Spring 初级面试题

作者：xiaokedamowang
原文链接：https://ld246.com/article/1629201467279
来源网站：链滴

Spring 初级面试题

一. 什么是IOC

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, 从创建到销毁都是架管理

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

二. 什么是AOP

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自带的动代理和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

三. 什么是BeanDefinition

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, 用哪个造方法创建, 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个对象, BeanDfinition就是这种对象实现的接口, 定义了很多获取Bean信息的方法

四. BeanFactory 和 FactoryBean

1. BeanFactory

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, 所以可理解为他就是spring容器, 常用的实现类有: ApplicationContext, DefaultListableBeanFactory, ClassathXmlApplicationContext, AnnotationConfigApplicationContext

2. FactoryBean

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个接口, 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getObject名前面加上$, 创建比较复杂的对象时, 适合用FactoryBean

五. 后置处理器(PostProcessor)

1. BeanFactoryPostProcessor

BeanFactoryPostProcessor是用来增强容器的, 里面定义了方法postProcessBeanFactory方法, 参数ConfigurableListableBeanFactory, 这个类实现了BeanFactory, 传进来的就是spring容器本身, 你可修改里面的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

2. BeanPostProcessor

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

六. spring的创建流程

从new AnnotationConfigApplicationContext()开始

1. 父类无参构造方法创建DefaultListableBeanFactory

2. 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, 创建需要被spring管理的Bean的BeanDefinition

3. **refresh方法被调用**

    1. 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

    2. 先执行BeanDefinitionRegistryPostProcessor, 再执行BeanFactoryPostProcessor(根据优先接口, 或排序接口, 顺序执行)

    3. 创建BeanPostPreocessor

    4. 执行BeanPostProcessor(根据优先级接口, 或排序接口, 顺序执行)

    5. 创建事件派发器, 并且注入到容器

    6. 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

    7. **finishBeanFactoryInitialization方法被调用**, 初始化剩余的Bean

        1. 获取剩下的所有的BeanDefinition

        2. 如果不是抽象的, 并且是单例的, 并且不是懒加载的, 就创建

            1. 如果是BeanFactory, 调用getObject方法

            2. 否则就调用getBean方法, getBean方法 **会调用doGetBean方法** **记点: getBean**

                1. 先从缓存中获取Bean, 判断是否有

                2. 如果没有, 创建Bean

                    1. 获取当前Bean依赖的其他Bean

                    2. 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**

                    3. 没有依赖, 就调用createBean

                    4. 判断是否需要自定义创建

                        1. 调用resolveBeforeInstantiation方法, 执行InstantiationAwareBeanPostProcessor类型的后置处理器, 返回自定义对象

                    5. 不需要自定义就调用, doCreateBean

                        1. 调用createBeanInstance, 创建Bean实例

                        2. 调用populateBean, 给对象属性赋值**(使用三级缓存解决循环依赖)**

                        3. 调用Aware接口的方法

                        4. 调用后置处理器的postProcessBeforeInitialization方法

5. 执行初始化方法
6. 调用后置处理器的postProcessAfterInitialization方法**(AOP在此处完成)**
7. 注册Bean的销毁方法
6. 把创建的对象塞进容器

七. Bean的生命周期

1. 实例化Bean
2. 设置对象属性
3. 调用Aware接口方法
4. BeanPostProcessor前置处理方法
5. Bean的初始化方法
6. BeanPostProcessor后置处理方法
7. 使用中...
8. 销毁

八. 循环依赖

spring使用三级缓存解决循环依赖

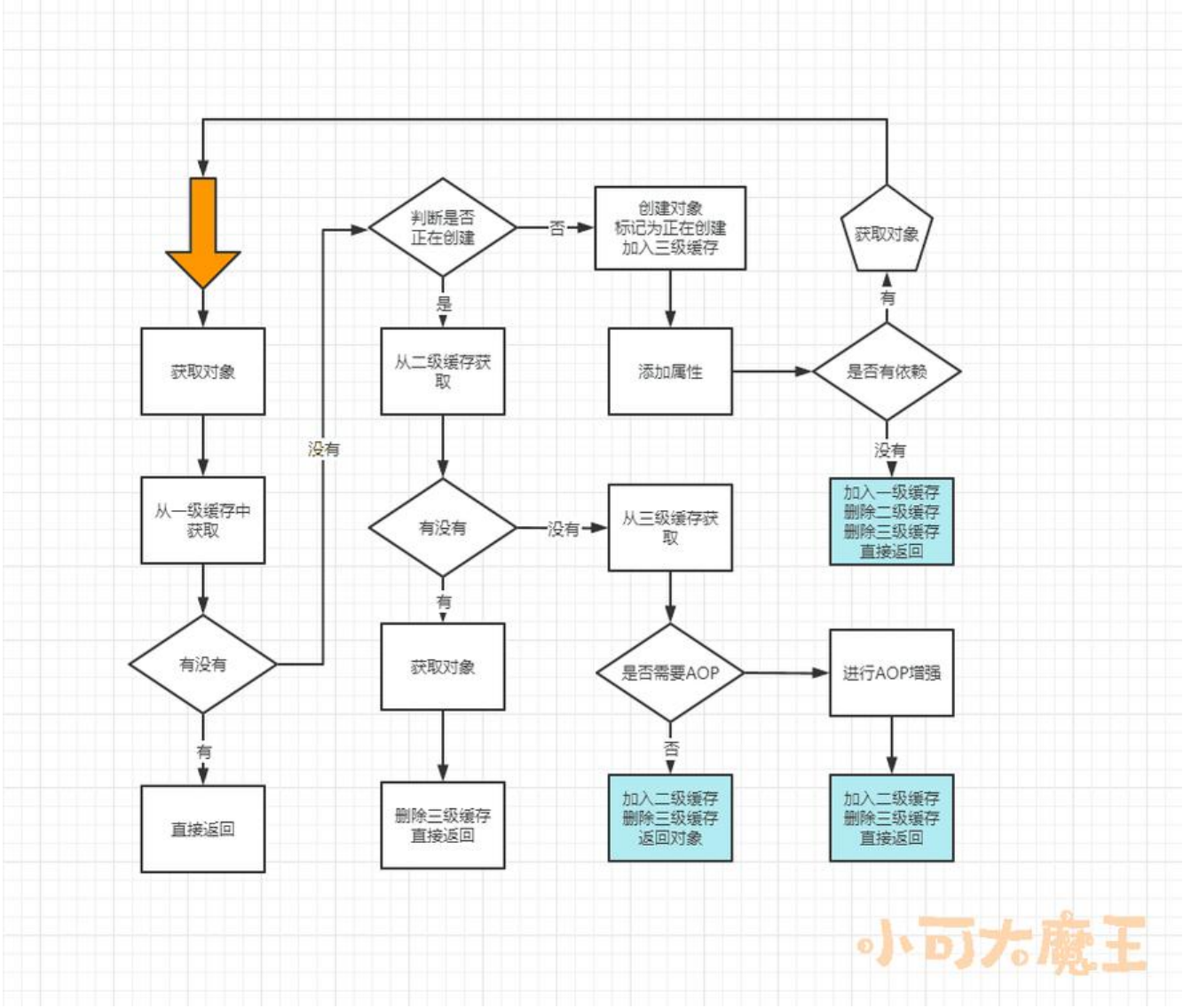
```
// 一级缓存
private final Map<String, Object> singletonObjects = new ConcurrentHashMap<>(256);
// 三级缓存 三级缓存value spring扔进去的是一个lambda表达式
private final Map<String, ObjectFactory<?>> singletonFactories = new HashMap<>(16);
// 二级缓存
private final Map<String, Object> earlySingletonObjects = new ConcurrentHashMap<>(16);

// 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));
}
```

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



九. spring事务原理

spring通过AOP实现动态代理, 通过ThreadLocal存取数据库连接, 实现事务

十. 事务传播

常量名称	常量解释
PROPAGATION_REQUIRED(默认)	如果没有, 就建事务, 如果有就加入
PROPAGATION_REQUIRES_NEW	不管外面有有, 都创建新的事务 (**多数据源的时候使用, 下面其他的我都没用过**)
PROPAGATION_SUPPORTS	如果有事务, 就加入, 如果没有, 不创建
PROPAGATION_MANDATORY	必须运行在事中, 如果没有事务, 就报错
PROPAGATION_NOT_SUPPORTED	表示该方法应该运行在事务中, 如果外面有事务, 挂起该事务

PROPAGATION_NEVER 表示该方法不应该运行事务中, 如果外面有事务, 直接报错

PROPAGATION_NESTED 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, 必须等外面的事务完成才能提交 (**具体用途我不清楚**)

十一. 事务失效

1. 方法不是public
2. 抛出异常类型错误, 或者被吃掉了
3. 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

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方法内部调了B方法(**此时的B方法是原始对象的**), 所有事务失效

十二. 哪些设计模式

- 工厂

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

- 适配器

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

- 装饰器

DataSource使用了装饰器模式

- 代理

AOP就是动态代理

- 观察者

spring的事件通知

- 策略

创建对象的时候使用了策略模式

- 模板方法

spring中的xxxTemplate都使用了模板方法

十三. spring整合mybatis原理

1. 扫描包, 获取全部的接口
2. 注入1个ImportBeanDefinitionRegistrar, 通过registerBeanDefinitions方法, 注册BeanFactory型的BeanDefinition

3. 通过BeanFactory, 然后在getObject调用sqlSession.getMapper(Class clazz)生成代理对象

---

springmvc顺便就和spring放在一起了

十四. springmvc执行流程

1. 请求先进入DispachServlet
2. 通过URL去HandlerMappings查找对应的HandlerExecutionChain
3. 基于HandlerExecutionChain查找适配器HandlerAdapter
4. 先调用拦截器pre方法
5. 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对象
6. 再调用拦截器post方法
7. 视图解析器ViewResolver找到对应的视图View
8. 渲染视图View, 填充数据
9. 最后调用拦截器after方法
10. 返回View