方法；<
li>
如果 bean 实现了 BeanFactoryAware 接口，spring 将调用 setBeanFactory 方法，将 BeanFac
ory 实例传进来；使 bean 获得访问 Spring 容器的能力。
如果 bean 实现了 ApplicationContextAware 接口，它的 setApplicationContext()方法将被调
，将应用上下文的引用传入到 bean 中；使得 bean 获取访问对应上下文的能力。
如果 bean 实现了 BeanPostProcessor 接口，它的 postProcessBeforeInitialization 方法将被
用；如 AutowiredAnnotationBeanPostProcessor
如果 bean 实现了 InitializingBean 接口，spring 将调用它的 afterPropertiesSet 接口方法，类
的如果 bean 使用了 init-method 属性声明了初始化方法，该方法也会被调用；
如果 bean 实现了 BeanPostProcessor 接口，它的 postProcessAfterInitialization 接口方法将
调用；
此时 bean 已经准备就绪，可以被应用程序使用了，他们将一直驻留在应用上下文中，直到该应
上下文被销毁；
若 bean 实现了 DisposableBean 接口，spring 将调用它的 distroy()接口方法。同样的，如果 b
an 使用了 destroy-method 属性声明了销毁方法，则该方法被调用；

<h2 id="Spring中注解Autowired和Resource的区别-">Spring 中注解 Autowired 和 Resource 的
别？</h2>
共同点:
两者都可以写在字段和 setter 方法上。两者如果都写在字段上，那么就不需要再写 setter 方法

不同点 :
（1）@Autowired 为 Spring 提供的注解，需要导入包 org.springframework.beans.factory.a
notation.Autowired;
只按照 byType 注入。 @Autowired 注解是按照类型（byType）装配依赖对象，默认情况下它要求
赖对象必须存在，如果允许 null 值，
可以设置它的 required 属性为 false。如果我们想使用按照名称（byName）来装配，可以结合 @Q
alifier 注解一起使用。
（2）@Resource 默认按照 ByName 自动注入，由 J2EE 提供，需要导入包 javax.annotation.R
source。@Resource 有两个重要的属性：
name 和 type，而 Spring 将 @Resource 注解的 name 属性解析为 bean 的名字，而 type 属性则
析为 bean 的类型。所以，如果使用 name 属性，
则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。如果既不制定
ame 也不制定 type 属性，这时将通过反射
机制使用 byName 自动注入策略。如果按 name 找不到对应的 bean 的话回退到 ByType 的自动注
策略。
<h2 id="-Controller和-RestController的区别-">@Controller 和 @RestController 的区别？</h2

@Controller

<pre><code class="highlight-chroma">方法的返回值。默认是视图页面的跳转路径。
如果想返回json对，必须在方法的上面加@ResponseBody
</code></pre>
@RestController
<pre><code class="highlight-chroma">方法返回值，默认是json对象，也就是相当于@Controller里面的方法上添加了@ResponseBody
如果方法返回值需跳转视图页面的话，那么方法的返回类型必须是View 或者ModelAndView.
</code></pre>
<h2 id="5-依赖注入的方式有哪几种-">5.依赖注入的方式有哪几种？</h2>

set 方法注入
构造器注入
注解注入

<h2 id="-Spring中IOC的原理-">★Spring 中 IOC 的原理？</h2>
第一个过程是 Resource 定位过程。这个 Resource 定位指的是 BeanDefinition 的资源定位，由 ResourceLoader 通过统一的 Resource 接口来完成，这个 Resource 对各种形式的 BeanDefiniti n 的使用都提供了统一接口。对于这些 BeanDefinition 的存在形式，相信大家都不会感到陌生。比，在文件系统中的 Bean 定义信息可以使用 FileSystemResource 来进行抽象；在类路径中的 Bean 义信息可以使用前面提到的 ClassPathResource 来使用，等等。这个定位过程类似于容器寻找数据过程，就像用水桶装水先要把水找到一样。
第二个过程是 BeanDefinition 的载入。这个载入过程是把用户定义好的 Bean 表示成 IoC 容器部的数据结构，而这个容器内部的数据结构就是 BeanDefinition。下面介绍这个数据结构的详细定。具体来说，这个 BeanDefinition 实际上就是 POJO 对象在 IoC 容器中的抽象，通过这个 BeanDefi ition 定义的数据结构，使 IoC 容器能够方便地对 POJO 对象也就是 Bean 进行管理。在下面的章节，我们会对这个载入的过程进行详细的分析，使大家对整个过程有比较清楚的了解。
第三个过程是向 IoC 容器注册这些 BeanDefinition 的过程。这个过程是通过调用 BeanDefinitio Registry 接口的实现来完成的。这个注册过程把载入过程中解析得到的 BeanDefinition 向 IoC 容器行注册。通过分析，我们可以看到，在 IoC 容器内部将 BeanDefinition 注入到一个 HashMap 中去，oC 容器就是通过这个 HashMap 来持有这些 BeanDefinition 数据的。值得注意的是，这里谈的是 Io 容器初始化过程，在这个过程中，一般不包含 Bean 依赖注入的实现。在 Spring IoC 的设计中，Bea 定义的载入和依赖注入是两个独立的过程。依赖注入一般发生在应用第一次通过 getBean 向容器索取 Bean 的时候。但有一个例外值得注意，在使用 IoC 容器时有一个预实例化的配置，通过这个预实例的配置（具体来说，可以通过为 Bean 定义信息中的 lazyinit 属性），用户可以对容器初始化过程作个微小的控制，从而改变这个被设置了 lazyinit 属性的 Bean 的依赖注入过程。
<h2 id="Spring容器如何创建-">Spring 容器如何创建？</h2>
在 web.xml 文件中配置 ServletContextListener 为 spring 的环境启动监听器。
然后启动的时候，读取配置好的 xml 文件，解析这个 xml 文件，根据 xml 文件中的 bean 定义，生实例，填入到 Bean 容器中。之后就是注入了。
<h2 id="SpringMVC容器和Spring容器的区别-">SpringMVC 容器和 Spring 容器的区别？</h2>
Spring 是 web.xml 中注册启动监听器时，读取 spring 相关的 xml 去生成 bean，放入容器。<r>
SringMVC 是 web.xml 中注册的一个 Servlet，一般来说所有请求都经过这个 servlet，这个 serlvet 初始化的时候
读取 springmvc.xml 中的数据，扫描指定的包，注册好所有 Controller,解析请求对应的方法，放入个 Map 中，当
有请求进来的时候就从这个 map 中取对应的 Controller。
<h2 id="SpringMVC的控制器是不是单例模式-如果是-有什么问题-怎么解决-">SpringMVC 的控器是不是单例模式,如果是,有什么问题,怎么解决？</h2>
是单例的。如何保证并发安全：1.不使用 controller 中的实例变量或类变量。2.或者在 controller 注解上加 @Scope("prototype")，使之成为非单例的。3.使用 ThreadLocal 变量。

<h2 id="-Spring中BeanFactory和ApplicationContext的区别-">★Spring 中 BeanFactory 和 ApplicationContext 的区别？</h2>

BeanFactory 和 ApplicationContext 都是接口，并且 ApplicationContext 是 BeanFactory 的接口。

BeanFactory 是 Spring 中最底层的接口，提供了最简单的容器的功能，只提供了实例化对象和对象的功能。

而 ApplicationContext 是 Spring 的一个更高级的容器，提供了更多的有用的功能。提供更多向应用的功能。

ApplicationContext 提供的额外的功能：国际化的功能、消息发送、响应机制、统一加载资源功能、强大的事件机制、对 Web 应用的支持等等。

事件传播：载入多个（有继承关系）上下文，使得每一个上下文都专注于一个特定的层次，比应用的 web 层，更易于创建实际应用。

加载方式的区别：BeanFactory 采用的是延迟加载的形式来注入 Bean；ApplicationContext 相反的，它是在 Ioc 启动时就一次性创建所有的 Bean,好处是可以马上发现 Spring 配置文件中的错，坏处是启动时间会比较耗时。

<h2 id="什么是IoC和DI-DI是如何实现的-">什么是 IoC 和 DI？DI 是如何实现的？</h2>

IOC 是控制反转（Inversion of Control，缩写为 IoC），是面向对象编程中的一种设计原则，循了依赖倒置的原则。可以用来减低计算机代码之间的耦合度。控制指的是对实现类的控制,反转指的这种控制权从调用类中移除,交给第三方（容器）决定。
而控制反转其中最常见实现方式叫做依赖注入（Dependency Injection，简称 DI）。通过控制转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用传递(注)给它。
还有一种 ioc 的实现方式叫“依赖查找”（Dependency Lookup），但 Martin Fowler 认为 DI 的实现方式更灵活解耦。

<h2 id="-请问Spring中Bean的作用域有哪些-">★ 请问 Spring 中 Bean 的作用域有哪些？</h2>

singleton 为默认值，IOC 容器中仅存在一个 Bean 实例，Bean 都以单例模式存在
prototype，在每次请求获取 Bean 的时候，都会创建一个新的实例，它在容器初始化的时候不创建实例，采用的是延迟加载的形式注入
Bean，当你使用的时候，才会进行实例化，每次实例化获取的对象都不是同一个 就像 BeanFactory 实例化模式 实例不唯一
request，在每一次 http 请求时会创建一个实例，该实例仅在当前 http request 有效
session，在每一次 http 请求时会创建一个实例，该实例仅在当前 http session 有效
globalSession，全局 Session，仅供基于 Porlet 的 web 环境进行使用。（很少使用到）

<h2 id="-Spring容器对Bean组件是如何管理的-">★Spring 容器对 Bean 组件是如何管理的？</h2

Bean 对象创建的时机：懒加载或饥饿加载
Bean 对象创建的个数：原型或者单例
Bean 对象初始化和销毁：可配置初始化或者销毁调用的方法。
实例化 Bean 地方时：构造器、静态工厂、实例工厂

<h2 id="请谈一谈Spring中自动装配的方式有哪些-">请谈一谈 Spring 中自动装配的方式有哪些？<
h2>

no：不进行自动装配，手动设置 Bean 的依赖关系。
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：类似于 byType，不过是应用于构造器的参数，如果正好有一个 Bean 与构造器的数类型相同则可以自动装配，否则会导致错误。
autodetect：如果有默认的构造器，则通过 constructor 的方式进行自动装配，否则使用 byType 的方式进行自动装配

<h2 id="aop的应用场景-">aop 的应用场景？</h2>

事务
权限
日志
缓存
性能统计

<h2 id="Spring-AOP的配置">Spring AOP 的配置</h2>
SpringAOP，XML 配置，切面 切点，连接切点和通知方法 和 等，注解可以直接使 @before 执行方法 @after，@before(“pointcut()”)，@after("pointcu")， @Aroud("excutete())，@AfteReturning，@AfterThrowing，可作日志事务，权限等待，AOP 即通过把具体的类创建对应的 代理类，从代理类来对具体进行操作。
目标实现了接口，默认采用 JDK 实现 AOP，也可以强制使用 CGlib 来实现 AOP，目标没有实接口的话，则必须采用 CGlib，Spring 自动在 JDK 和 CGlib 切换。如果要求 spring 强制使用 CGlib 实现 AOP，则可以配置，添加 Cglib 库。。。jar， Spring 配置文件中加入 <aop:aspecj-autoproy proxy-target-Class=true>
<h2 id="AOP的原理-">AOP 的原理？</h2>
<pre><code class="highlight-chroma">Spring AOP中的代理使用的默认策略是：
如果目标对象实现接口，则默认采用JDK动态代理
如果目标对象没有现接口，则采用CgLib进行动态代理
如果目标对象实现接口，且强制CgLib代理，则采用CgLib进行动态代理
</code></pre>
<h2 id="-你如何理解AOP中的连接点-Joinpoint--切点-Pointcut--增强-Advice--引介-Introduction-织入-Weaving--切面-Aspect-这些概念-">★ 你如何理解 AOP 中的连接点（Joinpoint）、切点（Pintcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念</h2>

连接点（Joinpoint）：程序执行的某个特定位置（如：某个方法调用前、调用后，方法抛出异后）。一个类或一段程序代码拥有一些具有边界性质的特定点，这些代码中的特定点就是连接点。Spring 仅支持方法的连接点。
切点（Pointcut）：如果连接点相当于数据中的记录，那么切点相当于查询条件，一个切点可以

配多个连接点。Spring AOP 的规则解析引擎负责解析切点所设定的查询条件，找到对应的连接点。</i>
增强（Advice）：增强是织入到目标类连接点上的一段程序代码。Spring 提供的增强接口都是方位名的，如：BeforeAdvice、AfterReturningAdvice、ThrowsAdvice 等。很多资料上将增强译“通知”，这明显是个词不达意的翻译，让很多程序员困惑了许久。
说明： Advice 在国内的很多书面资料中都被翻译成”通知”，但是很显然这个翻译无法表达其本质有少量的读物上将这个词翻译为”增强”，这个翻译是对 Advice 较为准确的诠释，我们通过 AOP 横切关注功能加到原有的业务逻辑上，这就是对原有业务逻辑的一种增强，这种增强可以是前置增强后置增强、返回后增强、抛异常时增强和包围型增强。
引介（Introduction）：引介是一种特殊的增强，它为类添加一些属性和方法。这样，即使一个务类原本没有实现某个接口，通过引介功能，可以动态的为该业务类添加接口的实现逻辑，让业务类为这个接口的实现类。
织入（Weaving）：织入是将增强添加到目标类具体连接点上的过程，AOP 有三种织入方式：1编译期织入：需要特殊的 Java 编译期（例如 AspectJ 的 ajc）；2.装载期织入：要求使用特殊的类加器，在装载类的时候对类进行增强；3.运行时织入：在运行时为目标类生成代理实现增强。Spring 采了动态代理的方式实现了运行时织入，而 AspectJ 采用了编译期织入和装载期织入的方式。
切面（Aspect）：切面是由切点和增强（引介）组成的，它包括了对横切关注功能的定义，也包了对连接点的定义。

<h2 id="请问Spring支持的事务管理类型有哪些-">请问 Spring 支持的事务管理类型有哪些？</h2>

声明式事务
本质使用 AOP，将业务和事务管理分离，降低耦合度和提高事务的复用能力。声明式事务可以通过解 @Transactional 和配置来管理事务，操作简单。

编程式事务
编程式事务是在代码中使用 TransactionTemplate 进行硬编码，与业务的耦合度高，难以复用。但制事务的范围比较灵活。

<h2 id="Spring事务的传播特性-">Spring 事务的传播特性？</h2>
1.PROPAGATION_REQUIRED 如果存在一个事务，则支持当前事务。如果没有事务则开启一个的事务。
2.PROPAGATION_SUPPORTS 如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行但是对于事务同步的事务管理器，PROPAGATION_SUPPORTS 与不使用事务有少许不同。
3.PROPAGATION_MANDATORY 如果已经存在一个事务，支持当前事务。如果没有一个活动的事，则抛出异常。
4.PROPAGATION_REQUIRES_NEW 总是开启一个新的事务。如果一个事务已经存在，则将这个存的事务挂起。
5.PROPAGATION_NOT_SUPPORTED 总是非事务地执行，并挂起任何存在的事务。
6.PROPAGATION_NEVER 总是非事务地执行，如果存在一个活动事务，则抛出异常
7.PROPAGATION_NESTED 如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事, 则按 TransactionDefinition.PROPAGATION_REQUIRED 属性执行
<h2 id="Spring事务的隔离级别-">Spring 事务的隔离级别？</h2>
ISOLATION_READ_UNCOMMITTED 读未提交
ISOLATION_READ_COMMITTED 读已提交
ISOLATION_REPEATABLE_READ 可重复读
ISOLATION_SERIALIZABLE 串行读
<h2 id="Spring的通知类型有哪些-">Spring 的通知类型有哪些？</h2>
前置通知、正常返回通知、异常返回通知、返回通知、环绕通知
<h2 id="Spring的优点-">Spring 的优点？</h2>

降低了组件之间的耦合性，实现了软件各层之间的解耦
可以使用容易提供的众多服务，如事务管理，消息服务等
容器提供单例模式支持
容器提供了 AOP 技术，利用它很容易实现如权限拦截，运行期监控等功能
容器提供了众多的辅助类，能加快应用的开发
spring 对于主流的应用框架提供了集成支持，如 hibernate，JPA，Struts 等
spring 属于低侵入式设计，代码的污染极低
独立于各种应用服务器
spring 的 DI 机制降低了业务对象替换的复杂性
Spring 的高度开放性，并不强制应用完全依赖于 Spring，开发者可以自由选择 spring 的部分全部

<h2 id="请阐述一下Hibernate实体对象的三种状态是什么-以及对应的转换关系是什么--">请阐述下 Hibernate 实体对象的三种状态是什么？以及对应的转换关系是什么？？</h2>
<pre><code class="highlight-chroma">瞬时态，持久态和游离态。
瞬时态没有id，对刚new出来的时候处于瞬时态。
持久态有id（与数库中的主键关联）,通过get()或者load()方法获取的对象处于持久态。
游离态:持久态的对脱离了session的管理变成游离态，处于游离态的对象如果不被其它对象引用则会被垃圾回收机制回。
转化:
瞬时转持久：Session对象的save()或saveOrUpdate()方法保存对象
持久转瞬时，调用delete()方法
持久转游离：调用Session对象的clear()/close()/evict()
游离转瞬时：调用delete()方法
游离转持久: 调用update()或者saveOrUpdate()
</code></pre>
<h2 id="Hibernate中SessionFactory是线程安全的吗-Session是线程安全的吗-两个线程能够共享一个Session吗--">Hibernate 中 SessionFactory 是线程安全的吗？Session 是线程安全的吗（两个程能够共享同一个 Session 吗）？</h2>
<pre><code class="highlight-chroma">SessionFactory对应Hibernate的一个数据存储的概念，它是线程安全的，可以被多个线程并发问。
SessionFactory一只会在启动的时候构建。对于应用程序，最好将SessionFactory通过单例模式进行封装以便于访问。
Session是一个轻级非线程安全的对象（线程间不能共享session），它表示与数据库进行交互的一个工作单元。
Session是由SessioFactory创建的，在任务完成之后它会被关闭。Session是持久层服务对外提供的主要接口。
Session会延迟获数据库连接（也就是在需要的时候才会获取）。为了避免创建太多的session，可以使用ThreadLocal
将session和当前程绑定在一起，这样可以让同一个线程获得的总是同一个session。Hibernate 3中SessionFactory的
getCurrentSession)方法就可以做到。
</code></pre>

<h2 id="Hibernate中Session的load和get方法的区别是什么-">Hibernate 中 Session 的 load 和 get 方法的区别是什么？</h2>

如果没有找到符合条件的记录，get 方法返回 null，load 方法抛出异常。
get 方法直接返回实体类对象，load 方法返回实体类对象的代理(如果 lazy='true'，就使用延迟载，返回的是代理，不是真实对象)。
在 Hibernate 3 之前，get 方法只在一级缓存中进行数据查找，如果没有找到对应的数据则越过级缓存，直接发出 SQL 语句完成数据读取；load 方法则可以从二级缓存中获取数据；从 Hibernate 3 开始，get 方法不再是对二级缓存只写不读，它也是可以访问二级缓存的。

<h2 id="请问Query接口的list方法和iterate方法有什么区别-">请问 Query 接口的 list 方法和 iterate 方法有什么区别？</h2>

list()方法返回的每个对象都是完整的，而 iterator()方法所返回的对象中仅包含了主键值（标识）。
只有当你对 iterator 中的对象进行操作时，Hibernate 才会向数据库再次发送 SQL 语句来获取对象的属性值。相当于延迟加载。

<h2 id="如何理解Hibernate的延迟加载机制-在实际应用中-延迟加载与Session关闭的矛盾是如何理的-">如何理解 Hibernate 的延迟加载机制？在实际应用中，延迟加载与 Session 关闭的矛盾是如处理的？</h2>
<pre><code class="highlight-chroma">延迟加载就是并不是在读取的时候就把数据加载进来，而是等到使用时再加载。Hibernate使用虚拟代理机制实现
延迟加载，我们使Session的load()方法加载数据或者一对多关联映射在使用延迟加载的情况下从一的一方加载多
的一方，得到的都虚拟代理，简单的说返回给用户的并不是实体本身，而是实体对象的代理。代理对象在用户调用
getter方法时才会数据库加载数据。但加载数据就需要数据库连接。而当我们把会话关闭时，数据库连接就同时关
闭了。

延迟加载与sessio关闭的矛盾一般可以这样处理：
1. 关闭延迟加载特。这种方式操作起来比较简单，因为Hibernate的延迟加载特性是可以通过映射文件或者注解进行配的，
但这种解决方案存明显的缺陷。首先，出现"no session or session was closed"通常说明系统中已经存在主外键关联，

如果去掉延迟加载话，每次查询的开销都会变得很大。
2. 在session关闭前先获取需要查询的数据，可以使用工具方法Hibernate.isInitialized()判断对象是否被加载，如果没
有被加载则可以使Hibernate.initialize()方法加载对象。
3. 使用拦截器或过器延长Session的生命周期直到视图获得数据。Spring整合Hibernate提供的OpenSessionInViewFilte和
OpenSessionInViwInterceptor就是这种做法。
</code></pre>
<h2 id="hibernate和ibatis的区别">hibernate 和 ibatis 的区别</h2>

Hibernate 是完全自动的，利用配置好的对象关系模型，不用写任何 sql 语句就能实现数据库的作。
而 Mybatis 是半自动的，仅自动映射字段，但基本的 sql 是需要自己编写。
Hibernate 使用完全自动的映射操作数据库以及 hql 语句使得数据库移植性更强。
Mybatis 在 sql 的直接优化上优于 Hibernate。

<h2 id="解释一下MyBatis中命名空间-namespace-的作用-">解释一下 MyBatis 中命名空间（namespace）的作用。</h2>
<pre><code class="highlight-chroma">在大型项目中，可能存在大量的SQL语句，这时候为每个SQL语句起一个唯一的标识（ID）就变并不容易了。
为了解决这个问题在MyBatis中，可以为每个映射文件起一个唯一的命名空间，这样定义在这个映射文件中
的每个SQL语句就了定义在这个命名空间中的一个ID。只要我们能够保证每个命名空间中这个ID是唯一的，
即使在不同映射文中的语句ID相同，也不会再产生冲突了。
</code></pre>
<h2 id="MyBatis中的动态SQL是什么意思-">MyBatis 中的动态 SQL 是什么意思？</h2>
<pre><code class="highlight-chroma">MyBatis的动态SQL是基于OGNL表达式的，它可以帮助我们方便的在SQL语句中实现某些逻辑。
</code></pre>
<h2 id="MyBatis中-和-的区别-">MyBatis 中#和 $ 的区别？</h2>
通过直接拼接进sql中，#是预编译再填入参数。所以</span
不安全，#较安全。
<h2 id="MyBatis一级缓存原理以及失效情况-">MyBatis 一级缓存原理以及失效情况？</h2>
一级缓存的作用域在 session 级别，在一个事务中，两次查询用的是一个 sqlsession，非事务中两次查询用的是不同的 sqlsession。配置正确的情况下，失效的原因一般都是没有在同一个 session。
<h2 id="MyBatis二级缓存的使用-">MyBatis 二级缓存的使用？</h2>
二级缓存是针对 Mapper 的 namespace 的，同 namespace 的 Mapper 中不同 sqlsession 可获取到缓存，但发生更新操作的时候，会清除缓存。在查询操作远远多于增删改操作的情况下可以使二级缓存。因为任何增删改操作都将刷新二级缓存，对二级缓存的频繁刷新将降低系统性能。最好在表上使用二级缓存。
<h2 id="springmvc和spring-boot区别-">springmvc 和 spring-boot 区别？</h2>
<pre><code class="highlight-chroma">SpringMVC是MVC框架,Spring Boot是快速开发基于Spring的应用.Spring Boot 通过stater的式简化
整个Spring生态配, SpringMVC, 对应的starter就是spring-boot-starter-web, 开发体验上来说,
没有很多xml配置,内置了tomcat.
</code></pre>
<h2 id="-看过MyBatis源码吗-请说说它的工作流程-">★ 看过 MyBatis 源码吗，请说说它的工作流？</h2>
1.加载配置文件。
2.创建会话工厂。
3.创建会话。
4.创建执行器。（可能会执行拦截器）
5.输入映射封装到 sql,去数据库查询，返回结果集封装给应用。
<h2 id="-MyBatis拦截器原理-">★MyBatis 拦截器原理？</h2>

mybatis 给 Executor、StatementHandler、ResultSetHandler、ParameterHandler 提供了截器功能，

Executor 提供了增删改查的接口.
StatementHandler 负责处理 Mybatis 与 JDBC 之间 Statement 的交互.
ResultSetHandler 负责处理 Statement 执行后产生的结果集，生成结果列表.
ParameterHandler 是 Mybatis 实现 Sql 入参设置的对象。
拦截器采用了责任链模式，把请求发送者和请求处理者分开，各司其职。

<h2 id="-Spring-MVC完整工作流程">★Spring MVC 完整工作流程</h2>

用户发起请求到前端控制器（DispatcherServlet），该控制器会过滤出哪些请求可以访问 Servle
、哪些不能访问。就是 url-pattern 的作用，并且会加载 springmvc.xml 配置文件。
前端控制器会找到处理器映射器（HandlerMapping），通过 HandlerMapping 完成 url 到 con
roller 映射的组件，简单来说，就是将在 springmvc.xml 中配置的或者注解的 url 与对应的处理类找
并进行存储，用 map<url,handler> 这样的方式来存储。
HandlerMapping 有了映射关系，并且找到 url 对应的处理器，HandlerMapping 就会将其处
器（Handler）返回，在返回前，会加上很多拦截器。
DispatcherServlet 拿到 Handler 后，找到 HandlerAdapter（处理器适配器），通过它来访问
理器，并执行处理器。
执行处理器
处理器会返回一个 ModelAndView 对象给 HandlerAdapter
通过 HandlerAdapter 将 ModelAndView 对象返回给前端控制器(DispatcherServlet)
前端控制器请求视图解析器(ViewResolver)去进行视图解析，根据逻辑视图名解析成真正的视图(j
p)，其实就是将 ModelAndView 对象中存放视图的名称进行查找，找到对应的页面形成视图对象</li

返回视图对象到前端控制器。
视图渲染，就是将 ModelAndView 对象中的数据放到 request 域中，用来让页面加载数据的。<
li>
通过第 8 步，通过名称找到了对应的页面，通过第 10 步，request 域中有了所需要的数据，那
就能够进行视图渲染了。最后将其返回即可。

<h2 id="SpringMVC如何处理JSON数据-">SpringMVC 如何处理 JSON 数据？</h2>
在 xml 中配置转换 Json 的类，运行时就会调用这个类进行转换。
<h2 id="SpringMVC常见注解有哪些-">SpringMVC 常见注解有哪些？</h2>
@Controller 、@RequestMapping、@Autowired、@Resource、@PathVariable、@Requ
stParam、
@ModelAttribute、@SessionAttribute
<h2 id="Hibernate继承映射策略">Hibernate 继承映射策略</h2>
继承关系的映射策略有三种：

每个继承结构一张表（table per class hierarchy），不管多少个子类都用一张表。
每个子类一张表（table per subclass），公共信息放一张表，特有信息放单独的表。
每个具体类一张表（table per concrete class），有多少个子类就有多少张表。

<h2 id="SpringMVC拦截器原理-如何自定义拦截器-">SpringMVC 拦截器原理，如何自定义拦截
？</h2>
跟 web.xml 的过滤器链差不多。通过统一的 Servlet 入口进入，针对的是 Controller，在 Contr
ller 前后做一些增强性的操作。
<h2 id="SpringMVC如何将请求映射定位到方法上面-结合源码阐述-">SpringMVC 如何将请求映
定位到方法上面？结合源码阐述？</h2>
SpringMVC 根据请求的路径，在一个类似于 Map 的结构中，以路径为键值找到对应的 Handle
。
<h2 id="-请说说SpringBoot自动装配原理-">★ 请说说 SpringBoot 自动装配原理？</h2>
自动装配的过程：

通过各种注解 + 继承，引入包含自动装配核心方法的类

SpringApplication.run(Application.class, args)在运行时，调用自动装配方法
自动装配方法会读取 spring-boot-autoconfigure.jar 里面的 spring.factories 配置文件，配置件中有所有自动装配类的配置类的类名
生成对应功能的 Configuration 类，这些功能配置类要生效的话，会去 classpath 中找是否有该的依赖类（也就是 pom.xml 必须有对应功能的 jar 包才行）
配置类里再通过判断生成最后的功能类，并且配置类里面注入了默认属性值类，功能类可以引用赋默认值。生成功能类的原则是自定义优先，没有自定义时才会使用自动装配类。

综上所述，要想自动装配一个类需要满足 2 个条件：

spring.factories 里面有这个类的配置类（一个配置类可以创建多个围绕该功能的依赖类）
2 .pom.xml 里面需要有对应的 jar 包

自动装配的结果：

根据各种判断和依赖，最终生成了业务需要的类并且注入到 IOC 容器当中了
自动装配生成的类赋予了一些默认的属性值

<h2 id="SpringBoot的优点-">SpringBoot 的优点？</h2>
快速构建项目，极大的提高了开发、部署效率。
对主流开发框架的无配置集成。
项目可独立运行，无须外部依赖 Servlet 容器。
提供运行时的应用监控。
<h2 id="-Spring-MVC和-Spring-IOC的生命周期">★Spring MVC 和 Spring IOC 的生命周期</h2

SpirngMVC 的生命周期 ：

A，-》DispatcherSerlvet（前端控制器）

B，-》HandlerMapping（处理器映射器），根据 xml 注解查找对应的 Hander -》 返回 Handlr

C，-》处理器适配器去执行 Handler

D，-》Handler 执行完成后给处理器适配器返回 ModelAndView

E，-》前端控制器请求视图解析器去执行视图解析，根据逻辑视图名解析成真正的视图 JSP，向端控制器返回 view

F，-》前端控制器进行视图渲染，将模型数据放到 request-》返回给用户

SpringBean 的生命周期：
Instance 实例化-》设置属性值-》调用 BeanNameAware 的 setBeanName 方法-》调用 BeanostProsessor 的预初始化方法-》调用 InitializationBean 的 afterPropertiesSet()的方法-》调用定的初始化方法 callCustom 的 init-method-》调用 BeanPostProsessor 的后初始化方法-》Bean 可使用了 -》 容器关闭-》 调用 DisposableBean 的 destroy 方法-》调用定制的销毁方法 CallCustom 的 destroy-method。

<h2 id="-Spring-MyBatis实现读写分离简述-">★Spring+MyBatis 实现读写分离简述？</h2>
<pre><code class="highlight-chroma">方案一：通过MyBatis配置文件创建读写分离两个DataSource，每个SqlSessionFactoryBean对的mapperLocations属性制定两个读写数据源的配置文件。将所有读的操作配置在读文件中，所有写操作配置在写文件中。
方案二：通过Sprin AOP在业务层实现读写分离，在DAO层调用前定义切面，利用Spring的AbstractRoutingDataSourc解决多数据源的问题，实现动态选择数据源
方案三：通过Myba is的Plugin在业务层实现数据库读写分离，在MyBatis创建Statement对象前通过拦截器选择真正的据源，在拦截器中根据方法名称不同（select、update、insert、delete）选择数据源。
方案四：通过sprin 的AbstractRoutingDataSource和mybatis Plugin拦截器实现非常友好的读写分离，原有代码不需要何改变。推荐第四种方案
</code></pre>
<h2 id="Quartz实现原理-">Quartz 实现原理?</h2>
A、scheduler 是一个计划调度器容器（总部），容器里面可以盛放众多的 JobDetail 和 trigger 当容器启动后，里面的每个 JobDetail 都会根据 trigger 按部就班自动去执行。
B、JobDetail 是一个可执行的工作，它本身可能是有状态的。
C、Trigger 代表一个调度参数的配置，什么时候去调。
D、当 JobDetail 和 Trigger 在 scheduler 容器上注册后，形成了装配好的作业（JobDetail 和 Trigg r 所组成的一对儿），就可以伴随容器启动而调度执行了。
E、scheduler 是个容器，容器中有一个线程池，用来并行调度执行每个作业，这样可以提高容器效 。
<h2 id="springmvc用到的注解-作用是什么-">springmvc 用到的注解，作用是什么？</h2>

@Controller 或 @RestController 标记该类是一个控制器
@RequestMapping 通过这个注解可以定义不同的处理器映射规则，即为控制器指定可以处理些 URL 请求。
@RequestBody 用于读取 http 请求的内容（字符串），通过 springMVC 提供的 HttpMessag Converter 接口将读取到的内容转换为 json、xml 等格式的数据，再转换为 java 对象绑定到 Control er 类方法的参数上。
@ResponseBody 注解表示该方法的返回的结果直接写入 HTTP 响应正文（ResponseBody），一般在异步获取数据时使用，通常是在使用 @RequestMapping 后，返回值通常解析为跳转路径加上 @Responsebody 后返回结果不会被解析为跳转路径，而是直接写入 HTTP 响应正文中。
@ModelAndAttribute 绑定请求参数到指定对象\暴露表单引用对象为模型数据 \暴露 @Reques Mapping 方法返回值为模型数据
@RequestParam 处理简单类型的绑定
@PathVariable 绑定 URL 占位符到入参。
@ExceptionHandler 注解到方法上, 出现异常时会执行该方法。
@ControllerAdvice 使一个 Controller 成为全局的异常处理类, 类中用 ExceptinHandler 方 注解的方法可以处理所有 Controller 发生的异常。
@Autowired 它可以对类成员变量、方法以及构造函数进行标注，完成自动装配的工作。自动装 的意思就是让 Spring 从应用上下文中找到对应的 bean 的引用，并将它们注入到指定的 bean。

<h2 id="-springboot启动机制-">★springboot 启动机制。</h2>

如果我们使用的是 SpringApplication 的静态 run 方法，那么，这个方法里面首先要创建一个 Sp ingApplication 对象实例，然后调用这个创建好的 SpringApplication 的实例方法。在 SpringAppli ation 实例初始化的时候，它会提前做几件事情：根据 classpath 里面是否存在某个特征类（org.spri gframework.web.context.ConfigurableWebApplicationContext）来决定是否应该创建一个为 We 应用使用的 ApplicationContext 类型。使用 SpringFactoriesLoader 在应用的 classpath 中查找并 载所有可用的 ApplicationContextInitializer。使用 SpringFactoriesLoader 在应用的 classpath 中 找并加载所有可用的 ApplicationListener。推断并设置 main 方法的定义类。

SpringApplication 实例初始化完成并且完成设置后，就开始执行 run 方法的逻辑了，方法执行始，首先遍历执行所有通过 SpringFactoriesLoader 可以查找到并加载的 SpringApplicationRunListner。调用它们的 started()方法，告诉这些 SpringApplicationRunListener，“嘿，SpringBoot 应要开始执行咯！”。
创建并配置当前 Spring Boot 应用将要使用的 Environment（包括配置要使用的 PropertySourc 以及 Profile）。
遍历调用所有 SpringApplicationRunListener 的 environmentPrepared()的方法，告诉他们：当前 SpringBoot 应用使用的 Environment 准备好了咯！”。
如果 SpringApplication 的 showBanner 属性被设置为 true，则打印 banner。
根据用户是否明确设置了 applicationContextClass 类型以及初始化阶段的推断结果，决定该为前 SpringBoot 应用创建什么类型的 ApplicationContext 并创建完成，然后根据条件决定是否添加 SutdownHook，决定是否使用自定义的 BeanNameGenerator，决定是否使用自定义的 ResourceLoder，当然，最重要的，将之前准备好的 Environment 设置给创建好的 ApplicationContext 使用。
ApplicationContext 创建好之后，SpringApplication 会再次借助 Spring-FactoriesLoader，找并加载 classpath 中所有可用的 ApplicationContext-Initializer，然后遍历调用这些 ApplicationCntextInitializer 的 initialize（applicationContext）方法来对已经创建好的 ApplicationContext 进进一步的处理。
遍历调用所有 SpringApplicationRunListener 的 contextPrepared()方法。
最核心的一步，将之前通过 @EnableAutoConfiguration 获取的所有配置以及其他形式的 IoC 器配置加载到已经准备完毕的 ApplicationContext。
遍历调用所有 SpringApplicationRunListener 的 contextLoaded()方法。
调用 ApplicationContext 的 refresh()方法，完成 IoC 容器可用的最后一道工序。
查找当前 ApplicationContext 中是否注册有 CommandLineRunner，如果有，则遍历执行它。
正常情况下，遍历执行 SpringApplicationRunListener 的 finished()方法、（如果整个过程出异常，则依然调用所有 SpringApplicationRunListener 的 finished()方法，只不过这种情况下会将常信息一并传入处理）

<h2 id="-srpingMVC启动原理">★srpingMVC 启动原理</h2>

解析键值对
创建一个 application 对象即 ServletContext，servlet 上下文，用于全局共享
将键值对放入 ServletContext 中，web 应用全局共享
读取标签，创建监听器，一般使用 ContextLoaderListener，如果使用了 ContextLoaderListene，Spring 就会创建一个 WebApplicationContext 对象，这个就是 IOC 容器, ContextLoaderListene 创建的 IOC 容器是全局共享的，并将其放在 ServletContext 中, 键名为 WebApplicationContext.ROT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, 读取 web.xml 文件里的 contextConfigLocation 配置中的 xml 文件来创建 bean
listener 创建完毕后如果有 Filter 会去创建 Filter
初始化 Servlet，一般使用 DispatchServlet 类
DispatchServlet 的父类 FrameworkServlet 会重写其父类的 initServletBean 方法，并调用 iniWebApplicationContext()以及 onRefresh()方法
initWebApplicationContext()方法会创建一个当前 servlet 的一个 IOC 子容器，如果存在上述全局 WebApplicationContext 则将其设置为父容器，如果不存在上述全局的则父容器为 null。
读取标签的配置的 xml 文件并加载相关 Bean
onRefresh()方法创建 Web 应用相关组件

<h2 id="-AOP的原理详解">★AOP 的原理详解</h2>
AOP 实现中，可以看到三个主要的步骤，一个是代理对象的生成，然后是切面的织入，然后是行拦截器链。

Spring AOP 代理对象的生成：

Spring 提供了两种方式来生成代理对象: JDKProxy 和 Cglib，具体使用哪种方式生成由 AopProxyFactory 根据 AdvisedSupport 对象的配置来决定。默认的策略是如果目标类是接口，则使用 JDK 态代理技术，否则使用 Cglib 来生成代理。Spring 使用 JDK 来生成代理对象，具体的生成代码放在 JkDynamicAopProxy 这个类中。
2. 切面的织入：
获取可以应用到此方法上的通知链,Advised 中配置能够应用到连接点或者目标类的 Advisor 全被转化成了 MethodInterceptor.
3. 执行拦截器链：
如果得到的拦截器链为空，则直接反射调用目标方法，否则创建 MethodInvocation，调用其 proceed 方法，触发拦截器链的执行，
注意:

CGLib 不能对声明为 final 的方法进行代理，因为 CGLib 原理是动态生成被代理类的子类。
JDK 代理三要素 Proxy.newProxyInstance(类加载器，目标接口，InvocationHandler 实现)，JKDynaimicAopProxy 实现了 InvocationHandler 接口，并使用了 AdvisedSupport 中存在的目标口，及设置的类加载器，进行 JDK 的动态代理。
CGLIB 创建代理主要是创建 Enhancer enhancer，并通过 AdvisedSupport 设置相应的属性，如目标类 rootClass，如果由接口合并接口给代理类，最主要的是设置 Callback 集合和 CallbackFilter 使用 CallBackFilter 可以根据方法的不同使用不同的 Callback 进行拦截和增强方法。其中最主要的用于 AOP 的 Callback 是 DynamicAdvisedInterceptor。

<h2 id="-spring中循环注入的方式">★spring 中循环注入的方式</h2>

能成功的循环注入方式都是因为能先创建出实例放入到临时创建 Bean 池当中。
构造器循环依赖（失败）
setter 方法循环注入（成功）
setter 方法注入 单例模式(scope=singleton)（成功）
setter 方法注入 非单例模式(scope=prototype)（失败）原型 bean 的话，不会放入到临时创建 ean 池当中。

<h2 id="-Spring的BeanFactory和FactoryBean的区别">★Spring 的 BeanFactory 和 FactoryBean 的区别</h2>
BeanFactory 接口用来生产 Bean，它处理生产 bean 的接口体系的最顶层，它为其他具体的 IOC 容器提供了最基本的规范，例如 DefaultListableBeanFactory，XmlBeanFactory，ApplicationContext 等具体的容器都是实现了 BeanFactory，再在其基础之上附加了其他的功能。
FactoryBean 接口用来定制 Bean 的生产过程，getObject 方法中可以实现自定义过程。MyBatis 的 qlSessionFactoryBean 中，Spring 会调用 SqlSessionFactoryBean 这个实现了 FactoryBean 的工厂 Bean 同时加载 dataSource,Mapper 文件的路径,对 sqlSessionFactory 进行初始化。
FactoryBean 比 BeanFactory 在生产 Bean 的时候灵活，还能修饰对象，带有工厂模式和装饰模式设计思想在里面，不过它的存在还是以 Bean 的形式存在。BeanFactory 因为是核心接口，编写复杂辑很容易接触到其他不必要的接口，不好实现。
<h2 id="spring-boot特性-优势-适用场景等">spring boot 特性，优势，适用场景等</h2>

约定优于配置思想
专注与业务逻辑之间思维切换
基于 Spring 的开发提供更快入门体验
开箱即用，没有代码生成，无需 XML 配置。
支持修改默认配置满足特定需求
提供大型项目中常见的非功能性特性，如嵌入 Tomcat 服务器、安全、指标、健康检测、外部配等
很契合微服务场景。

<h2 id="-Mybatis的底层实现原理">★Mybatis 的底层实现原理</h2>

加载 mybatis 全局配置文件（数据源、mapper 映射文件等），解析配置文件，MyBatis 基于 ML 配置文件生成 Configuration，和一个个 MappedStatement（包括了参数映射配置、动态 SQL 语句、结果映射配置），其对应着 <select | update | delete | insert> 标签项。

SqlSessionFactoryBuilder 通过 Configuration 对象生成 SqlSessionFactory，用来开启 SqlSession。

SqlSession 对象完成和数据库的交互：

<pre><code class="highlight-chroma">a、用户程序调用mybatis接口层api（即Mapper接口中的方法）
b、SqlSession通调用api的Statement ID找到对应的MappedStatement对象
c、通过Executor负责动态SQL的生成和查询缓存的维护）将MappedStatement对象进行解析，
sql参数转化、动态ql拼接，生成jdbc Statement对象
d、JDBC执行sql。
e、借助MappedStatement中的结果映射关系，将返回结果转化成HashMap、JavaBean等存储结构并返回。
</code></pre>
<h2 id="quartz和timer对比">quartz 和 timer 对比</h2>
Quartz：

非常灵活，可以使用多种形式的定时方式

非常轻量级，而且只需要很少的设置/配置

是容错的，在系统重启后记住以前的记录

可以参与事务
Timer：

定时器没有持久性机制

创建方便简单，不用另外引入 jar 包

<h2 id="-动态代理的几种方式">★ 动态代理的几种方式</h2>
JDK 原生动态代理：

Proxy：Proxy 是所有动态代理的父类，它提供了一个静态方法来创建动态代理的 class 对象和例。
InvovationHandler：每个动态代理实例都有一个关联的 InvocationHandler。在代理实例上调

方法时，方法调用将被转发到 InvocationHandler 的 invoke 方法。
JDK 原生动态代理是 Java 原生支持的，不需要任何外部依赖，但是它只能基于接口进行代理。

Cglib 代理：是一个基于 ASM 的字节码生成库，它允许我们在运行时对字节码进行修改和动态成。Cglib 通过继承方式进行代理。

Enhancer：来指定要代理的目标对象、实际处理代理逻辑的对象，最终通过调用 create()方法得代理对象，对这个对象所有非 final 方法的调用都会转发给 MethodInterceptor;
MethodInterceptor：动态代理对象的方法调用都会转发到 intercept 方法进行增强。

Cglib 通过继承的方式进行代理，无论目标对象有没有实现接口都可以进行代理，但是无法处理 final 的情况。
<h2 id="tomcat如何调优-涉及哪些参数--">tomcat 如何调优，涉及哪些参数 。</h2>
一般调线程参数：

maxThreads :Tomcat 使用线程来处理接收的每个请求，这个值表示 Tomcat 可创建的最大的线数，默认值是 200
minSpareThreads：最小空闲线程数，Tomcat 启动时的初始化的线程数，表示即使没有人使用开这么多空线程等待，默认值是 10。
maxSpareThreads：最大备用线程数，一旦创建的线程超过这个值，Tomcat 就会关闭不再需的 socket 线程。上边配置的参数，最大线程 500（一般服务器足以），要根据自己的实际情况合理置，设置越大会耗费内存和 CPU，因为 CPU 疲于线程上下文切换，没有精力提供请求服务了，最小闲线程数 20，线程最大空闲时间 60 秒，当然允许的最大线程连接数还受制于操作系统的内核参数设，设置多大要根据自己的需求与环境。当然线程可以配置在“tomcatThreadPool”中，也可以直接置在“Connector”中，但不可以重复配置。
URIEncoding：指定 Tomcat 容器的 URL 编码格式，语言编码格式这块倒不如其它 WEB 服务软件配置方便，需要分别指定。
connnectionTimeout： 网络连接超时，单位：毫秒，设置为 0 表示永不超时，这样设置有隐的。通常可设置为 30000 毫秒，可根据检测实际情况，适当修改。
enableLookups： 是否反查域名，以返回远程主机的主机名，取值为：true 或 false，如果设置 false，则直接返回 IP 地址，为了提高处理能力，应设置为 false。
disableUploadTimeout：上传时是否使用超时机制。
connectionUploadTimeout：上传超时时间，毕竟文件上传可能需要消耗更多的时间，这个根你自己的业务需要自己调，以使 Servlet 有较长的时间来完成它的执行，需要与上一个参数一起配合用才会生效。
acceptCount：指定当所有可以使用的处理请求的线程数都被使用时，可传入连接请求的最大队长度，超过这个数的请求将不予处理，默认为 100 个。
keepAliveTimeout：长连接最大保持时间（毫秒），表示在下次请求过来之前，Tomcat 保持连接多久，默认是使用 connectionTimeout 时间，-1 为不限制超时。
maxKeepAliveRequests：表示在服务器关闭之前，该连接最大支持的请求数。超过该请求数的接也将被关闭，1 表示禁用，-1 表示不限制个数，默认 100 个，一般设置在 100~200 之间。
compression：是否对响应的数据进行 GZIP 压缩，off：表示禁止压缩；on：表示允许压缩（本将被压缩）、force：表示所有情况下都进行压缩，默认值为 off，压缩数据后可以有效的减少页面大小，一般可以减小 1/3 左右，节省带宽。
compressionMinSize：表示压缩响应的最小值，只有当响应报文大小大于这个值的时候才会对文进行压缩，如果开启了压缩功能，默认值就是 2048。
compressableMimeType：压缩类型，指定对哪些类型的文件进行数据压缩。noCompression serAgents="gozilla, traviata"： 对于以下的浏览器，不启用压缩。如果已经对代码进行了动静分离静态页面和图片等数据就不需要 Tomcat 处理了，那么也就不需要配置在 Tomcat 中配置压缩了。

<h2 id="-Springmvc-中DispatcherServlet初始化过程-">★Springmvc 中 DispatcherServlet 初化过程。</h2>

DispatcherServlet 主要的继承体系如下，具体的初始化过程，使用了模板方法模式，将初始化个步骤依次从顶级父类开始执行如下：

HttpServletBean：

　　主要做一些初始化的工作，将 web.xml 中配置的参数设置到 Servlet 中。比如 servlet 标签子标签 init-param 标签中配置的参数。
2. FrameworkServlet：
　　将 Servlet 与 Spring 容器上下文关联。其实也就是初始化 FrameworkServlet 的属性 webpplicationContext，这个属性代表 SpringMVC 上下文，它有个父类上下文，既 web.xml 中配置的ontextLoaderListener 监听器初始化的容器上下文。
3. DispatcherServlet：
　　初始化各个功能的实现类。比如异常处理、视图处理、请求映射处理等。
转自我的 github
<h5 id="技术讨论群QQ-1398880">技术讨论群 QQ:1398880</h5>