链滴

Spring 初级面试题

作者：xiaokedamowang

原文链接：https://ld246.com/article/1629201467279

来源网站：链滴

<h2 id="Spring-初级面试题">Spring 初级面试题</h2>
<h3 id="一--什么是IOC">一. 什么是 IOC</h3>
<blockquote>
什么是 IOC: IOC 是一种设计思想, 对象的管理方式由程序员控制变成了由框架控制, 从创建到销都是框架管理
spring 是通过什么方式来实现 IOC 的: 通过 DI(依赖注入实现)
什么是 DI: DI, 依赖注入, 就是把对应的属性注入到具体的对象中
</blockquote>
<h3 id="二--什么是AOP">二. 什么是 AOP</h3>
<blockquote>
AOP 是一种编程思想, 是对 OOP 面向对象编程的补充, 目前比较常见的 AOP 方式有 2 种, JDK 带的动态代理和 CGlib, 我们需要知道一些 AOP 的术语

通知(Advice): 就是我们写的增强对象的代码
连接点(JoinPoint): 可以被切入的点, 如方法调用前, 方法调用后, 方法返回, 方法抛出异常等
切点(PointCut): 真正被你选中的连接点, 那么多连接点, 你并不是所有的地方需要增强
切面(Aspect): 切面是通知和切点的结合, 把通知应用到切点的过程叫切面

</blockquote>
<h3 id="三--什么是BeanDefinition">三. 什么是 BeanDefinition</h3>
<blockquote>
我们需要 spring 帮我们管理对象, 起码让 spring 知道我们哪些对象需要被管理, 是否需要单例, 哪个构造方法创建, 初始化方法是什么 有很多很多东西需要让 spring 知道, 这些东西就封装成 1 个对, BeanDefinition 就是这种对象实现的接口, 定义了很多获取 Bean 信息的方法
</blockquote>
<h3 id="四--BeanFactory-和-FactoryBean">四. BeanFactory 和 FactoryBean</h3>

BeanFactory
<blockquote>
BeanFactory 是一个接口, 定义了一些基本的方法, springIOC 容器实现了 BeanFactory 接口, 以可以理解为他就是 spring 容器, 常用的实现类有: ApplicationContext, DefaultListableBeanFactory, ClassPathXmlApplicationContext, AnnotationConfigApplicationContext
</blockquote>

FactoryBean
<blockquote>
FactoryBean 也是一个接口, 他定义了 1 个 getObject 的方法, 如果需要容器创建的类实现了这接口, 那么容器会把调用 getObject 方法返回的对象和自己都加入容器, FactoryBean 的名称是在 getbject 名称前面加上 <code>$</code>, 创建比较复杂的对象时, 适合用 FactoryBean
</blockquote>

<h3 id="五--后置处理器-PostProcessor-">五. 后置处理器(PostProcessor)</h3>

BeanFactoryPostProcessor
BeanFactoryPostProcessor 是用来增强容器的, 里面定义了方法 postProcessBeanFactory 方法, 参数是 ConfigurableListableBeanFactory, 这个类实现了 BeanFactory, 传进来的就是 spring 容器身, 你可以修改里面的 BeanDefinition, 或者手动创建 1 个 BeanDefinition 塞进容器
postProcessBeanFactory 在所有的 BeanDefinition 加载完成后调用

BeanPostProcessor
BeanPostProcessor 是用来增强或处理 Bean 的, 在初始化方法前调用 postProcessBeforeInitialization 方法, 在初始化方法后面调用 postProcessAfterInitialization

<h3 id="六--spring的创建流程">六. spring 的创建流程</h3>
<blockquote>
从 new AnnotationConfigApplicationContext()开始
</blockquote>

父类无参构造方法创建 DefaultListableBeanFactory
加载配置类, 创建 spring 自身启动需要的对象的 BeanDefinition, 如: 一些后置处理器, 创建需要 spring 管理的 Bean 的 BeanDefinition
refresh 方法被调用

BeanFactory 的预准备工作, 如:设置类的加载器, 表达式解析器, 加载系统的环境变量等
先执行 BeanDefinitionRegistryPostProcessor, 再执行 BeanFactoryPostProcessor(根据优先接口, 或排序接口, 顺序执行)
创建 BeanPostPreocessor
执行 BeanPostProcessor(根据优先级接口, 或排序接口, 顺序执行)
创建事件派发器, 并且注入到容器
创建所有的监听器, 并绑定到事件派发器中, 然后派发之前步骤产生的事件
finishBeanFactoryInitialization 方法被调用, 初始化剩余的 Bean

获取剩下的所有的 BeanDefinition
如果不是抽象的, 并且是单例的, 并且不是懒加载的, 就创建

如果是 BeanFactory, 调用 getObject 方法
否则就调用 getBean 方法, getBean 方法会调用 doGetBean 方法 标记点: getBean

先从缓存中获取 Bean, 判断是否有
如果没有, 创建 Bean

获取当前 Bean 依赖的其他 Bean
如果有依赖的 Bean, 调用 getBean 方法先创建依赖的 Bean, 走递归, 递归点: getBean
没有依赖, 就调用 createBean
判断是否需要自定义创建

调用 resolveBeforeInstantiation 方法, 执行 InstantiationAwareBeanPostProcessor 类型的后处理器, 返回自定义对象

不需要自定义就调用, doCreateBean

调用 createBeanInstance, 创建 Bean 实例
调用 populateBean, 给对象属性赋值**(使用三级缓存解决循环依赖)**
调用 Aware 接口的方法
调用后置处理器的 postProcessBeforeInitialization 方法
执行初始化方法
调用后置处理器的 postProcessAfterInitialization 方法**(AOP 在此处完成)**
注册 Bean 的销毁方法

把创建的对象塞进容器

<h3 id="七--Bean的生命周期">七. Bean 的生命周期</h3>

实例化 Bean
设置对象属性
调用 Aware 接口方法
BeanPostProcessor 前置处理方法
Bean 的初始化方法
BeanPostProcessor 后置处理方法
使用中...
销毁

<h3 id="八--循环依赖">八. 循环依赖</h3>
<blockquote>
spring 使用三级缓存解决循环依赖
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">// 一级缓存
<span cla
s="highlight-c1">private <span class="highlig
t-kd">final Map<
/span>String, <span
class="highlight-n">Object> <span class="h
ghlight-n">singletonObjects = <span class="hi
hlight-k">new ConcurrentHashMap<span class
"highlight-o"><>(256<span class="highlig
t-o">);
<span class="high
ight-c1">// 三级缓存 三级缓存value spring扔进去的是一个lambda表达式
<span cla
s="highlight-c1">private <span class="highlig
t-kd">final Map<
/span>String, <span
class="highlight-n">ObjectFactory<?>>
singletonFactories =
new HashMap<span
class="highlight-o"><>(16<span class="hi
hlight-o">);
<span class="high
ight-c1">// 二级缓存
<span cla
s="highlight-c1">private <span class="highlig
t-kd">final Map<

/span>String, Object> earlySingletonObjects = new ConcurrentHashMap<>(16);
</code></pre>
<pre><code class="language-java highlight-chroma">// 扔进三级缓存的lambda表达式
if (earlySingletonExposure) {
    if (logger.isTraceEnabled()) {
        logger.trace("Eagerly caching bean '" + beanName +
                "' to allow for resolving potential circular references");
    }
    // 这里就是扔进去的lambda表达式
    addSingletonFactory(beanName, () -> getEarlyBeanReference(beanName, mbd, bean));
}
</code></pre>
</blockquote>
<blockquote>
使用三级缓存是为了解决 AOP 的循环依赖
如果没有 AOP, 二级缓存就可以解决
下图是大概的过程: 并不完整, 为了方便理解, 少了正常 AOP 的过程
</blockquote>

data-src="https://b3logfile.com/file/2021/08/solo-fetchupload-7217358699861995389-f55948a.png?imageView2/2/interlace/1/format/jpg">
<h3 id="九--spring事务原理">九. spring 事务原理</h3>
spring 通过 AOP 实现动态代理, 通过 ThreadLocal 存取数据库连接, 实现事务
<h3 id="十--事务传播">十. 事务传播</h3>
<table>
<thead>

<tr>
<th align="left">常量名称</th>
<th align="left">常量解释</th>
</tr>
</thead>
<tbody>
<tr>
<td align="left">PROPAGATION_REQUIRED(默认)</td>
<td align="left">如果没有, 就新建事务, 如果有就加入</td>
</tr>
<tr>
<td align="left">PROPAGATION_REQUIRES_NEW</td>
<td align="left">不管外面有没有, 都创建新的事务 (多数据源的时候使用, 下面其他的我没用过)</td>
</tr>
<tr>
<td align="left">PROPAGATION_SUPPORTS</td>
<td align="left">如果有事务, 就加入, 如果没有, 不创建</td>
</tr>
<tr>
<td align="left">PROPAGATION_MANDATORY</td>
<td align="left">必须运行在事务中, 如果没有事务, 就报错</td>
</tr>
<tr>
<td align="left">PROPAGATION_NOT_SUPPORTED</td>
<td align="left">表示该方法不应该运行在事务中, 如果外面有事务, 挂起该事务</td>
</tr>
<tr>
<td align="left">PROPAGATION_NEVER</td>
<td align="left">表示该方法不应该运行在事务中, 如果外面有事务, 直接报错</td>
</tr>
<tr>
<td align="left">PROPAGATION_NESTED</td>
<td align="left">如果没有事务, 开启新事物, 如果有事务, 开启子事务, 事务嵌套, 必须等外面的事务成才能提交 (具体用途我不清楚)</td>
</tr>
</tbody>
</table>
<h3 id="十一--事务失效">十一. 事务失效</h3>

方法不是 public

抛出异常类型错误, 或者被吃掉了

没有开启事务的 A 方法调用了开启事务的 B 方法, 事务会失效
<blockquote>
代理对象内部有 1 个属性是原始对象, 代理对象的 A 方法没有事务,B 方法有事务, 调用过程如下
调用代理对象的 A 方法(没有事务), 代理对象的 A 方法调用原始对象的 A 方法, 原始方法的 A 方内部调用了 B 方法(此时的 B 方法是原始对象的), 所有事务失效
</blockquote>

<h3 id="十二--哪些设计模式">十二. 哪些设计模式</h3>

工厂
<blockquote>
spring 自己就可以算是 1 个大工厂, FactoryBean 也算是生产对象的工厂
</blockquote>

适配器
<blockquote>
AOP 中的 Advice 通知使用了适配器模式
springMVC 的 handler 也使用了适配器模式
</blockquote>

装饰器
<blockquote>
DataSource 使用了装饰器模式
</blockquote>

代理
<blockquote>
AOP 就是动态代理
</blockquote>

观察者
<blockquote>
spring 的事件通知
</blockquote>

策略
<blockquote>
创建对象的时候使用了策略模式
</blockquote>

模板方法
<blockquote>
spring 中的 xxxTemplate 都使用了模板方法
</blockquote>

<h3 id="十三--spring整合mybatis原理">十三. spring 整合 mybatis 原理</h3>

扫描包, 获取全部的接口
注入 1 个 ImportBeanDefinitionRegistrar, 通过 registerBeanDefinitions 方法, 注册 BeanFact
ry 类型的 BeanDefinition
通过 BeanFactory, 然后在 getObject 调用 sqlSession.getMapper(Class clazz)生成代理对象</l

>

<hr>
<blockquote>
springmvc 顺便就和 spring 放在一起了
</blockquote>
<h3 id="十四--springmvc执行流程">十四. springmvc 执行流程</h3>

请求先进入 DispachServlet
通过 URL 去 HandlerMappings 查找对应的 HandlerExecutionChain
基于 HandlerExecutionChain 查找适配器 HandlerAdapter
先调用拦截器 pre 方法
然后再执行 HandlerAdapter(这里执行之后就会调用我们写的 Controller 里的方法)返回 View
象
再调用拦截器 post 方法
视图解析器 ViewResolver 找到对应的视图 View
渲染视图 View, 填充数据
最后调用拦截器 after 方法
返回 View
