金三银四Java面试突击专题

Spring底层篇

=== 图灵： 楼兰 ===

一、什么是Spring？谈谈你对IOC和AOP的理解。

Spring： 是一个企业级java应用框架，他的作用主要是简化软件的开发以及配置过程，简化项目部署环境。

Spring的有点：

1、Spring低侵入设计，对业务代码的污染非常低。

2、Spring的DI机制将对象之间的关系交由框架处理，减少组件的耦合。

3、Spring提供了AOP技术，支持将一些通用的功能进行集中式管理，从而提供更好的复用。

4、Spring对于主流框架提供了非常好的支持。

IOC就是控制反转，指创建对象的控制权转移给Spring来进行管理。简单来说，就是应用不用去new对象了，而全部交由Spring自动生产。

IOC有三种注入方式：1、 构造器注入 2、setter方法注入 3、根据注解注入。

AOP 面向切面。用于将那些与业务无关，但却对多个对象产生影响的公共行为。抽取并封装成一个可重用的模块。AOP的核心就是动态代理。JDK的动态代理 和CGLIB动态代理。

二、Spring容器的启动流程是怎么样的?

使用AnnotationConfigApplicationContext 来跟踪一下启动流程：

this(); 初始化reader和scanner

scan(basePackages); 使用scanner组件扫描basePackage下的所有对象，将配置类的BeanDefinition注册到容器中。

refresh(); 刷新容器。

prepareRefresh 刷新前的预处理

obtainFreshBeanFactory： 获取在容器初始化时创建的BeanFactory

prepareBeanFactory: BeanFactory的预处理工作，会向容器中添加一些组件。

postProcessBeanFactory： 子类重写该方法，可以实现在BeanFactory创建并预处理完成后做进一步的设置。

invokeBeanFactoryPostProcessors： 在BeanFactory初始化之后执行BeanFactory的后处理器。

registerBeanPostProcessors： 向容器中注册Bean的后处理器，他的主要作用就是干预Spring初始化Bean的流程，完成代理、自动注入、循环依赖等这些功能。

initMessageSource： 初始化messagesource组件，主要用于国际化。

initApplicationEventMulticaster： 初始化事件分发器

onRefresh： 留给子容器，子类重写的方法，在容器刷新的时候可以自定义一些逻辑。

registerListeners: 注册监听器。

finishBeanFactoryInitialization： 完成BeanFactory的初始化，主要作用是初始化所有剩下的单例Bean。

finishRefresh： 完成整个容器的初始化，发布BeanFactory容器刷新完成的事件。

三、Spring框架中Bean的创建过程是怎样的?

首先，简单来说，Spring框架中的Bean经过四个阶段： 实例化 -》 属性赋值 -》 初始化 -》 销毁

然后： 具体来说，Spring中Bean 经过了以下几个步骤：

1、实例化： new xxx(); 两个时机： 1、当客户端向容器申请一个Bean时，2、当容器在初始化一个Bean时发现还需要依赖另一个Bean。 BeanDefinition 对象保存。-到底是new一个对象还是创建一个动态代理?

2、设置对象属性(依赖注入)：Spring通过BeanDefinition找到对象依赖的其他对象，并将这些对象赋予当前对象。

3、处理Aware接口：Spring会检测对象是否实现了xxxAware接口，如果实现了，就会调用对应的方法。

BeanNameAware、BeanClassLoaderAware、BeanFactoryAware、ApplicationContextAware

4、BeanPostProcessor前置处理： 调用BeanPostProcessor的postProcessBeforeInitialization方法

5、InitializingBean： Spring检测对象如果实现了这个接口，就会执行他的afterPropertiesSet()方法，定制初始化逻辑。

6、init-method： 如果Spring发现Bean配置了这个属性，就会调用他的配置方法，执行初始化逻辑。@PostConstruct

7、BeanPostProcessor后置处理： 调用BeanPostProcessor的postProcessAfterInitialization方法

> 到这里，这个Bean的创建过程就完成了， Bean就可以正常使用了。

8、DisposableBean： 当Bean实现了这个接口，在对象销毁前就会调用destory()方法。

9、destroy-method: @PreDestroy

四、Spring框架中的Bean是线程安全的吗?如果线程不安全，要如何处理?

Spring容器本身没有提供Bean的线程安全策略，因此，也可以说Spring容器中的Bean不是线程安全的。

要如何处理线程安全问题，就要分情况来分析。

Spring中的作用域： 1、 singtон 2、prototype： 为每个Bean请求创建给实例。 3、request：为每个request请求创建一个实例，请求完成后失效。 4、 session：与request是类似的。 5、global-session：全局作用域。

对于线程安全问题：

1> 对于prototype作用域，每次都是生成一个新的对象，所以不存在线程安全问题。

2>singtон作用域： 默认就是线程不完全的。但是对于开发中大部分的Bean，其实是无状态的，不需要保证线程安全。所以在平常的MVC开发中，是不会有线程安全问题的。

> 无状态表示这个实例没有属性对象， 不能保存数据，是不变的类。比如： controller、service、dao
>
> 有状态表示示例是有属性对象，可以保存数据，是线程不安全的， 比如pojo.

但是如果要保证线程安全，可以将Bean的作用域改为prototype 比如像 Model View。

另外还可以采用ThreadLocal来解决线程安全问题。ThreadLocal为每个线程保存一个副本变量， 每个线程只操作自己的副本变量。

五、Spring如何处理循环依赖问题?

循环依赖： 多个对象之间存在循环的引用关系，在初始化过程当中，就会出现"先有蛋还是先有鸡"的问题。

一种是使用@Lazy注解： 解决构造方法造成的循环依赖问题

另一种是使用三级缓存

一级缓存：缓存最终的单例池对象： private final Map<String, Object> singletonObjects = new ConcurrentHashMap<>(256);

二级缓存：缓存初始化的对象：private final Map<String, Object> earlySingletonObjects = new ConcurrentHashMap<>(16);

三级缓存：缓存对象的ObjectFactory: private final Map<String, ObjectFactory<?>> singletonFactories = new HashMap<>(16);

对于对象之间的普通引用，二级缓存会保存new出来的不完整对象，这样当单例池中找到不依赖的属性时，就可以先从二级缓存中获取到不完整对象，完成对象创建，在后续的依赖注入过程中，将单例池中对象的引用关系调整完成。

三级缓存：如果引用的对象配置了AOP，那在单例池中最终就会需要注入动态代理对象，而不是原对象。而生成动态代理是要在对象初始化完成之后才开始的。于是Spring增加三级缓存，保存所有对象的动态代理配置信息。在发现有循环依赖时，将这个对象的动态代理信息获取出来，提前进行AOP，生成动态代理。

核心代码就在DefaultSingletonBeanRegistry的getSingleton方法当中。

```
protected Object getSingleton(String beanName, boolean allowEarlyReference)
{
        // Quick check for existing instance without full singleton lock
        Object singletonObject = this.singletonObjects.get(beanName);
        if (singletonObject == null &&
isSingletonCurrentlyInCreation(beanName)) {
            singletonObject = this.earlySingletonObjects.get(beanName);
            if (singletonObject == null && allowEarlyReference) {
                synchronized (this.singletonObjects) {
                    // Consistent creation of early reference within full
singleton lock
                    singletonObject = this.singletonObjects.get(beanName);
                    if (singletonObject == null) {
                        singletonObject =
this.earlySingletonObjects.get(beanName);
                        if (singletonObject == null) {
                            ObjectFactory<?> singletonFactory =
this.singletonFactories.get(beanName);
                            if (singletonFactory != null) {
                                singletonObject =
singletonFactory.getObject();
                                this.earlySingletonObjects.put(beanName,
singletonObject);
                                this.singletonFactories.remove(beanName);
                            }
```

```
                        }
                    }
                }
            }
        }
        return singletonObject;
    }
```

六、Spring如何处理事务?

Spring当中支持编程式事务管理和声明式事务管理两种方式：

1、编程式事务可以使用TransactionTemplate。

2、声明式事务： 是Spring在AOP基础上提供的事务实现机制。他的最大优点就是不需要在业务代码中添加事务管理的代码，只需要在配置文件中做相关的事务规则声明就可以了。但是声明式事务只能针对方法级别，无法控制代码级别的事务管理。

Spring中对事务定义了不同的传播级别： Propagation

1、 PROPAGATION_REQUIRED：默认传播行为。 如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入到事务中。

2、PROPAGATION_SUPPORTS： 如果当前存在事务，就加入到该事务。如果当前不存在事务，就以非事务方式运行。

3、PROPAGATION_MANDATORY： 如果当前存在事务，就加入该事务。如果当前不存在事务，就抛出异常。

4、PROPAGATION_REQUIRES_NEW： 无论当前存不存在事务，都创建新事务进行执行。

5、PROPAGATION_NOT_SUPPORTED： 以非事务方式运行。如果当前存在事务，就将当前事务挂起。

6、PROPAGATION_NEVER ： 以非事务方式运行。如果当前存在事务，就抛出异常。

7、PROPAGATION_NESTED： 如果当前存在事务，则在嵌套事务内执行；如果当前没有事务，则按REQUEIRED属性执行。

Spring中事务的隔离级别：

1、ISOLATION_DEFAULT： 使用数据库默认的事务隔离级别。

2、ISOLATION_READ_UNCOMMITTED： 读未提交。允许事务在执行过程中，读取其他事务未提交的数据。

3、ISOLATION_READ_COMMITTED： 读已提交。允许事务在执行过程中，读取其他事务已经提交的数据。

4、ISOLATION_REPEATABLE_READ： 可重复读。 在同一个事务内，任意时刻的查询结果是一致的。

5、ISOLATION_SERIALIZABLE： 所有事务依次执行。

七、SpringMVC中的控制器是不是单例模式？如果是，如何保证线程安全？

控制器是单例模式。

单例模式下就会有线程安全问题。

Spring中保证线程安全的方法

1、将scop设置成非singleton。 prototype, request。

2、最好的方式是将控制器设计成无状态模式。在控制器中，不要携带数据。但是可以引用无状态的service和dao。